PsychoPy Builder で作る心理学実験

リリース *6.0*

十河宏行

2025 年 04 月 28 日

目次

はじめに

本書は PsychoPy Builder に関する日本語の情報が少なかった 2014 年 1 月頃、PsychoPy 1.79.01 に準拠して執筆を始めました。同年 9 月に第 1 版を公開した後、PscyhoPy のバージョンアップにあわせて改訂を重ねてきましたが、コンポーネントの追加や廃止、出力するコードの仕様変更、JavaScript への対応など、既存の文章に追記するだけでは対応が難しくなってきていました。そこで今回、章立ての変更を含めた、これまでより大きな改訂をおこなうことにしました。主な変更点を以下に挙げます。便宜上構成の変更、書式等の変更、その他の変更としていますが、複数のカテゴリにまたがる内容もあってあまりうまく分類できていません。重要な変更を強調表示してあります。

構成の変更

- 出力された Python のコードの構造を前提とした内容 (旧版第 12 章「実験の流れを制御しよう—強化スケジュール」) を削除した。
- psyexp ファイルを直接編集する内容 (旧版 13.3 節「テキストエディタを用いて多数のコンポーネントを追加しよう」、旧版 13.9.1 節「XML 形式による実験の表現」) を削除した。
- ベンチマークウィザードは事実上放置されているので、ベンチマークウィザードに関する記述を削除した (第 1 章)。
- Button コンポーネントが実用的になったため、Button コンポーネントを活用するようにした。
- Button, Slider, Form コンポーネントは利用場面が多いうえに、Mouse コンポーネントを利用せずにマウスを活用できるため、Mouse コンポーネントより先に解説することにした (Button, Slider, Form コンポーネントは 8 章、Mouse コンポーネントは 9 章)。
- **Button, Slider, Form コンポーネントの解説を「以前の章で作成した実験に組み込む」という構成とした。具体的には、第 6 章の実験に Slider コンポーネントと Form コンポーネント、第 7 章の実験に Button コンポーネントを組み込むこととした。**
- staircase と interleaved staircases(旧版第 11 章) はもともと独立した章とするにはボリュームが少なかったため、廃止して第 3 章のトピックスへ移動した。
- 旧版 9.2 節「クリックした視覚刺激オブジェクトを記録しよう」の内容は新しい第 9 章 (旧版の第 8 章に相当) のトピックスへ移動した。
- 視覚刺激コンポーネントの **[ドラッグ可能]** について新しい 9 章のトピックスで簡単に解説した。
- **新たな第 12 章として「正答率や経過時間で終了する課題を作成しよう」を追加し、廃止された旧版第 12 章で扱っていたループの中断について現在の Builder の仕様に合わせた方法で解説した。また、新たな内容としてルーチンのテンプレート化についての解説もこの章に追加した。**
- **プラグイン/パッケージマネージャに関する解説を附録に追加した。**
- **TextBox コンポーネントや Form コンポーネントで日本語テキストを入力させたい場合の解決方法のひとつとして、JavaScript で出力した実験をローカルブラウザで実行する方法の解説を附録に追加した。**

- 実験記録ファイルから分析に必要な情報を取り出すスクリプトの例を附録に追加した。
- 第 2 章で Pilot モードを導入し、Pilot モードでウィンドウモードを強制する際のウィンドウサイズの変更法を解説した。
- **多重ループによるブロック化の解説に Counterbalance コンポーネントを導入した (第 4 章)。**
- Variable コンポーネントの解説を独自変数の出力についての解説の補足とした (第 7 章)。
- コンポーネントのコピー&ペーストを積極的に活用するように、実験作成手順の解説を見直した。
- プロパティに値が正しく入力されているか確認するために「実験内を検索...」を活用するようにした (第 7 章, 第 11 章)。
- **[位置揃え] を積極的に活用するようにした。これにより第 7 章の錯視図形の作図が簡単になった。**
- 各章の実験作成手順を解説する際に、その実験用のフォルダを作成する指示を追加した。これに伴い、同一のフォルダに複数の psyexp ファイルを作る指示を削除した。
- **実験が正常に終了しなかったときに iohub が問題を生じることがあるため、各章の実験作成手順を解説する際に、入力のバックエンドを Psychtoolbox に設定する指示を追加した (PsychoPy 2025.1.0 では標準で Psychtoolbox になるように変更された)。**

書式等の変更

- 実験作成手順を箇条書きしている部分について、各プロパティに入力する値を地の文と異なる書式 (Sphinx のインラインリテラル) にした。
- 実験作成手順を箇条書きしている部分について、2024.2.5 のウィンドウレイアウトに準拠してプロパティのタブ名を明示するとともに、プロパティ設定ダイアログ等に表示される順序に記述するようにした。具体的には、2024.2.5 では **[サイズ [w, h] $]** と **[位置 [x, y] $]** では **[サイズ [w, h] $]** の方がダイアログ上で先に表示されるので、この両者を編集する場合は **[サイズ [w, h] $]** を先に書くようにした。
- カラーピッカーの出力に合わせて、色の RGB 成分表記は ( ) や [ ] で囲まない形式とした。これに関連して RGB 成分表記および$記号に関する記述を見直した。
- **[位置 [x, y] $]** など初期値がシーケンスであるプロパティについて、最近のバージョンの Builder はタプルに変更されているので本書の記述でもタプルに変更した (図は更新していないので図の中でリストとなっている場合がある)。

その他変更

- 第 1 章のインストール方法を 2024.2.5 に合わせて更新した。特に Windows 版 Standalone インストーラーでインストールした場合の UI の日本語化の問題について解説し、Portable PsychoPy についても言及した。
- 「繰り返し毎に更新」の設定を忘れた時の Builder の挙動が変化したことに対応した。
- expInfo で数値と解釈できる文字列が入力された時の挙動が変化したことに対応した。

- 内部変数 theseKeys の挙動が変化した (ルーチン開始後直後に theseKeys が未定義となる) ことに対応した。
- Polygon コンポーネントで直線を描画する際、 **[サイズ [w, h] $]** の第 2 要素の扱いが変化したことに対応した。
- Microphone コンポーネントでルーチン内に終了が空欄のコンポーネントがあるときのエラーが解消されたことに対応した。
- Movie コンポーネント、Camera コンポーネントの仕様変更に対応した。
- 第 5 版で height に単位を変更した際の修正漏れを修正した。
- 第 5 版で norm 単位に関する記述を削除した際の修正漏れを修正した。
- 教示文などで長すぎて画面に入りきらないものを修正した。
- Image コンポーネントなどでサブフォルダに配置したファイルを参照する際に、プロパティに式を入力してサブフォルダ名を結合するのではなく、条件ファイルに完全な相対パスを書くようにした (オンライン実験化する際に Builder がリソースを検出できる可能性が高まる)。
- key 名を表示するデモ (第 3 章) は Pilot モードで実行するようにした。

改訂にあたって、第 5 章の扱い (順序変更、内容変更、削除) に悩みましたが、この章に手を入れると 6 章以降の記述を全面的に見直す必要があるため、今回はほぼそのまま残す事にしました。2025 年度の授業期間が始まるまでに改訂を終える予定で作業を始めたものの、なかなか作業が進まず予定を 1 ヵ月近く過ぎてしまいました。後半は授業準備の合間の細切れな時間で作業したため、修正できていない見落としがたくさんあるかもしれません。誤字や内容の誤りの指摘、その他内容についての要望などございましたら、十河までご連絡いただけましたら幸いです。

2025 年 4 月 24 日 十河 宏行

第 1 章

PsychoPy の準備

1.1 PsychoPy ってなに？

PsychoPy とは、パーソナルコンピューター (PC) を使って心理学実験を行うためのツールです。心理学実験では刺激画像をぴったり 0.2 秒間提示するとか、刺激音声が提示されてからボタンを押すまでの反応時間をミリ秒の精度で記録するといった作業が求められることがあります。こういった手作業で行うのは困難ですが、PC を使うとかなり正確にコントロールすることができます。また、このような時間的精度が必要ない実験でも、何種類もの刺激を 20 回ずつ無作為な順序で提示するといった作業を、手作業で間違えずにおこなうのは大変ですが、PC はこういった作業が得意です。とはいえ、多くの場合「こんな実験をしたい」と思いついた実験を実際に PC におこなわせようとすると、手持ちのアプリケーションのみではどうにもならず、プログラムを書く必要が出てきます。PsychoPy はこのプログラムの作成と実行を支援してくれるアプリケーションで、簡単な手続きの実験であれば自分でプログラムを書かずに実現することが出来ます。さらに、少しプログラミングを勉強すれば、PsychoPy の標準的な機能では実現できない作業内容だけプログラムを書くといったことが可能です。Microsoft Windows や Apple MacOS など、複数のオペレーティングシステムで動作しますので、自分は Windows を使っているけど共同研究者は MacOS を使用しているといった場合でも、両方の環境で実行できる実験を作成できます。

この章では、PscyhoPy をインストールする手順を解説します。そして、PsychoPy が 3 つのアプリケーションで構成されていることを確認し、それぞれがどのような役割を果たすのかを概説します。

1.2 PsychoPy をインストールしよう (Standalone インストーラー編)

PsychoPy をインストールする方法は複数ありますが、初心者にお勧めなのは「Standalone インストーラー」を用いる方法です。以下の 3 つのいずれにも当てはまらない方 (または意味がわからない方) は Standalone インストーラーでのインストールをお勧めします。

- インストール予定の PC ですでに Python の実行環境を構築していて、そこへ PsychoPy を導入したい。(pip でのインストールがお勧め: 「1.4.3:PsychoPy をインストールしよう (pip 編) 」参照)
- Ubuntu などの Linux 系 OS で使用したい。 (pip でのインストールがお勧め: 「1.4.3:PsychoPy をインストールしよう (pip 編) 」参照)
- 研究室や家庭で他の人と共同で使用している PC で使用したいが、勝手にソフトウェアをインストールできない。(ポータブル版がお勧め: 「1.4.4:PsychoPy をインストールしよう (Portable PsychoPy 編) 」参照))

Standalone インストーラーは PsychoPy の公式 web ページからダウンロードできます。URL は以下の通りです。

- https://www.psychopy.org/download.html

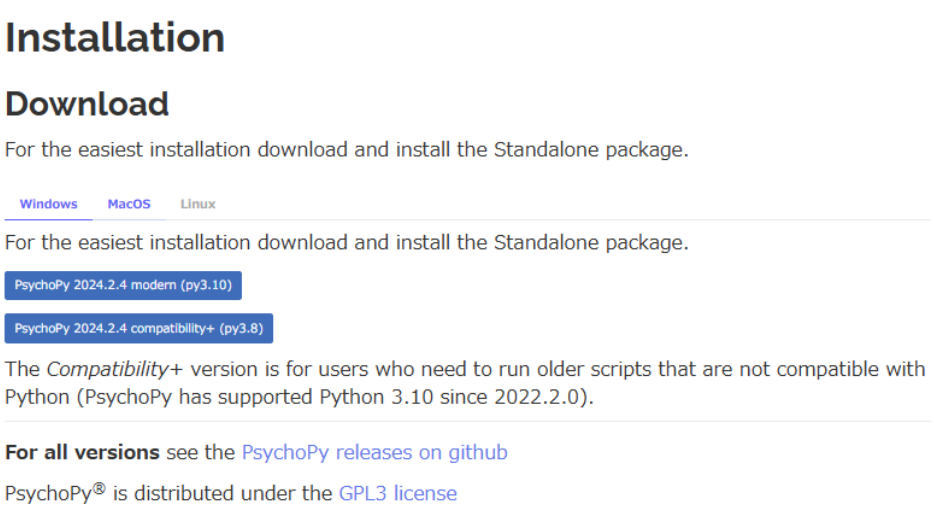


図 1.1　PsychoPy のダウンロード画面。使用している PC のオペレーティングシステムに合わせてお勧めのバージョンをダウンロードするボタンが表示されます。

図 1.1 はダウンロードページを開いた様子です。青色で表示されているボタンをクリックすると、PsychoPy の開発チームがお勧めするバージョンの Standalone インストーラーのダウンロードが始まります。2025 年 2 月現在、Windows 及び MacOS でアクセスすると、図 1.1 のように青いボタンが 2 つ表示されます。これは過去のバージョンの PsychoPy で作成した実験を動かす必要がある人向けの compatibility+ と、その必要がない人向けの modern の 2 種類が公開されているためです。どちらがよいかわからない人は、とりあえず modern を試してみることをお勧めします。研究室の先輩や指導教員が昔作った実験を動かす必要が生じて、 modern ではうまく動かないという場合に改めて compatibility+ を試すとよいでしょう。

なお、これも執筆時点 (2025 年 2 月) での筆者の印象ですが、ここ数年、新機能が導入されてまだ十分に安定性が確認されていないバージョンが「お勧めのバージョン」として表示されてしまう状態が続いています。青いボタンのすぐ下にある For all versions see the PsychoPy releases on github と書かれているリンクをクリックすると、異なるバージョンのインストーラーを入手することが出来ますので、敢えて少し前のバージョンを選択するのもよい選択だと思います (「少し前」の判断については「1.4.1:PsychoPy のバージョン番号の規則とバージョンの選択について 」を参照してください)。「大学の研究室では古いバージョンを使用していて、それと同じものを使用したい」とか「実験に使用する装置が古い Python でなければ動かない」といった事情がある場合も、For all versions see the PsychoPy reelases on github のリンクから目的のバージョンのインストーラーを探すと良いでしょう。

Windows では、ダウンロードしたファイルをダブルクリックするとインストーラーを起動することができます。Windows の保護機能によって 図 1.2 のような警告が出ることがありますが、「詳細情報」をクリックすると「実行」ボタンが出現して実行することができるようになります。

インストールが無事に終了したら、スタート画面を表示してみましょう。PsychoPy3 という項目ができているはずです (図 1.3)。このアイコンをクリックすると PsychoPy が起動します。

MacOS の方は、ダウンロードした dmg ファイルを開くとディスクイメージが開きますので、PsychoPy のア

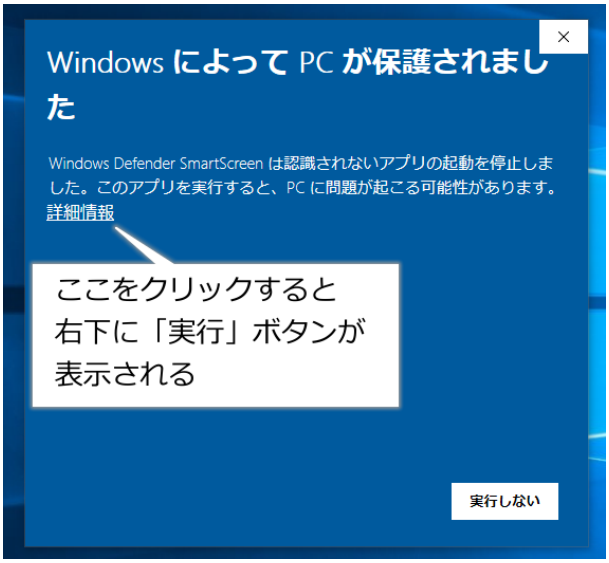


図 1.2　Windows ではインストーラーの実行時に警告が表示されることがあります。

図 1.3　インストールが終了するとスタートメニューの「すべてのアプリ」に PsychoPy3 というグループができています。この中に PsychoPy3 というアイコン (Builder とか Coder とか後ろについていないもの) がありますのでこれを選択して起動します。

イコンをアプリケーションフォルダにドラッグ＆ドロップしてください (図 1.4)。これだけでインストールは終了です。

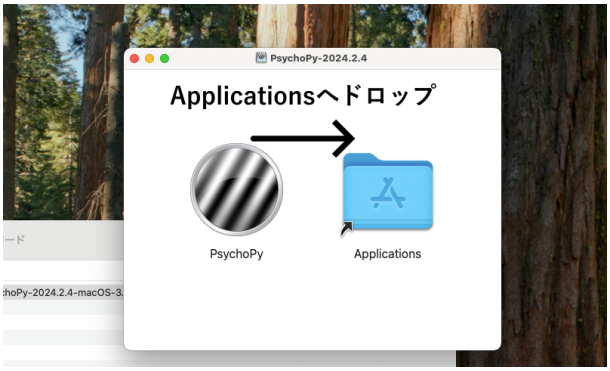


図 1.4　MacOS ではスタンドアローンインストーラーを起動するとこのようなウィンドウが表示されます。PsychoPy のアイコンを Applications のアイコンにドロップするとインストールできます。

さて、PsychoPy を実行すると、図 1.5 に示すウィンドウが画面に出現します。図 1.5 左のように、ウィンドウ内が三分割されていて右側にアイコンが表示されているウィンドウが Builder です。Coder と Runner は

似ていますが、Runner はウィンドウ右端にアイコンが縦に並んでいるのが特徴です。本書では主に Builder と Runner を使用します。なお、スタートメニューに作られたアイコンのうち、PsychoPy Builder、PsychoPy Coder、PsychoPy Runner と書かれたものは、それぞれ Builder、Coder、Runner のウィンドウのみを開いて起動します。後から他のウィンドウを開かなければ行けなくなった場合は、ウィンドウ上部のメニューの「ビュー」から「Builder を開く」「Coder を開く」「Runner を開く」から開くことができます。

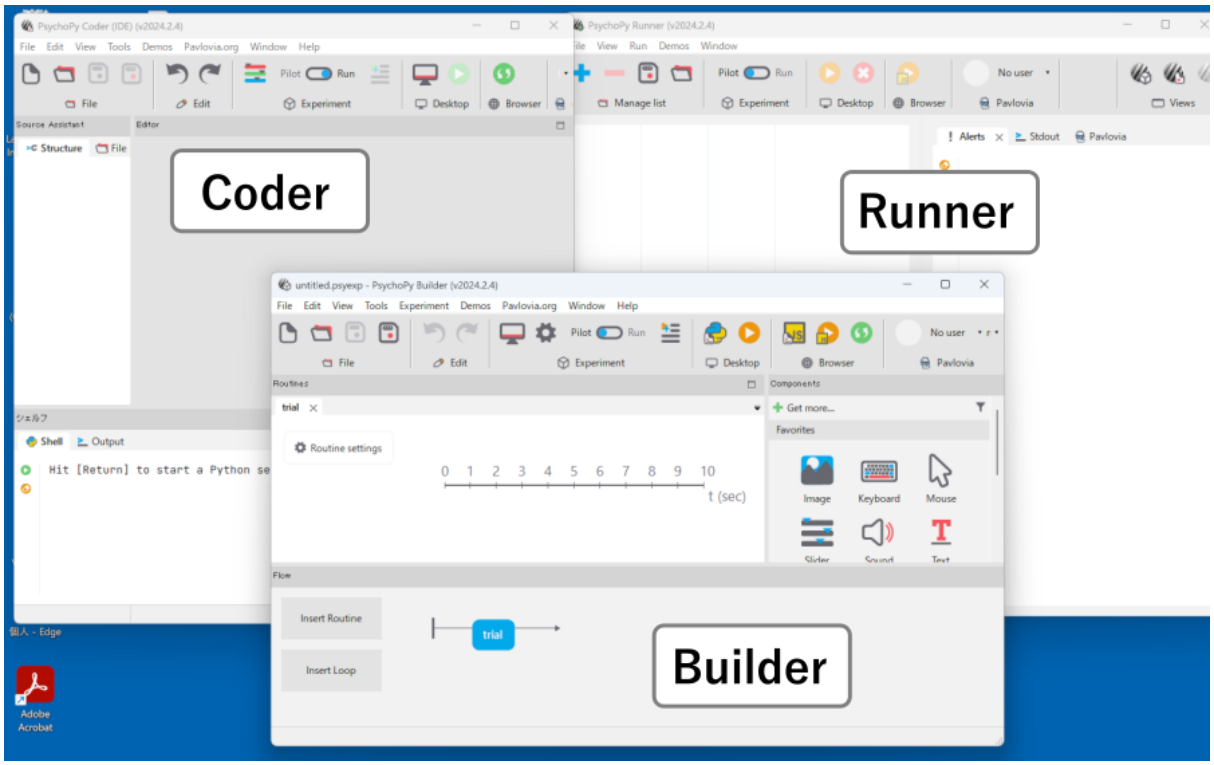


図 1.5　PsychoPy Builder(中央下)、Coder(左)、Runner(右)。Builder はウィンドウ内が三分割されていて、右側にアイコンが並んでいます。Coder と Runner は似ていますが、Runner はウィンドウ右端にアイコンが並んでいるのが特徴です。

使用しているパソコンが日本語環境のものであれば、メニューなどは自動的に日本語で表示されますが、英語で表示されてしまう場合には、ロケールを設定する必要があります。「1.4.2: ロケールの変更 」を見て設定してください。日本語環境のパソコンを使用しているけどあえて英語でメニューなどを表示させたいという場合にもロケールの設定は有効です。なお、 **Windows で PsychoPy 2024.2.5 などのバージョン 2024 系をインストールした場合は、ロケールの設定が正しくても英語で表示されてしまう不具合があります** が、この不具合の修正についても「1.4.2: ロケールの変更 」をご覧ください。

さて、これで PsychoPy のインストールが完了しましたが、解説を進める前に確認しておきたいことが 2 つあります。まず、皆さんが Builder を使って実験を作成する PC には Microsoft Excel (以下 Excel) がインストールされていますか？ Builder では Excel 2007 以降でサポートされた xlsx 形式の Excel ファイルを利用して、実験条件の設定を簡単に行うことができます。このテキストでも 第 3 章 以降で Excel を利用します。

Excel を持っていない場合は、フリーソフトウェアの LibreOffice Calc を利用することもできます。LibreOffice は以下の URL でダウンロードすることができます。

- http://www.libreoffice.org/
- http://ja.libreoffice.org/home/ (日本語)

日本語版の web ページをブラウザで開いた様子を 図 1.6 に示します。「LibreOffice をダウンロード」をクリックしてダウンロードページへ移動し、メインインストールパッケージとヘルプパッケージをダウンロードします。ダウンロードが終了したらメインインストールパッケージ、ヘルプパッケージの順にダウンロードしたインストーラーをダブルクリックしてインストールを済ませてください。LibreOffice Calc は Excel と比べてメニューやツールバーのレイアウトが異なるほか、機能面でもいろいろな違いがありますが、本書で解説している用途の範囲では十分に Excel の代わりとして使えます。

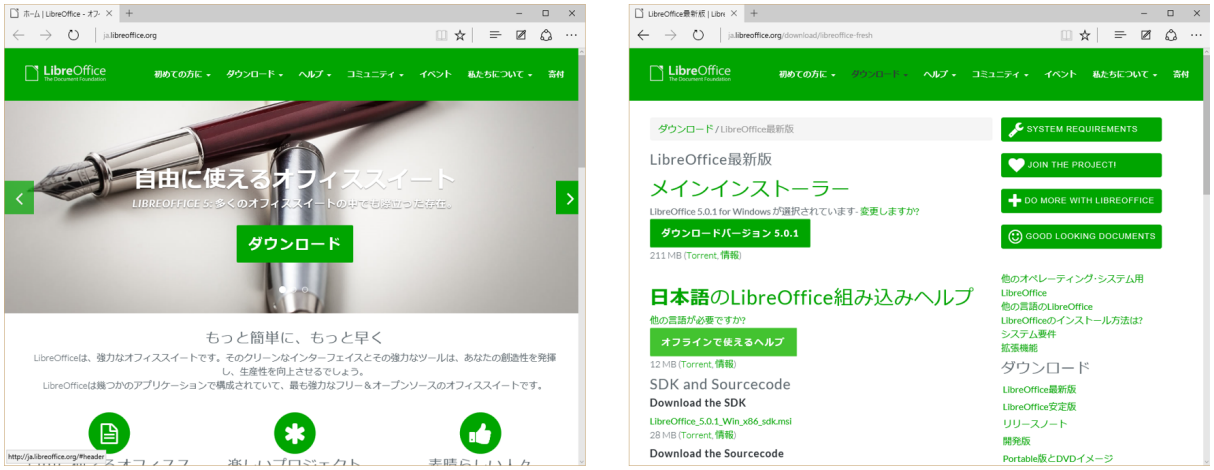


図 1.6　LibreOffice の日本語版 web ページ (左)。「LibreOffice をダウンロード」をクリックするとダウンロードページ (右) へ移動するのでメインインストールパッケージとヘルプパッケージをダウンロードします。

Windows ユーザーの方は、もう一つの確認していただくポイントがあります。Excel のファイルをデスクトップに新規作成してみてください。Excel がインストールされていない場合はテキストドキュメントなどでも構いません。ファイル名の最後に.xlsx(テキストドキュメントなら.txt) という文字列が表示されているでしょうか、それとも表示されていないでしょうか (図 1.7)。このファイル名の最後についている「ピリオド + 英数文字」を **拡張子** と呼びます。拡張子はファイル名の一部で、ファイルの内容を示す記号として用いられます。Windows は拡張子に基づいてファイルを編集するアプリケーションを決定するため、拡張子をうっかり変更してしまうとどのアプリケーションで編集すればいいのかわからなくなってしまいます。このようなトラブルを防ぐために、Windows の標準設定では拡張子を表示されないようになっています。ところが、拡張子が表示されていないと次の章からの解説が非常にわかりにくくなってしまいますので、次の章へ進む前に拡張子が表示されるように設定しておいてください。Windows11 の場合は、エクスプローラーのウィンドウの上部の「表示」メニューから「表示」という項目を選択し、「ファイル名拡張子」という項目にチェックをつけてください。

チェックリスト

- PsychoPy を起動できる。
- Coder や Runner のウィンドウから Builder のウィンドウを開くことができる。
- Excel または LibreOffice Calc のどちらかを起動できる。
- ファイルの拡張子を表示できる。

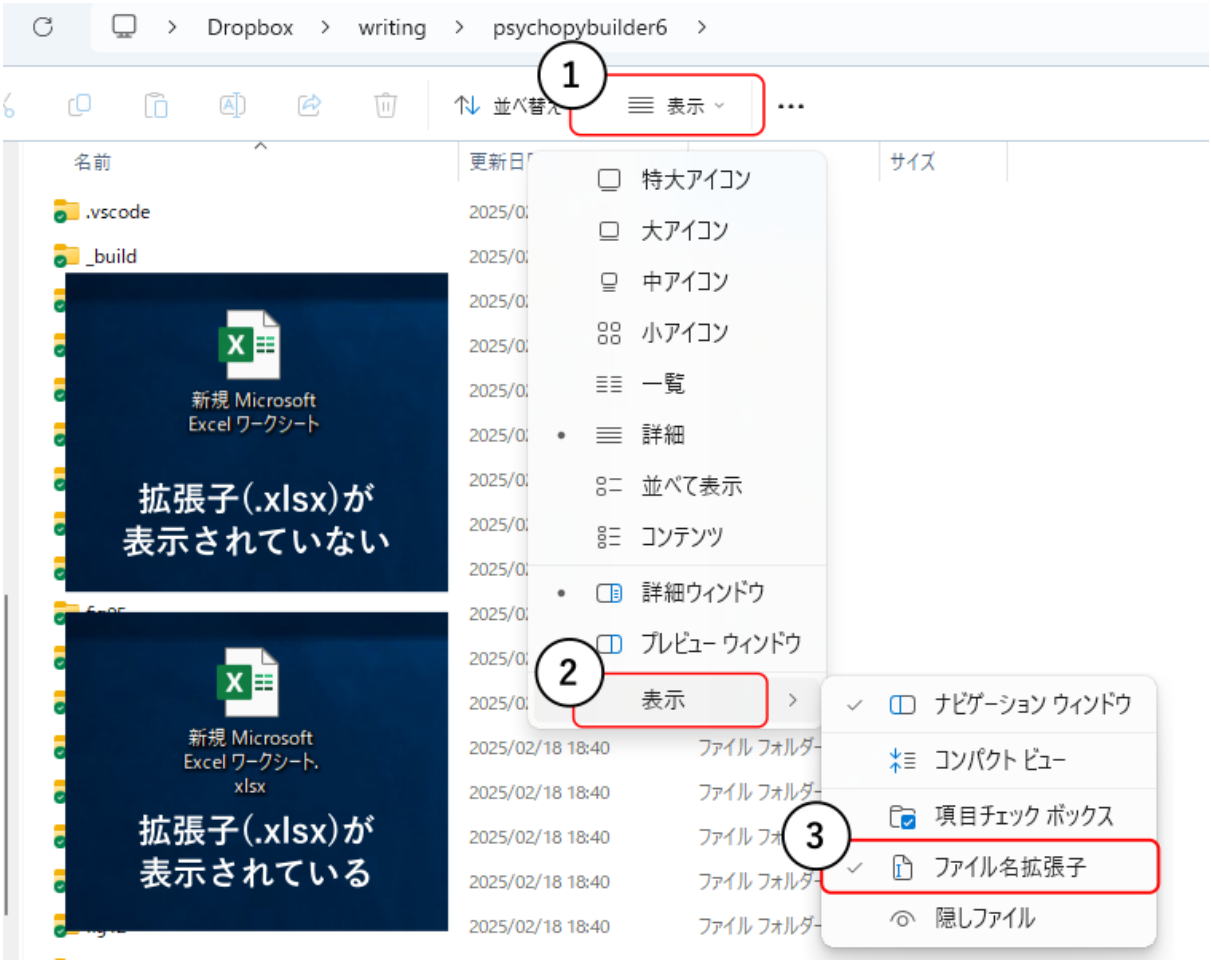


図 1.7　Windows10 で拡張子が表示されていない時の表示方法。

1.3 PsychoPy のしくみ

前節で PsychoPy を起動すると Builder, Coder, Runner という 3 つのウィンドウが開くことを確認しましたが、それぞれの役割を理解するには PsychoPy がどのような仕組みで PC を用いた心理学実験を実現しているのかを知っておく必要があります。

図 1.6 は Builder, Coder, Runner の役割を模式的に示したものです。

PC に何かを行わせるには、その作業手順を記述した **プログラム** を作成する必要がありますが、PC が直接理解することが出来る言語 (機械語) は人間にとって読み書きが非常に困難なので、人間にも読みやすく機械語に翻訳も出来るように設計されたプログラミング言語を使って記述するのが一般的です。皆さんが PC やスマートフォンで使用している多くのアプリケーションは、誰かがプログラムを書いて機械語に翻訳した形で配布されています。一方、心理学実験に限らず、研究場面では人が読み書きできるプログラミング言語で書かれたままのファイルを、使いたいときにその場で機械語に翻訳するという方法がよくとられています。理由はいくつかあるのでしょうが、研究場面では実行した結果を見て内容を修正したりといったことがよくおこなわれるので、いちいち機械語に翻訳したファイルを作成するよりも身軽に作業できる点が大きいと思います。この方式でプログラムの翻訳をおこなうソフトウェアを **インタプリタ** と呼びます。インタプリタで実行することを前提にプログラミング言語で作業内容を記述したファイルを **スクリプト** と呼ぶことがあります。本書では以後このスクリプトという用語を使いますので覚えておいてください。

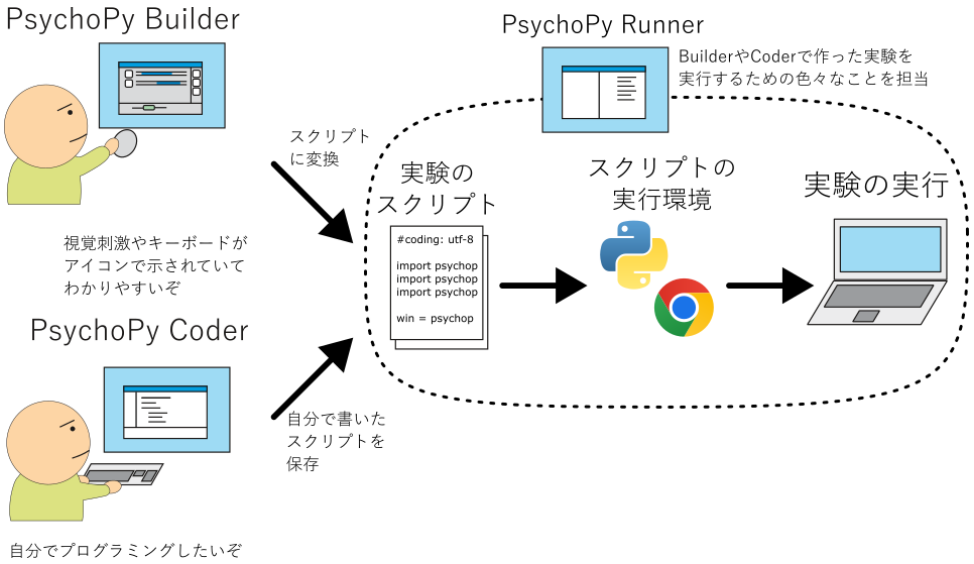


図 1.8　Psychopy Builder,　Coder, Runner の関係。Builder はアイコンなどで表現された視覚刺激や

インタプリタを使ってプログラムを実行するには、「1. スクリプトを書く」、「2. インタプリタを実行できる環境を用意して、スクリプトを実行する」の 2 つの作業が必要ですが、PsychoPy Builder は「スクリプトを書く」という作業を肩代わりしてくれるアプリケーションです。Builder のウィンドウの右側には、視覚刺激やキーボード、マウスなどを示すアイコンがたくさん並んでいて、これらを組み合わせて実験を作成することができます。作成した実験は PsychoPy 独自の形式で保存され、実行するときに自動的にスクリプトに変換することができます。Builder によって変換されたスクリプトを実行する役割を担うのが PsychoPy Runner です。Runner はスクリプトに対応する適切なインタプリタを起動して実験を自動的に実行するほか、エラーメッセージの受け取ったり実験を中断したりといった機能も持っています。Builder と Runner を組み合わせれば、実験の設計から実行まで一連の作業を PsychoPy でカバーすることができます。

それでは残りひとつの Coder は何かというと、プログラミングの知識や経験がある人が、Builder を使わずに自分でスクリプトを書くためのエディタ (テキスト編集アプリケーション) です。プログラミング経験が豊富な人ならすでに自分のお気に入りのエディタを持っていることも多いと思いますが、Coder は Runner と連携してスクリプトを実行する機能を持っていますので、スクリプトは使い慣れた別のエディタで書いて、実行してみたり少し修正したりする場合だけスクリプトを Coder で開くといった使い方が便利なのではないかと思います。Builder で実験を作成するなら基本的には Coder を使う必要はないのですが、Builder を使いこなせるようになってくると「自分が設計したこの実験はどういうスクリプトに変換されるのかな」ということを確認したくなる場合があります。そういった時に Coder を使うとさっとスクリプトを開くことができて便利です。本書では Buidler と Runner の使い方を解説しますので、とりあえず「Builder で実験を作ってスクリプトに変換し、そのスクリプトを Runner で実行する」と覚えておいてください。

以上で PsychoPy を使い始める前に知っておきたい Builder、Coder、Runner の役割についての解説はおしまいですが、最後に Builder が出力するスクリプトが使用しているプログラミング言語について触れておきます。実は、ここまで敢えて具体的なプログラミング言語やインタプリタの名前を避けて解説してきたのですが、これは現在の PsychoPy が 2 種類のプログラミング言語を使用するからです。PsychoPy はもともと Python というプログラミング言語用のパッケージとして開発され、Builder と Runner は Python で書かれたスクリプトを作成し、Python インタプリタを用いて実験を実行していました。しかし 2010 年代後半に Web ブラウザとインターネットを用いて遠隔地にある PC のブラウザ上で実験をおこない、結果をインターネット上のサーバに保存するという **リモート実験** や **オンライン実験** と呼ばれる方式の実験を作成する機能がを PsychoPy に導入

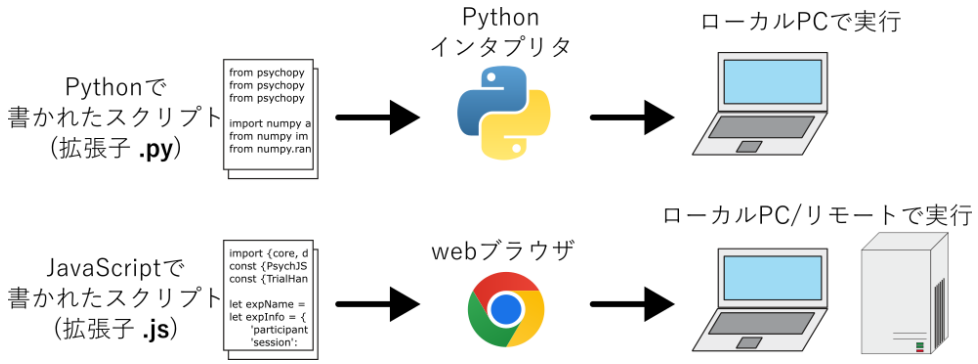


図 1.9　Psychopy Builder,　Coder, Runner の関係。Builder はアイコンなどで表現された視覚刺激やキーボード等のパーツを組み合わせて実験を作成する機能と、作成された実験をスクリプトに変換する機能を持ちます。Runner はスクリプトを実行、中断したりエラーメッセージを受け取って表示したりする機能を持ちます。Coder は自分でスクリプトを書ける人のためのエディタとしての機能を持っています。本書では Builder と Runner を使って実験を作成、実行する方法を中心に解説します。

されました。Web ブラウザ上で実験を実行するには Python よりも JavaScript という言語を使用する方が都合がよいため、Python への変換機能とは別に JavaScript への変換機能が追加されたのです。以上の状況を整理すると 図 1.6 のようになります。以後、本書では、Python スクリプトを出力して目の前にある PC(ローカル PC) で実験を実行する形式を **ローカル実験** 、JavaScript スクリプトを出力して場所を問わずブラウザ上で実験を実行する形式をオンライン実験と表記します。本書は主にローカル実験の作成手順を解説しますが、Builder や Runnner にはオンライン実験のためのメニューや設定項目がありますので、ローカル実験とオンライン実験の違いと、それぞれに Python と JavaScript が対応しているということは頭の片隅に置いておいてください。

それでは、次章から実際に PsychoPy Builder を使用してみましょう。

チェックリスト

- Builder と Runner の役割を説明できる。
- スクリプトという語の意味を説明できる。
- ローカル実験とオンライン実験の違いを説明できる。
- Builder が Python と JavaScript の 2 種類のスクリプト変換に対応している理由を説明できる。

1.4 この章のトピックス

1.4.1 PsychoPy のバージョン番号の規則とバージョンの選択について

多くのオープンソースソフトウェアではセマンティックバージョニング (semantic versioning) と呼ばれる規則に従ってバージョン番号が付けられています。これは **メジャーバージョン** 、 **マイナーバージョン** 、 **パッチバージョン** と呼ばれる 3 つの数値の組み合わせでバージョンを表すもので、例えば 3.10.8 なら 3 がメジャーバージョン、10 がマイナーバージョン、8 がパッチバージョンです。"."は小数点ではなく数値の区切り記号なので、小数のように繰り上がりません。例えば 3.9 の次のマイナーバージョンは 3.10(さん てん じゅう) となります。文脈上、パッチバージョンを省略しても問題ない場合は、3.10 のようにメジャーバージョンとマイナーバージョンのみ書くこともあります。PsychoPy のバージョン番号の規則はセマンティックバージョンとは少し異なるのですが、PsychoPy のベースとなる Python がセマンティックバージョンに従っており知っておいて損はしませんので、もう少し解説しておきます。

セマンティックバージョニングにおいては、メジャーバージョンは過去のバージョンと互換性がない変更が加えられた時に増加します。例えば Python は 2008 年にバージョン 2.6 の次のバージョンとして 2.7 と 3.0 が公開されました。2.7 は 2.6 用に書かれたプログラムを動かすことが出来ますが（これが「過去のバージョンと互換性がある」という意味)、3.0 は 2.6 用のプログラムを動かせない場合があります。メジャーバージョンが異なることによって、利用者に「3.0 では 2.6 用プログラムの動作を保証されていませんよ」とわかるようになっているわけです。これに対してマイナーバージョンは、「新機能が加わったけれども過去のバージョンとの互換性はありますよ」という時に増加します。ですから、再び Python を例に挙げると、2.7 には 2.6 にない新しい機能があります。したがって 2.7 の新機能を使って書いたプログラムは 2.6 で動作しません。しかし先に述べた通り 2.6 用のプログラムは 2.7 で動かすことが出来ます。最後のパッチバージョンは、バグを修正しただけで機能的には何ら変化していない (新機能が追加されたり既存の機能が削られたりしていない) 時に増加します。3.10.0, 3.10.1, 3.10.2,... はいずれもパッチバージョンが異なるだけなので、バグさえなければ原則としてまったく同じ機能を持ちます。

以上、セマンティックバージョニングを踏まえたうえで、PsychoPy のバージョン番号の規則について説明します。PsychoPy も同様に 3 つの数値の組み合わせでバージョンを表しますが、2020 年から最初の数値 (メジャーバージョンにあたる) は公開された年を表すことになりました。2020 年以降、毎年新しいバージョンが公開されているため 2020, 2021, 2022... とバージョン番号が変わっていますが、これらは **過去バージョンとの互換性の有無を表していません。** ですから 2021 で作成した実験を 2022 や 2023 で実行できる **かも知れません** 。いつ公開されたバージョンかわかりやすいのは良いことなのですが、互換性という観点からはわかりにくくなってしまいました。2 番目と 3 番目の数値は、ほぼセマンティックバージョニングのマイナーバージョンとパッチバージョンに対応しています。ですから 2024.1.0 から 2024.2.0 は機能的な変更がありますが、2024.1.0 から 2024.1.1 はバグの修正のみです。

本文で「少し前」のバージョンをどう判断するかという話題がありましたが、例えば現在「お勧め」されているバージョンが 2024.2.4 なら、2024.1 や 2024.0 は少し前と言えるでしょう。一般論として、パッチバージョンは新しいほどバグが修正されているので、パッチバージョンは一番新しいものが良いです。大きな変更が加えられた直後にはバグが多発しやすいので、大きな変更が加えられる直前のバージョンで最もパッチバージョンが新しいものが狙い目です。先述の通り PsychoPy のバージョン番号の規則では 1 つ目の数値が必ずしも大きな変更を表していないのですが、ここ数年の傾向としてやはり 1 つ目の数値が変わるタイミングで大変更が行われることが多いので、一年前のもので最も新しいバージョンを選ぶというのも悪くないでしょう。例えば「お勧め」されているバージョンが 2024.1.2 とすると、2024 年の一年前である 2023 から始まるものを確認します。すると 2023.2.3 が 2023 から始まるもので最も新しいバージョンなので、2023.2.3 を試してみるといった具合です (ちなみに 2025 年 2 月現在「お勧め」されているのは 2024.2.4 ですが、これは恐らく 2024 で最後のバージョンとなると思われるので、2024.2.4 をそのまま試すのもよいでしょう)。

すでにある程度 PC での心理学実験作成の経験があって、きちんとバージョン間の違いを理解して選択したい場合は、Changelog (https://www.psychopy.org/changelog.html) を読んでみてください。

1.4.2 ロケールの変更

Builder のメニュー等の表示に使用される言語を変更したい場合は、ロケールを設定してください。Coder でも Builder でも、ウィンドウの上部に 図 1.10 に示す工具のアイコンのボタンがあります。これをクリックすると PsychoPy の設定ダイアログが開きます。「アプリケーション」というページに「ロケール」という項目があります。

初期状態ではロケールは空白となっています。この場合、PsychoPy は使用中のパソコンの言語設定を利用し

て表示に用いる言語を決定しようとします。PsychoPy によって選ばれた言語と異なる言語で表示させたい場合は、ロケールをクリックして表示される言語の一覧から希望のものを選んでください。

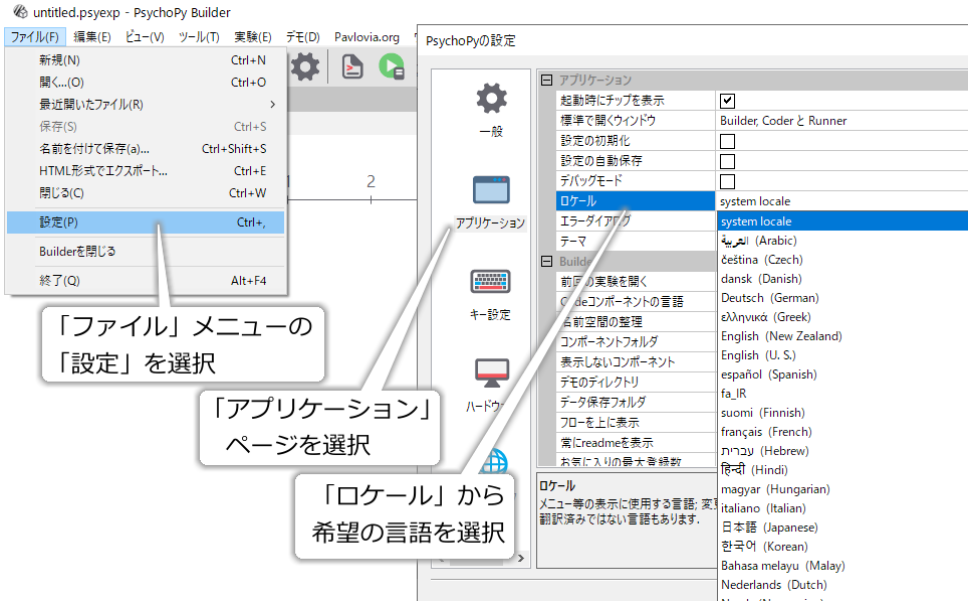


図 1.10　ロケールの設定。

なお、 **PsychoPy 2024 系の Windows 用スタンドアローンインストーラーには英語以外の言語用の翻訳ファイルが含まれていないというバグ** があります。筆者の Web サイトに修正スクリプトを下記 URL にアップロードしてありますのでご利用ください。

- http://s12600.net/psy/etc/python.html#fixpsychopy2024translation

筆者が配布している Portable PsychoPy 2024.2.5 はこの問題に対処済みですので、ポータブル版でよければこちらを利用していただくという選択肢もあります。「1.4.4:PsychoPy をインストールしよう (Portable PsychoPy 編)」を参照してください。

1.4.3 PsychoPy をインストールしよう (pip 編)

Windows や MacOS のスタンドアローンパッケージを使用しない場合は、Python のパッケージマネージャである pip を使用して PsychoPy をインストールするのが簡単です。Anaconda/Miniconda を使う方法や、Ubuntu 等の Linux ディストリビューションにパッケージとして用意されているものをインストールする方法もありますが、本書ではお勧めしません。2025 年 2 月現在、Python のバージョンは 3.10 が推奨されています。3.12 や 3.13 などの新しいバージョンでは PsychoPy が動作しない可能性がありますので (これらのバージョンに対応する依存パッケージが配布されていないなど)、場合によってはそれらの新しいバージョンの Python とは別に 3.10 の実行環境を用意しないといけないかもしれません。

以下、 `python` というコマンドで Python3.10 のインタプリタを起動できるという前提で話を進めます。読者の皆さんの環境によっては `py` , `py -3.10` `python3` とかかも知れませんので読み替えてください。まず、PsychoPy 以外の用途にも Python を使用している方は、PsychoPy 用の環境を用意しておくとパッケージの依存関係などで悩まずに済むので便利でしょう。以下は Windows でユーザープロファイルフォルダ (通常、C:Users ログインユーザー名) に venv_psychopy という名前のフォルダを作ってそこに PsychoPy 用の仮想環境を作り、仮想環境を有効にする例です。

```
python -m venv %USERPROFILE%/venv_psychopy
call %USERPROFILE%\venv_psychopy\Scripts\activate
```

MacOS や Linux なら以下のように. から始まるフォルダ名にして隠しフォルダにしておくと邪魔にならなくてよいかも知れません。この辺りは好みでどうぞ。

```
python -m venv ~/.venv_psychopy
source ~/.venv_psychopy/Scripts/activate
```

仮想環境が有効になった状態で pip を使って PsychoPy をインストールします。

```
python -m pip install psychopy
```

インストール完了後に PsychoPy を起動するには、以下のように psychopy.app.psychopyApp をスクリプトとして実行します。もし ModuleNotFoundError: No module named 'psychopy' と表示されて PsychoPy が起動しないのなら、おそらく pip によるインストールが失敗しています。失敗する理由はさまざまですが、筆者の個人的な経験では、PsychoPy が依存しているパッケージのビルドに必要な開発環境が整っていない場合が多いです。エラーメッセージを注意深く読み、足りないものを判断してインストールしてから、もう一度 pip での PsychoPy のインストールを試みてください。

```
python -m psychopy.app.psychopyApp
```

PsychoPy を終了した後、仮想環境を抜ける時には `deactivate` を実行してください。dactivate せずにシェルを閉じても問題ありません。Windows でアイコンをダブルクリックするだけで起動できるようにしたい場合は、以下のようなバッチファイルを作成すると良いでしょう。最後の deactivate は少なくとも現状の deactivate の処理内容では不要ですが、今後どうなるかわかりませんので実行しておいた方がよいでしょう。

```
call %USERPROFILE%\venv_psychopy\Scripts\activate
python -m psychopy.app.psychopyApp
deactivate
```

なお、2025 年 2 月現在、PsychoPy の公式ページで Linux 系 OS でのインストール方法として、仮想環境に入った後以下のようにリポジトリ内の installPsychoPy.py というスクリプトを実行する方法が紹介されれています。これは PsychoPy の依存パッケージの build に必要なパッケージなどのインストールをしたうえで pip で PsychoPy をインストールするものです。EXPERIMENTAL(実験的) ということで皆さんの環境で動作するとは限りませんが、Linux 系 OS へインストールしようとしている方は試してみるのもよいと思います。

```
python -c "$(curl -fsSL https://raw.githubusercontent.com/psychopy/psychopy/dev/
↪installPsychoPy.py)"
```

1.4.4 PsychoPy をインストールしよう (Portable PsychoPy 編)

Portable PsychoPy は、USB などにインストールして持ち運べる Windows 用 Python 実行環境である WinPython に PsychoPy をインストールしたものです。筆者が個人的に作成して公開しており、PsychoPy 公式のものではありません。大学の情報センターなどの PC のようにユーザーが勝手にアプリケーションをインストールできない環境で PsychoPy を使用したい場合や、研究室などで複数台の PC を管理していてそれぞれに PsychoPy をインストールして管理するのが大変な場合などに便利です。配布 URL は以下の通りです。

- http://s12600.net/psy/etc/python.html#portable

ダウンロードしたファイルは Zip 形式で圧縮されていますので、USB メモリなどに展開してください。展開したファイルの中に PortablePsychopyLauncher.exe という実行ファイルがあり、これをダブルクリックすると PsychoPy が起動します。

詳しい使い方は上記配布ページを見ていただきたいのですが、ひとつだけ注意しておきますと、この Portable PsychoPy は USB メモリに入れて持ち運ぶことができますが、PsychoPy 実行時に自動的に作成される設定ファイルは PC 内のフォルダ (%APPDATA%psychopy3) に作成されます。ですから、初めて Portable PsychoPy を使用する PC で起動したときには初期設定の状態になりますし、ある PC 上で設定変更を行っても別の PC にその設定は持ち越されません。一応、Portable PsychoPy には「終了時に PC から USB メモリ上に設定ファイルをコピーする」、「起動時に USB メモリから設定ファイルを PC へコピーする」という機能をオプションで用意していますが、この機能を使う場合は PsychoPy を終了した後、コピーが終わるまで USB メモリを抜かないようにしてください。やはり Portable PsychoPy のオプションで、コピーを含めて正常に PsychoPy が終了した時にメッセージを表示する機能を用意していますので、この機能を有効にしておけばコピーが終了したのを確認することができます。

第 2 章

刺激の位置や提示時間の指定方法を覚えよう

2.1 まずは表示してみよう

本章では、Builder を使って画面に視覚刺激を描画する際に知っておくべき位置や大きさ、色などの指定方法を学びます。最初の一歩として、三角形と長方形を PC の画面に表示してみましょう。Builder を起動するとウィンドウ内が三つに分割されています。この分割されたひとつひとつの部分をペイン (pane: 窓枠の一区画のこと) と呼びます。左のペインをルーチンペイン、右のペインをコンポーネントペイン、下のペインをフローペインと呼びます (図 2.1)。コンポーネントとは実験を作るための部品のようなもので、コンポーネントペインには Builder で使用できるコンポーネントを表すアイコンが並んでいます。ここから刺激を表示するためのコンポーネントなどを選んでルーチンペインに配置し、フローペインで実験の流れを指定するという手順で実験を作成します。

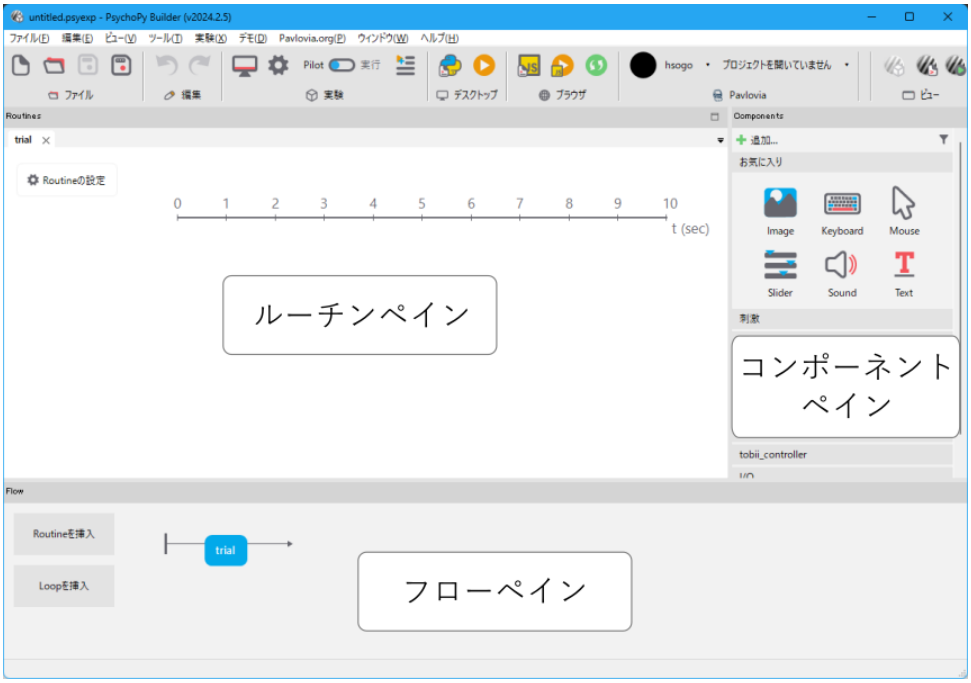


図 2.1　Builder の 3 つのペイン。コンポーネントペインから刺激等を選んでルーチンペインに配置し、フローペインでルーチンの実行順序を指定します。

この章では、コンポーネントペインとルーチンペインの使い方を覚えましょう。まず、コンポーネントペインの「刺激」と書いてある部分を何度かクリックしてみてください。コンポーネントのアイコンが現れたり消えたりするはずです。コンポーネントはよく使う「お気に入り」、刺激描画に使う「刺激」、反応計測に使う「反応」、高度な処理を行うための「カスタム」、脳波測定のための「EEG」、視線測定のための「アイトラッキング」、外部機器との入出力に使う「I/O」のカテゴリに分類されており、それぞれカテゴリ名をクリックするとそのカテゴリに含まれるコンポーネントのアイコンが表示されたり隠されたりします。

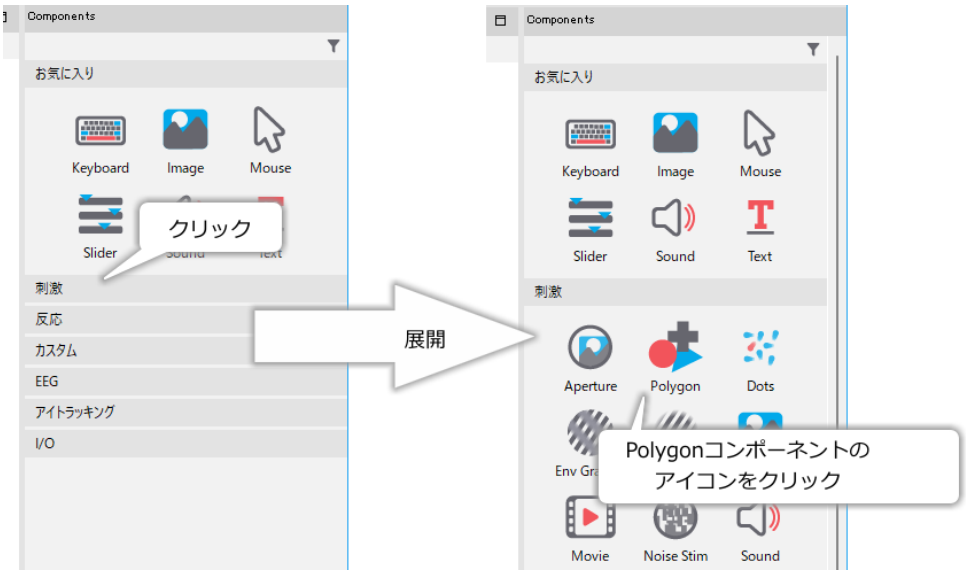


図 2.2　「刺激」カテゴリを開いて Polygon コンポーネントのアイコン (円、三角、十字が描かれたアイコン) をクリックします。

それでは「刺激」カテゴリに含まれている Polygon コンポーネントを使って実際に刺激を PC の画面に表示してみましょう。Polygon コンポーネントはの右下に示されている楕円、三角、十字が描かれたアイコンです (図 2.2)。このアイコンをクリックすると、描画する刺激の大きさや色といった特性（プロパティと呼びます）を設定するダイアログが表示されます (図 2.3)。ダイアログの左上の辺りに「基本」、「レイアウト」、「外観」、…と書かれている点に注目してください。これは「タブ」と呼ばれて、タブをクリックすることによって表示される内容が切り替わります。タブをクリックして表示内容を変更することを「ページを切り替える」などということもあります。

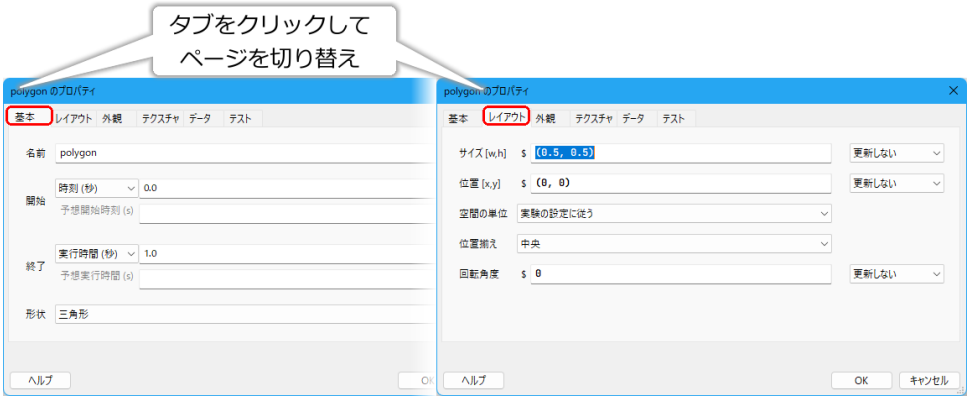


図 2.3　Polygon コンポーネントのプロパティを設定するダイアログ。左上のタブをクリックすることでページを切り替えられます。

まず一切プロパティを変更せずに右下の OK をクリックしてみましょう。すると、ルーチンペインに polygon

コンポーネントのアイコンが表示され、その右側に青い棒が表示されます。これで「コンポーネントをルーチンに配置する」という作業ができました。Builder では、このようにルーチンにコンポーネントを配置していくことによって刺激を作成します。

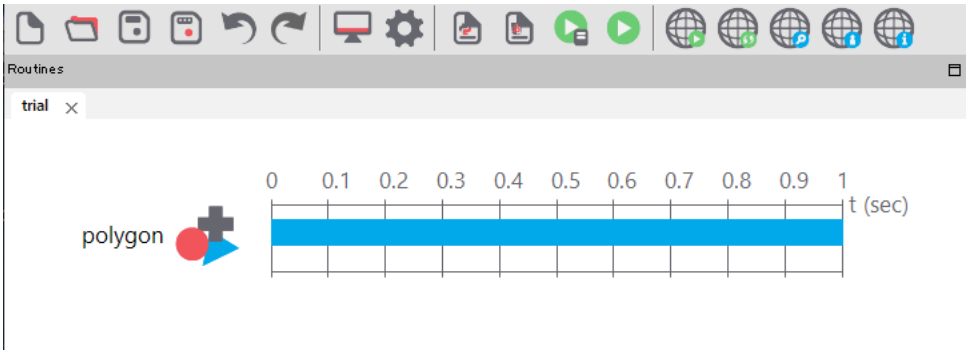


図 2.4　コンポーネントをルーチンに配置した状態。

では、この Polygon コンポーネントがひとつ配置されただけのシンプルな「実験」を実行してみましょう。「 1.3:PsychoPy のしくみ 」で述べた通り、実験を実行するのは Builder ではなく Runner の役割ですが、実際の実験の作成時には「Builder で少し作業して Runner で動作を確認する」という作業を繰り返すので、Builder から直接 Runner を呼び出せるようになっています。Builder のウィンドウ上部に「ファイル (F)」、「編集 (E)」といった項目が並んでいる部分を **メニュー** 、そのすぐ下にアイコンが横に並んでいる部分を **リボン** と呼びますが、リボンの中に「実験」、「デスクトップ」と書かれている区画があります。PsychoPy の初期設定では、図 2.5 上段のように、「実験」の区画の「Pilot」、「実行」と書かれているスイッチが「Pilot」側になっています。本書では以後このスイッチを **Run モードスイッチ** と呼びます。Run モードスイッチの右側、「デスクトップ」の区画にオーディオアプリの再生ボタンのような、オレンジ色の円の中に三角形がボタンがあります。以後このボタンを **Run ボタン** と呼びます。Run モードスイッチが Pilot だと Run ボタンはオレンジ色ですが、Run モードスイッチを実行にすると 図 2.5 下段のように Run ボタンが緑色になることを確認してください。これらの Run モードの違いを 表 2.1 に示します。実験を作っている最中にあれこれ試行錯誤するときは Pilot モード、実験が問題なく動作する自信がある場合や実験本番の時は実行モードを選択します。

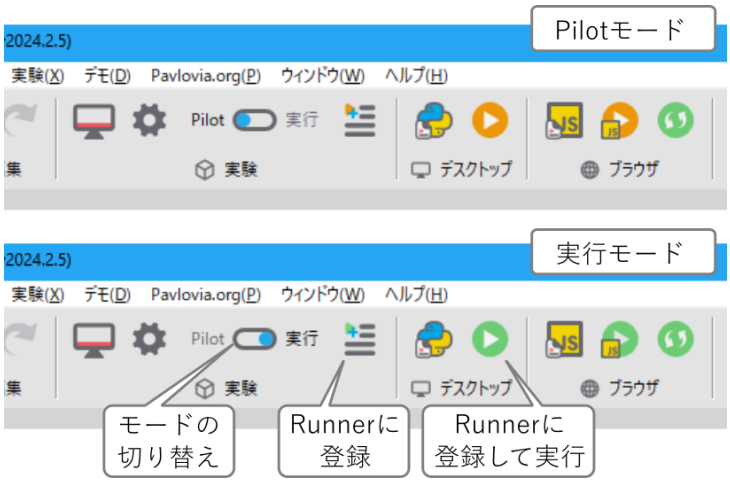


図 2.5　Run モードスイッチと Run ボタン。上段は Run モードスイッチが Pilot になっている状態、下段は Run モードスイッチが実行になっている状態を示しています。Run ボタンの色が Pilot モードではオレンジ色、実行モードでは緑色となる点に注意してください。

表 2.1　Run モード

単位	説明
Pilot	実験作成中に動作確認のために実行するときに便利なモード。PsychoPy の初期設定では、実験画面がモニター中央にオレンジ色の枠付きで小さく表示されるほか、Runner に実験の動作に関する情報が詳しく表示される。実験実行中に Runner のウィンドウを操作することができるので、実験の動作に問題があったときに Runner から中断することが容易である。
実行	実験本番向けのモード。一般的には実験画面がモニター全体に大きく表示され (**フルスクリーン** モードと呼ぶ)、画面描画や音声再生のタイミングができる限り正確になるように動作する。その代わりに、実験実行中に Runner のウィンドウから中断する操作ができなくなる。

今回は初めての実行ですので、Builder から Runner への連携がわかりやすいようにステップを踏んでいきましょう。Run モードスイッチの右側の、 表 2.1 で「Runner に登録」と示されているボタンをクリックしてください。このボタンは、Builder から Runner への引継ぎだけをおこなって実行まではおこないません。クリックすると、ここまで実験を一度も保存していないので、まず実験保存ダイアログが表示されます。ここでは chapter2.psyexp という名前で保存しておきます。拡張子が.psyexp のファイルには、Builder で作成した実験の情報が保存されています。以後、このファイルのことを **psyexp ファイル** と呼びます。psyexp ファイルの他にも Builder が作成するファイルがありますが、それらについては「 2.8:Builder が作成するファイルを確認しよう 」で解説します。

実験を保存すると、自動的に Runner のウィンドウ (図 2.6) が表示されます。Runner のウィンドウは大きく左右に分割されていて、左側には Builder から引き継いだ実験が登録されます (Coder や他のエディタを使って自分で作成したスクリプトも登録できますがここでは省略します)。先ほど保存した chapter2.psyexp が登録されていることを確認してください。左側は実験が適切に動作しているかを知るための様々な情報 (**ログ** と呼びます) が表示されます。ウィンドウ上部のリボンには、Builder にもあった Run モードスイッチや Run ボタンがありますが、注目していただきたいのは Run ボタンの隣のボタンです。これは実験実行中に何か問題が生じた時に強制的に終了させるためのボタン (中断ボタン) です。それでは、Runner ウィンドウ左側の実験一覧に登録されている chapter2.psyexp を選択 (マウスでクリックすれば選択できます) した状態で Run ボタンをクリックしてください。

Run ボタンをクリックしてしばらく待つと、 図 2.7 のような小さなダイアログが表示されます。これを **実験情報ダイアログ** と呼びます。少し古めの PC やノート PC を電源につながず実行している場合などには実験ダイアログが表示されるまで少し時間がかかりますのでご注意ください。同時に、Runner の中断ボタンが有効になります。中断ボタンをクリックすると実行が中断されます。

実験情報ダイアログは、実験参加者の氏名や実験の条件など、実験の実行に必要な情報や、データと一緒に保存しておきたい情報などを入力するためのものです。どのような情報を入力できるようにするかはもちろん設定できるのですが、ここではまだ何も設定していないので標準で設定されている session と participant という項目が表示されています。この章では実験情報ダイアログを利用するところまで進みませんので、とりあえず変更せずに OK をクリックしてください。すると 図 2.8 左のように画面中央に灰色の長方形が表示され、その中に白い文字で Attempting to measure frame rate of screen, please wait... と表示されます。数秒後、 図 2.8 右のように画面中央に白い三角形が 1 秒間表示されます。時間が短くてわかりにくいですが、灰色の領域の縁にオレンジ色の枠線がつき、左下に PILOTING: Switch to run mode before testing. と書かれていることにも注

図 2.6　Runner のウィンドウ。左側は登録済みの実験のリスト、右側は実験実行時のログが表示されます。ウィンドウ上部のリボンには Builder と同じ Run モードスイッチ、Run ボタンに加えて中断ボタンがあります。

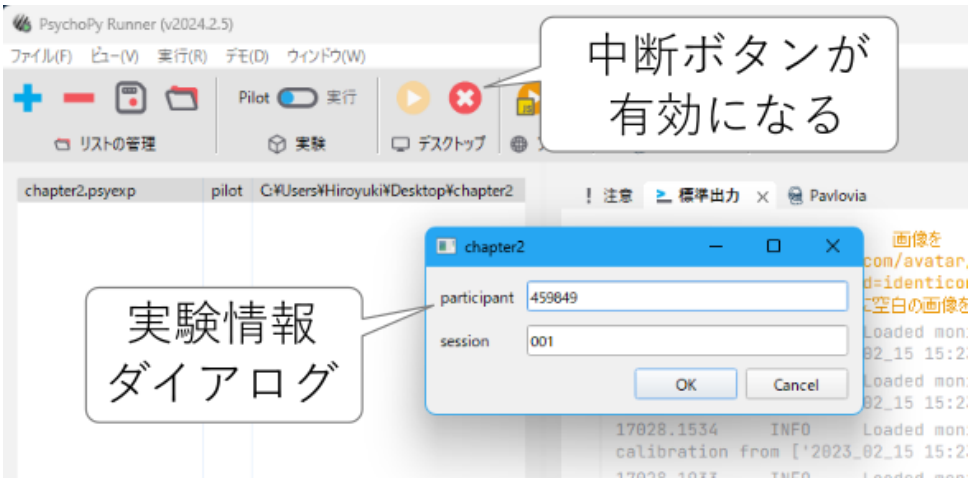


図 2.7　実行ボタンをクリックした後、少し待つと実験情報ダイアログが表示されます。同時に実験中断ボタンが有効になります。

目してください。三角形が出現してから 1 秒後に、元の Runner のウィンドウが表示されている画面に戻ります。なお、環境によっては白い三角形が表示される瞬間に、Runner のウィンドウが手前に表示されてしまって白い三角形が描画されたのがわからないことがあります (筆者が動作確認した限り、「2.9.6:「入力」タブ」で触れる **[キーボードバックエンド]** で ioHub を選択していると起こりやすい)。その場合は Runner のウィンドウを左右にすこし避けておいてもう一度 Run ボタンをクリックして実行してみてください。

Pilot モードできちんと動作することが確認できたら、Runner の Run モードスイッチを実行に切り替えて (Run ボタンが緑色になるのを確認してください)、Run ボタンをクリックして実行してみましょう。同じように実験情報ダイアログが表示され、OK ボタンをクリックしたら今度は画面全体が灰色になり、 Attempting to measure frame rate of screen, please wait... のメッセージの後に白い三角形が表示されるはずです。これがフルスクリーンモードです。フルスクリーンモード担っている間は Runner のウィンドウが見えないため、Runner の中断ボタンをクリックできないということがおわかりいただけるかと思います。あとは、Builder のウィンドウに戻って、Builder の Run ボタンからも同様に実験を実行できることを確認しておいてください。これで私たちが作った「最初の実験」が無事に終了しました。

チェックリスト

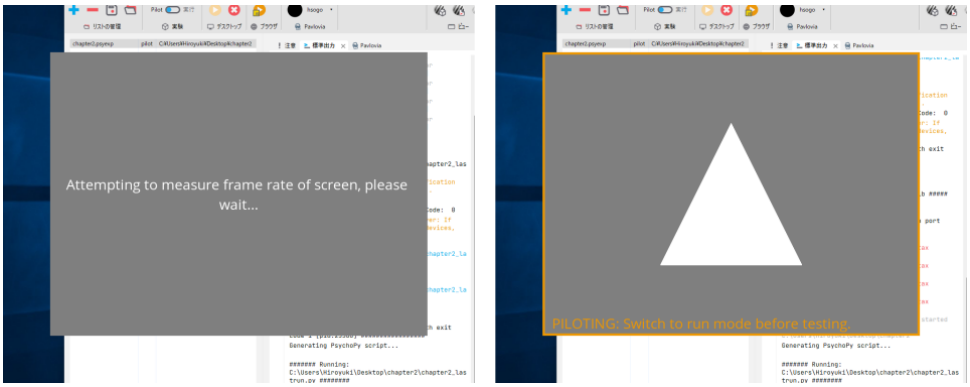


図 2.8　実験情報ダイアログの OK ボタンをクリックすると、左図のように灰色の領域が表示されその中に白いメッセージが表示されます。数秒後、右図のように灰色の領域の中央に白い三角形が表示されます。

- ルーチンペインにコンポーネントを配置できる。
- Pilot モードと実行モードを切り替えできる。Run ボタンの色の違いから現在どちらのモードなのか判断できる。
- Builder の実験ファイルの拡張子を答えられる。
- 作成した実験を実行できる。

2.2 コンポーネントの設定変更と削除の操作を覚えよう

続いて、コンポーネントの設定を変更してみましょう。Builder のウィンドウに戻って、ルーチンペインに表示されているコンポーネントのアイコンにマウスカーソルを動かして左ボタンをクリックしてみてください。すると先ほどコンポーネントを配置した時に表示されたプロパティ設定ダイアログが再び表示されます。ここで 図 2.9 のようにダイアログの「基本」タブにある「形状」を長方形に変更し、「外観」タブの「塗りつぶしの色」という項目に black と入力して、ダイアログ右下の OK をクリックしてみましょう (black と入力する時には **日本語入力を OFF** にしてください)。これで形状が長方形、刺激の塗りつぶし色が黒色に設定されました。もう一度実験を実行すると、今度はのように黒色に塗りつぶされた正方形が表示されるはずです。

図 2.9　「形状」を長方形、「塗りつぶしの色」を black に設定して実行してみましょう。

このように、配置済みコンポーネントのプロパティは何度も設定し直すことができます。この操作は今後繰り返しおこなうことになるので覚えておきましょう。本書ではこれ以後、プロパティ名を **[塗りつぶしの色]** の

ように [ ] で囲って表記します。

次の話題に進む前にもうひとつ覚えておきたい基本操作は、ルーチンペインに設置したコンポーネントの削除です。ルーチンペインに配置された Polygon コンポーネントの上にマウスカーソルを動かして今度は右クリックしてみてください。図 2.10 のようにポップアップメニューが表示されますので、「削除」を選択してください。コンポーネントがルーチンペインから取り除かれます。「削除」の項目に「(polygon)」とついているのは、削除しようとしているコンポーネントの名前を表しています。次節以降、ひとつのルーチンに複数のコンポーネントを配置しますが、そのような場合にどのコンポーネントを削除しようとしているのか一目でわかるように表示されています。

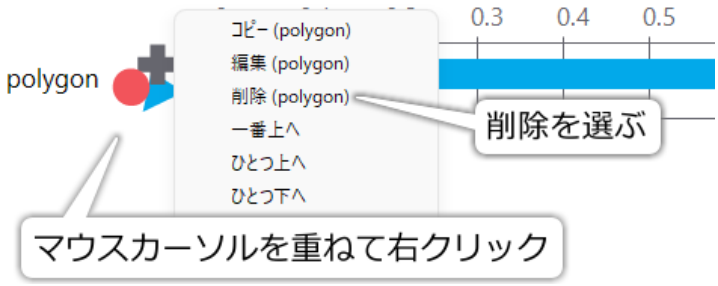


図 2.10　ルーチンペインのコンポーネントのアイコン上でマウスを右クリックするとメニューが表示されます。「削除」を選択するとコンポーネントを削除できます。

さて、これでコンポーネントをルーチンペインに配置し、プロパティを編集し、不必要なコンポーネントを削除し、実験を実行することができるようになりました。続いてコンポーネントのプロパティを編集して刺激の色や大きさを調節する方法を学びましょう。

チェックリスト

- Polygon コンポーネントを用いて三角形と長方形を表示できる。
- コンポーネントをルーチンから削除することができる。
- 本書で項目名を [ ] で囲んだ表記が何を表しているかを説明できる。

2.3 位置と大きさを指定しよう

再び Polygon コンポーネントを用いて、PsychoPy における視覚刺激の位置と大きさ、向きを指定する方法を習得しましょう。ルーチンペインに Polygon コンポーネントをひとつ配置して、 **[形状]** を長方形にしてください。

Polygon コンポーネントのプロパティ設定ダイアログの「レイアウト」タブを表示すると、 **[位置 [x, y] $]** 、**[サイズ [w, h] $]** という項目があります。それぞれ刺激の位置と大きさの指定に対応しています。これらの項目に 1.0 とか 150 とかいった値を入力することによって位置や大きさを指定するのですが、実際にこういった値を入力した時にどのような結果が得られるかを理解するためには、PsychoPy における位置と大きさの単位を理解する必要があります。

位置や大きさの単位は、同じタブにある **[空間の単位]** という項目で指定します。 表 2.2 に PsychoPy で使用できる単位を示します。スクリーン上の画素 (ピクセル) で指定する pix、センチメートルで指定する cm、スクリーンの大きさに対する比で指定する norm と height、視角で指定する deg, degFlat, degFlatPos があります。これらの単位の関係を、スクリーンの解像度が横 1920 ピクセル、縦 1080 ピクセル、寸法が幅 51.0cm、高さ

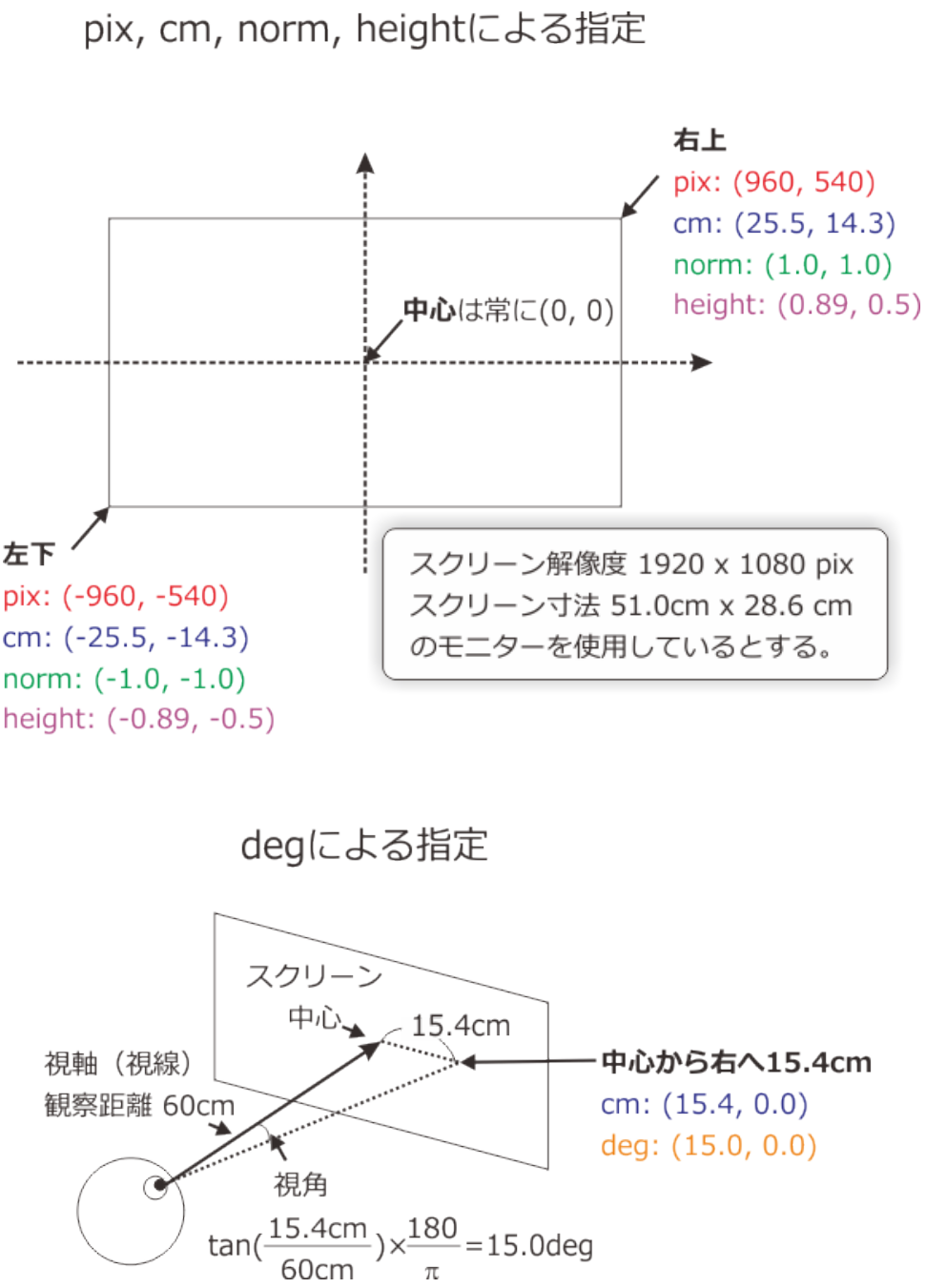


図 2.11　PsychoPy で使用できる単位。

28.6cm のモニターを例として示したのが 図 2.11 です。いずれの単位でもスクリーンの中心は常に原点 (0, 0) で、水平方向は右、垂直方向は上が正の方向です。スクリーンの右上の位置を pix で示す場合、スクリーンの横方向に 1920 ピクセルあるのですからスクリーン中心を基準にすればスクリーンの左端は 960 ピクセル (1920 ピクセルの半分) 進まなければいけません。同様に垂直方向に 1080 ピクセルありますからスクリーンの上端は中心から 540 ピクセル進まなければいけません。ですから、スクリーン右上の座標は (960, 540) です。スクリーン左下の座標は水平垂直共に負の方向に進まないといけないので、(-960, -540) です (厳密にな議論は「 2.12.1: スクリーン左下の座標についての厳密な議論 (上級) 」参照)。単位が cm の場合は同様の計算でスクリーン右上が (25.5, 14.3)、左下が (-25.5, -14.3) です。

norm と height は、Builder で実験を作成するときの（設定変更してい場合の）単位の初期値として使われる

重要な単位です。これらの単位はスクリーンの解像度に対する比で位置や長さを指定します。norm ではスクリーンの解像度に関わらず必ず右上の座標は (1.0, 1.0)、左下の座標は (-1.0, -1.0) になります。一般的に PC のモニターは水平方向の方が解像度は高いので、垂直方向の 1.0 よりも水平方向の 1.0 の方が画面上の長さは長くなります。そのため、正方形や円を表示したり図形を回転したりするときに注意が必要です（「 2.4: 刺激を回転させよう 」参照）。前節で「設定によっては三角形が横長になる」、「正方形ではなく長方形が描かれる」と書いたのは、単位が norm の場合に起こる現象です。PsychoPy 3.0.3 より前のバージョンでは、この norm が単位の初期値でした。

一方、PsychoPy 3.0.3 から単位の初期値となった height は、スクリーンの高さが 1.0 になるように水平方向の幅を決めます。 図 2.11 の例の場合、垂直方向 1080 ピクセルに対して水平方向に 1920 ピクセルありますので、スクリーンの幅は 1920 ÷ 1080 ≒ 1.78 です。スクリーンの幅が 1.78、高さが 1.0 なのですから、スクリーンの右上と左下の座標は (0.89, 0.5) と (-0.89, -0.5) になることに注意してください。norm と比べて正方形を表示したり図形を回転させたりするのが容易なのが特徴ですが、縦横比の異なるモニター間で右上や左下の座標が異なる点が面倒です。norm や height には、実験用と学会発表用で異なる解像度のモニターを使っている時に、学会発表用に実験プログラムを書きなおさなくてもモニターの解像度に合わせて刺激を調整できるというメリットがあります。

残るは deg, degFlat, degFlatPos ですが、まず deg から説明します。 図 2.11 下の図のように被験者が 60cm 離れた位置にあるスクリーンの中心に真っ直ぐ視線を向けているとします。眼から視線を向けている対象に引いた直線を視軸と呼びます。さて、スクリーン中心から右へ 15.4cm の位置に刺激があるとして、眼からこの刺激の位置まで引いた直線と視軸が成す角度を考えましょう。三角関数を思い出していただければ 15.4cm を観察距離 60cm で割った値の正接 (tan) を求めれば角度が得られます。この角度の単位はラジアンなので分かりやすいように 180/π を掛けて単位を度 (deg) にすると 15.0deg です。この角度を視角と呼びます。視覚を研究する時には、刺激がスクリーン上で中心から何 cm 離れていたかよりも、網膜の中心から何 deg 離れていたかの方が重要な意味を持つことがよくあるので、単位として視角が頻繁に用いられます。PsychoPy では、あらかじめスクリーンの寸法と観察距離を登録しておくことで、deg を単位として刺激の位置や大きさを指定できます。寸法と観察距離の登録方法はこの章の「 2.9: 実験の設定を変更しよう 」で触れますので、ひとまずは「deg という単位が使える」ということを覚えておいてください。なお、PsychoPy の deg の計算は恐らく実行速度を速めるために近似的な方法を用いています。より正確な値を必要とする人のために用意されているのが degFlat, degFlatPos という単位です。詳しくは「 2.12.2:PsychoPy における視角の計算について 」をご覧ください。

表 2.2　PsychoPy で使用できる位置と大きさの単位

単位	説明
pix	モニター上の画素に対応します。例えば 100pix であればモニターの 100 画素分に対応します。
cm	モニター上での 1cm に対応します。使用しているモニターの画面の寸法と縦横の画素数を Monitor Center に登録しておく必要があります。
deg	視角 1 度に対応します。例えば 2.5deg であれば視角 2.5 度に対応します。使用しているモニターの画面の寸法と縦横の画素数、モニターと参加者の距離を Monitor Center に登録しておく必要があります。
norm	モニターの中心から上下左右の端までの距離が 1.0 となるように正規化された単位です。一般的に PC 用のモニターは縦方向より横方向の方が長いので、norm の単位で幅と高さに同じ値を指定すると横長の長方形になります。
height	モニターの上端から下端の距離が 1.0 になるように正規化された単位です。norm と異なり、一般的な PC 用モニターで幅と高さに同じ値を指定するとほぼ正方形となります。「ほぼ」というのはモニターによっては画素の縦横の長さがわずかに異なる場合があり、そのようなモニターでは正確に正方形にならないからです。
degFlat	deg と同様ですが、deg よりも正確に計算します (「2.12.2:PsychoPy における視角の計算について 」参照)。
degFlatPos	deg と同様ですが、deg よりも正確に計算します (「2.12.2:PsychoPy における視角の計算について 」参照)。
実験の設定に従う	実験設定ダイアログで指定された単位に従います。実験設定ダイアログで「PsychoPy の設定に従う」に設定されている場合は PsychoPy 設定ダイアログの **[単位]** に従います。

では、実際に刺激の大きさを変化させてみましょう。Polygon コンポーネントをひとつルーチンペインに置いて、 図 2.12 上のように **[サイズ [w, h] $]** の値を (0.2, 0.1) にしてください。数値の両脇の括弧やカンマを忘れずに入力してください（ **括弧は丸括弧でも角括弧でも前後で一致していれば構いません** ）。 **[サイズ [w, h] $]** の w と h はそれぞれ width と height ですから、(0.2, 0.1) と入力すれば、スクリーンの高さを基準として幅 0.2、高さ 0.1 の長方形を表示するように指定したことになります。入力を終えたら実験を実行すると、 図 2.12 下のようにスクリーン中央に横幅が高さの 2 倍の長方形が表示されるはずです。三角形が表示された人は **[形状]** を長方形にし忘れているので変更してください。ものすごく大きな長方形が表示されたり、何も表示されなかったりした場合は、標準の単位の設定が何らかの理由で height になっていない可能性が考えられます (研究室の共用コンピュータを使用していて設定が変更されている場合など)。その場合は **[空間の単位]** を height に変更してみてください。毎回手作業で変更するのが面倒な場合は「 2.9.2:「スクリーン」タブ 」を参考に PsychoPy の設定を変更してください。

使用している環境によっては、 **[サイズ [w, h] $]** や **[位置 [x, y] $]** などに入力されたカンマ (,) と小数点 (.) が非常に区別しにくい場合があります。フォントの設定を変更すると改善される場合がありますので、気になる方は「 2.12.3:Builder の設定ダイアログで用いられるフォント 」をご覧ください。

また、使用している PC のグラフィック機能によっては長方形の対角線上の灰色の線が見える事があります (図 2.13)。PsychoPy では長方形を表示する時に実際には二つの直角三角形を並べているのですが、うまく並

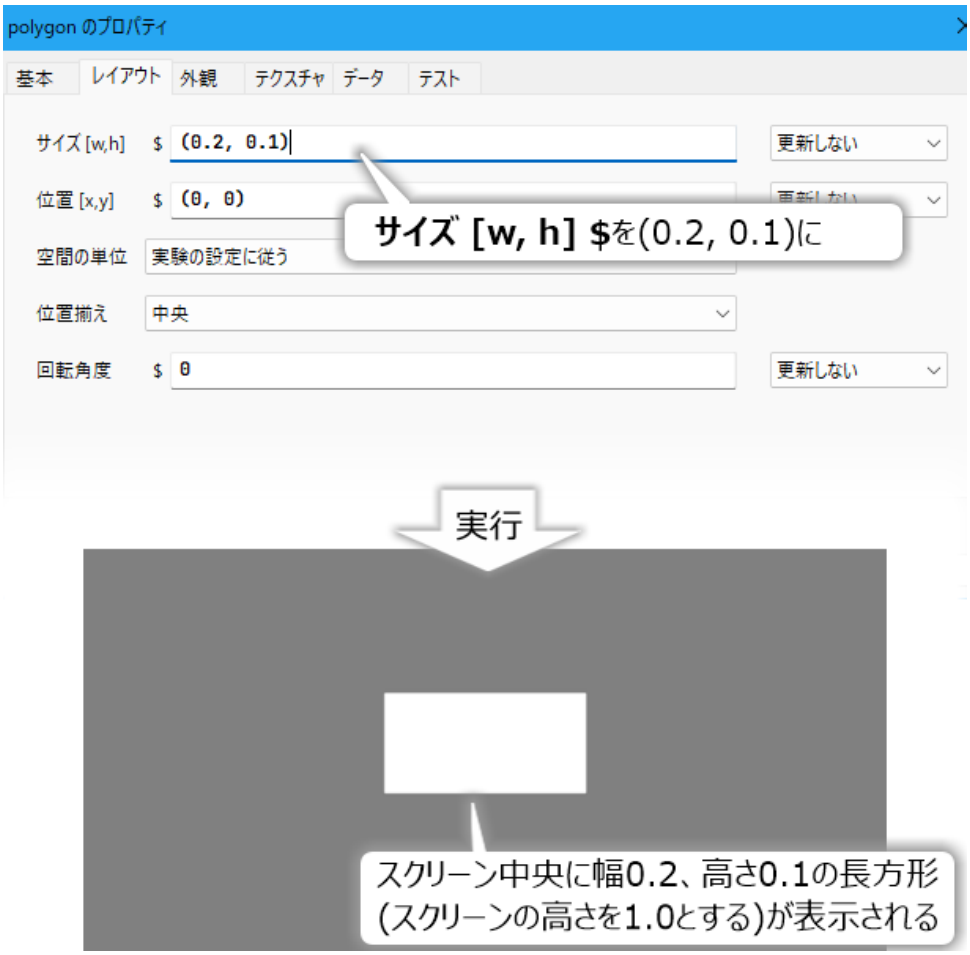


図 2.12　サイズの設定。Polygon コンポーネントの形状を長方形にして、横幅 0.2、高さ 0.1 に設定しています。

べられずに隙間ができてしまった時に生じる現象です。灰色の線は隙間から灰色の背景が見えてしまっているために生じています。多くの場合、Polygon コンポーネントのプロパティ設定ダイアログの「テクスチャ」タブにある **[補間]** というプロパティを変更するとこの問題は解消されます。

続いて刺激の位置を変更してみましょう。Builder の画面に戻ったら、先ほどの Polygon コンポーネントのプロパティ設定ダイアログを開いて、 図 2.14 上のように **[位置 [x, y] $]** に (0.1, 0) と入力して実行してみましょう。 **[位置 [x, y] $]** の x と y はそれぞれ水平 (X 軸) 方向、垂直 (Y 軸) 方向を表していますので、右と上が正の方向であることに注意すれば、(0.1, 0) はスクリーン中央から右へ 0.1 移動した位置を示しているはずです。実際に実行して確認すると、 図 2.14 下のように確かにスクリーン中央より右寄りに長方形が表示されます。

でも、 図 2.14 を見ただけでは長方形が右寄りに表示されていることがわかりますが、目視しただけでは本当に 0.1 右に寄っているのかの判断は困難です。そもそも、この (0.1, 0) という位置指定は長方形のどこを指しているのでしょうか？ 答えは「レイアウト」タブの **[位置揃え]** にあります。標準では **位置揃え** が「中央」に設定されているので、(0.1, 0) は長方形の中央の座標に対応します。そうすると、長方形の左下の頂点は中央の座標 (0.1, 0) から左へ横幅の半分、下へ高さの半分だけ移動した位置にあるはずです。確認するために、ルーチンペインにもうひとつ Polygon コンポーネントを配置してみましょう。

ルーチンにもうひとつ Polygon コンポーネントを配置するには、最初に Polygon コンポーネントを配置した時と同様に、コンポーネントペインの Polygon コンポーネントのアイコンをクリックします。そうするとやはり最初のコンポーネントを配置した時と同様にプロパティ設定ダイアログが表示されます。今回は、Polygon コ

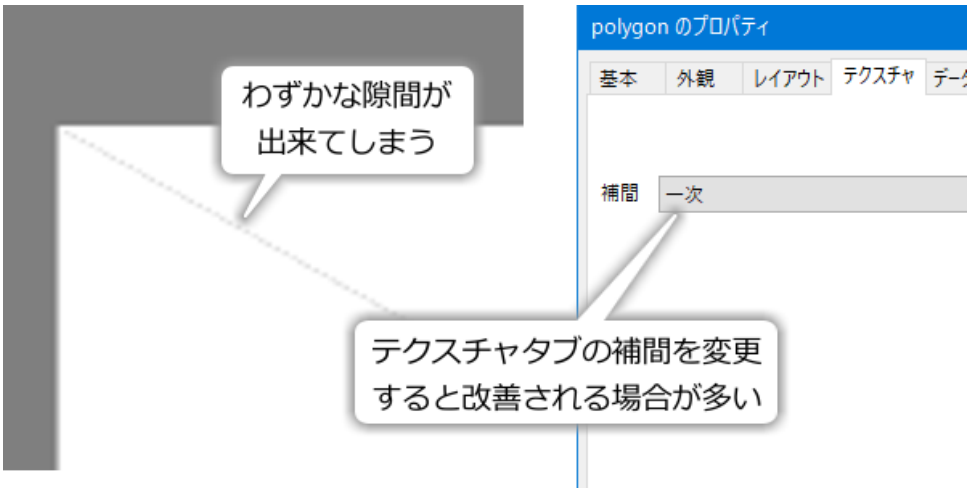


図 2.13　Polygon コンポーネントで長方形を表示すると細い線が見える事があります。「テクスチャ」タブの **[補間]** を変更すると多くの場合問題が解消されます。

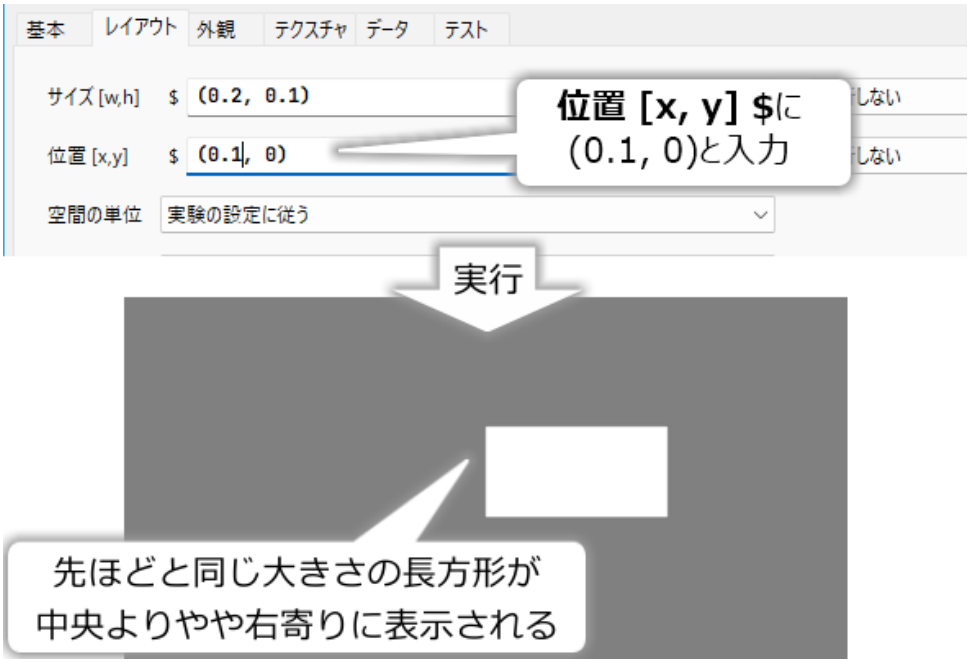


図 2.14　**[位置 [x, y] $]** に (0.1, 0) を設定すると長方形が右寄りに表示されます。

ンポーネントを使用する練習も兼ねて三角形を描画させてみましょう。まず「基本」タブの **形状** が三角形になっていることを確認し、「レイアウト」タブへ移動して 図 2.15 のように **[サイズ [w, h] $]** に (0.2, 0.2)、 **[位置 [x, y] $]** に (0, -0.05) と同じ値を入力してください。そして、 **[位置揃え]** を「中央上」にしましょう。入力ができたら実験を実行してください。 図 2.15 左のように、三角形の上側の頂点が長方形の左下の頂点と一致するはずです。

なぜこのようになるのかを解説したのが 図 2.15 の右です。先ほど述べたように、長方形の左下の頂点の座標は (0, -0.05) になるはずです。一方、三角形は 図 2.15 左の結果からおわかりのように底辺が水平な二等辺三角形として描かれます。三角形の **[位置 [x, y] $]** を (0.-0.05) にしたうえで **[位置揃え]** を中央上にしましたので、三角形の上の頂点が (0.-0.05) の位置になるように描かれます。この座標は長方形の左下の頂点の座標と一致しているので、 図 2.15 左のような出力が得られるというわけです。

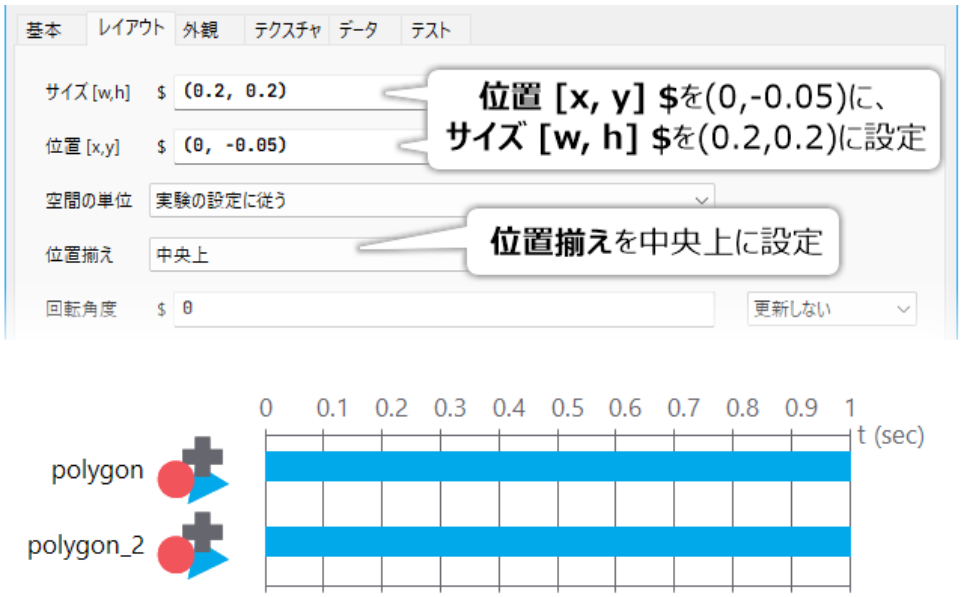


図 2.15　Polygon コンポーネントを追加して位置と大きさを設定します。コンポーネントが追加されると下の図のようにルーチンペイン上に複数のアイコンが並びます。

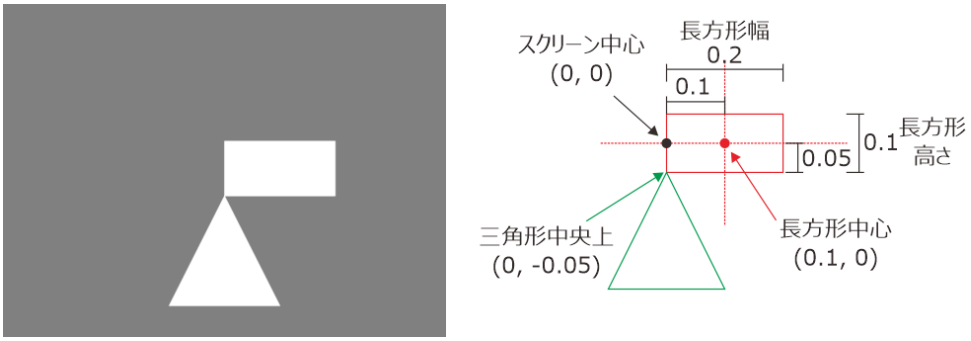


図 2.16　三角形と長方形の Polygon を描画した結果。長方形の左下の頂点と三角形の上の頂点の座標が一致することがわかります。

ここで 図 2.15 右に示されてる三角形が正三角形よりやや縦長である点に注意してください。高さ 0.2 の正三角形の幅は $0.2/\sqrt{3} \times 2$ = 約 0.2309 ですから、正三角形を表示するには **[サイズ [w, h] $]** を (0.2309, 0.2) としなければいけません。 **[位置揃え]** を中央にした時に **[位置 [x, y] $]** が指し示す位置が三角形の重心と一致しない事にも注意する必要があります。

なお、「 **[位置揃え]** が中央の時に **[位置 [x, y] $]** は 図形の左右端の中点、上下端の中点に対応する」という原則は本書で取り上げる他の視覚刺激に対しても成り立つのですが、Polygon コンポーネントで五角形以上の正多角形を描画した時のみは例外的に、 **[位置 [x, y] $]** が図形の左右端の中点、上下端の中点ではなく、外接する楕円の中心の座標に一致します。

試しに 図 2.17 左のように Polygon コンポーネントの **[形状]** を正多角形にしてください。すると **[頂点数]** の値が変更できるようになりますので 5 を指定してみましょう。指定を終えて実行すると画面上に正五角形が描かれますが、三角形や長方形を描いた時よりやや小さく描かれるはずです。 図 2.17 右には **[サイズ [w, h] $]** が (0.1, 0.1) の五角形と三角形を描画した結果を示していますが、底辺の位置が揃っていないことがわかります。

本節では PsychoPy の標準の単位である height の場合を例に解説してきましたが、他の単位でも考え方は同じ

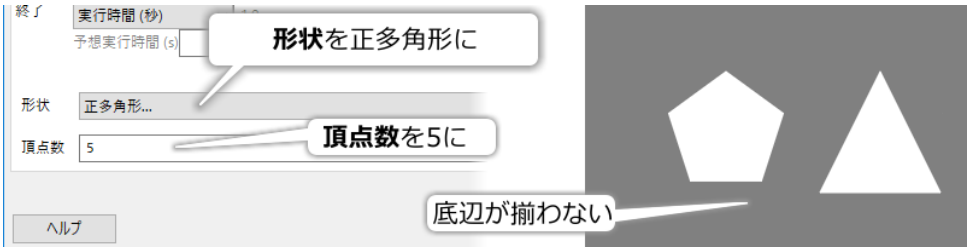


図 2.17　正五角形を描くための設定 (左) と正五角形を描いた結果 (右)。 比較のために同じ **[サイズ [w, h] $]** で描いた三角形を並べて描画しています。

です。単位はそれぞれのコンポーネントで独立して設定できるので、ひとつのポリゴンは height、もうひとつのポリゴンは deg を使うといったことも可能です。位置と大きさの指定についての解説はこのくらいにしておいて、次は図形を回転させてみましょう。

チェックリスト

- PsychoPy における座標系の原点と水平、垂直軸の正の方向を答えられる。
- height が単位の状態で **[位置 [x, y] $]** 、 **[サイズ [w, h] $]** 、 **[位置揃え]** を使って任意の大きさの多角形を任意の位置に表示させることができる。
- Polygon コンポーネントで正五角形以上の正多角形を描画できる。
- Polygon コンポーネントで **[頂点数]** が 5 以上の時に **[位置 [x, y] $]** が例外的に図形のどの位置に対応するかを答えることができる。
- cm、deg、norm、height という単位を説明できる。
- 複数のコンポーネントをルーチンペインに配置できる。
- **[位置揃え]** の働きを説明できる。

2.4 刺激を回転させよう

再び Polygon コンポーネントのプロパティ設定ダイアログを開いてください。 **[位置 [x, y] $]** の上に **[回転角度 $]** という項目があります。この項目に回転量を数値で入力することによって、刺激を回転させることができます。入力する数値の単位は「度」で、時計回りが正の回転方向です。つまり、 **[回転角度 $]** を 30 に設定すると時計回りに 45 度、90 に設定すると 90 度回転します。回転の中心は **[位置 [x, y] $]** によって指定されている位置ですが、 **[位置揃え]** の設定によって結果が異なることに注意してください。 図 2.18 に縦長の三角形を **[位置揃え]** 「中央」と「中央上」に設定して 45 度、90 度回転した結果を示します。なお、回転角度として負の値を設定すると反時計回りに回転します。

刺激の回転について学んだついでに、先ほど「norm を単位にすると図形の回転が難しい」と述べた点について確認しておきましょう。Polygon コンポーネントをルーチンペインにひとつ配置して、 **[空間の単位]** を norm に、 **[サイズ [w, h] $]** を [0.5, 0.5] にしてください。そして、 **[回転角度 $]** に 0 を入力した場合と 30 した場合の結果を比較してみてください。他のコンポーネントを置いていても構いませんが、他のコンポーネントと重なるとわかりにくいので削除しておいた方がよいと思います。実行すると、 図 2.19 左のように **[回転角度 $]** が 0 であれば長方形が表示され、30 であれば傾いた平行四辺形が表示されたはずです。これは、norm を単位に使用した時に、スクリーンの右上の座標が (1, 1)、左下の座標が (-1, -1) になるように変換を行うために生じ

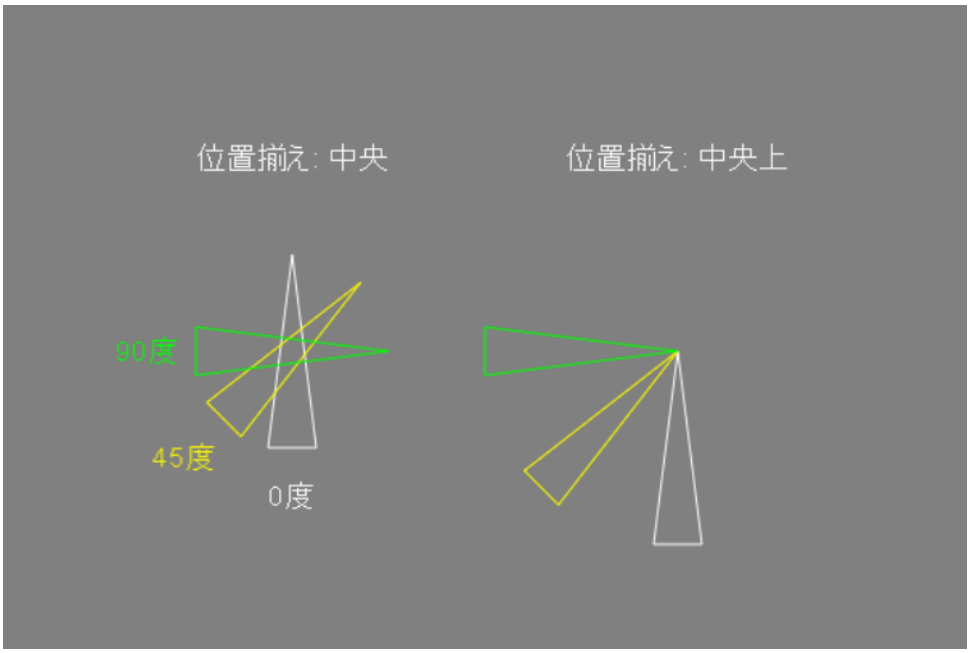


図 2.18　**[回転角度 $]** の指定による図形の回転。 **[位置揃え]** による違いに注意。

る現象です。通常、PC に接続されているモニターのスクリーンは横に長いので、変換の際に横方向に引き伸ばされてしまうのです。そのため、水平軸や垂直軸に平行な辺しか含まない図形は norm を使用しても単に横長に見えるだけですが、平行ではない辺を含む図形では辺が交わる角度が変わってしまうのです。横長のスクリーンを使用時に norm を使う限り、この問題は回避できません。height を使った方が良いでしょう。

さて、以上で図形の大きさ、位置、回転方向の指定方法の解説が終わりました。これらの設定に用いるプロパティ、**[サイズ [w, h] $]**、**[位置 [x, y] $]**、**[回転角度 $]**、**[空間の単位]** の使い方は、視覚刺激を表示するコンポーネントでほぼ共通していますので、使い方をしっかりマスターしておきましょう。

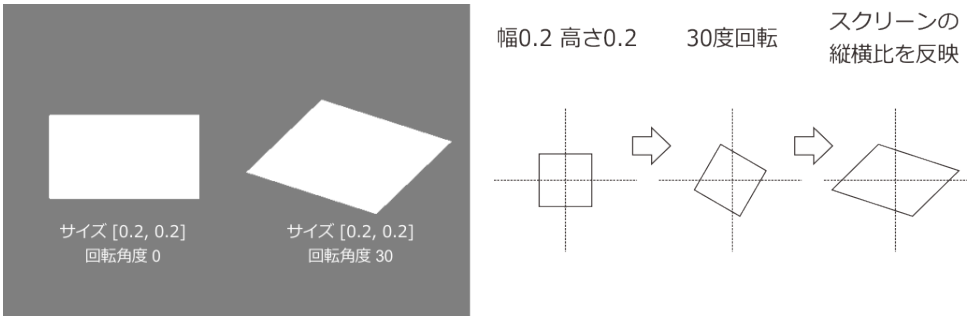


図 2.19　**[空間の単位]** に norm を指定した場合の回転。回転してからスクリーンの縦横比を反映させるため図形が歪みます。

チェックリスト

- **[回転角度 $]** に適切な値を設定して図形を回転させて表示させることができる。
- 図形の正の回転方向を答えられる。
- 単位が norm の時に図形を回転させた時に生じる図形のひずみを説明できる。

2.5 色の指定方法を理解しよう

今度は Text コンポーネントを使う練習をしながら、色の指定方法をマスターしましょう。ルーチンに Text コンポーネント (図 2.20 のアイコン) をひとつ配置してください。Polygon コンポーネントなど他のコンポーネントを配置している人は削除しておいてください。

配置した Text コンポーネントのプロパティ設定ダイアログを開いてください。 **[位置 [x, y] $]** 、 **[回転角度 $]** 、 **[回転角度 $]** 、 **[空間の単位]** は Polygon コンポーネントの同名のプロパティと同じ働きをします。ここではこれらに加えて「基本」タブの **[文字列]** と「書式」タブの **[文字の高さ $]** 、「レイアウト」タブの **[反転]** 、 **[折り返し幅 $]** を使ってみましょう。

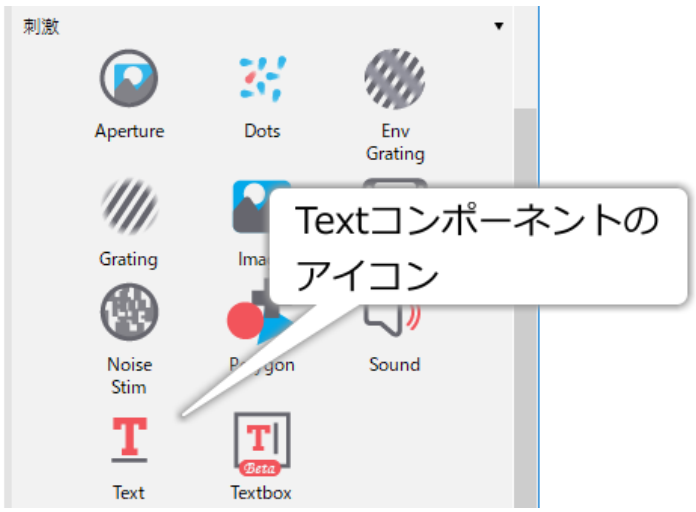


図 2.20　Text コンポーネントのアイコン

まず、**[文字列]** に適当な文字列を入力してください。日本語でも英語でも構いませんし、改行しても構いません。そして「書式」タブの **[文字の高さ $]** に 0.02 や 0.04 といった数値を指定して実行してみましょう。スクリーン上に **[文字列]** に入力した文字列が表示されるはずです。 **[文字の高さ $]** を変更して実行し、文字の大きさが変わることを確認してください (MacOS で日本語の文字が欠けてしまう場合は「 2.12.4:Mac 上で日本語の文字が欠けてしまう場合の対策 」をご覧ください)。

[空間の単位] を pix にして **[文字の高さ $]** を 0.1 などにして実行すると、文字が 1 ピクセルより小さくなって何も表示されません。同様に **[空間の単位]** に height が設定されている時に **[文字の高さ $]** を 24 などにしてしまうとスクリーンよりも文字がはるかに大きくなってしまい正常に表示されません。特に数値が大きすぎる場合はエラーダイアログが出て実験自体が実行できない場合があります。ありがちなミスなので注意してください。

なお、「書式」タブの **[言語スタイル]** は文字の書き方を指定します。左から右 (LTR: left to right)、右から左 (RTL: right to left)、Arabic のいずれかです。初期値は LTR で、通常は変更する必要はないでしょう。

続いて「レイアウト」タブの **[反転]** と **[折り返し幅 $]** です。 **[反転]** は None(または空白) にしておくと通常の文字列が表示されますが、vert と入力すると上下反転、horiz と入力すると左右反転して文字列が表示されます (図 2.21)。 **[折り返し幅 $]** は、 **[文字列]** に改行を含まない長い文字列が入力されたときに自動的に折り返す幅を指定します。折り返し幅の単位は **[空間の単位]** プロパティに従います。 図 2.21 では **[折り返し幅 $]** を指定しなかった場合、0.5 を指定した場合、0.3 を指定した場合を示しています。いずれも **[文字列]** には改行を含めずに文を入力してあるのですが、適切に折り返しが行われていることがわかります。ただし残念なこと

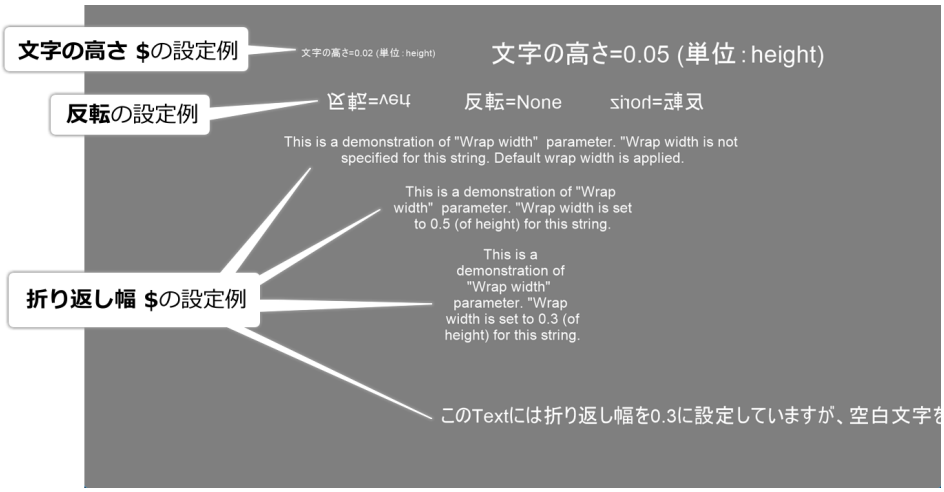


図 2.21　Text コンポーネントにおける文字の高さ、反転、折り返し幅の設定例。

に、文字列の自動折り返しは日本語ではうまく機能しません。 図 2.21 の一番下の日本語の文字列は **[折り返し幅 $]** に上の英文と同じ 0.3 を指定しているのですが、折り返されずに画面からはみ出してしまっています。

Text コンポーネントに慣れたところで、次は色の指定をしてみましょう。色を指定するには「外観」タブの **[前景色]** に値を設定します。 **[前景色]** の入力欄の右側に小さなボタンがありますが (図 2.22)、これをクリックすると「カラーピッカー」という色選択ツールを開くことができます。カラーピッカーは便利なのですが、PsychoPy における色指定の仕組みを理解していないとわかりにくい内容もありますので、まずは色指定の仕組みを学びましょう。

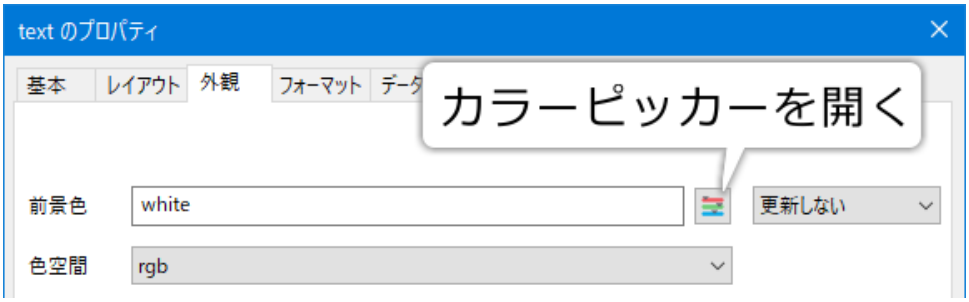


図 2.22　色を指定する項目には右側に「カラーピッカー」を開くボタンがあります

PsychoPy では、色を表す値として web/X11 Color name と呼ばれる色名と、16 進数表記の web カラーと、色空間における座標値を利用することができます。図 2.23 は web/X11 Color name の一覧を示しています。先ほどの Polygon コンポーネントの表示で **[塗りつぶしの色]** に black と書くことで黒く塗りつぶすことができたのは、この web/X11 Color name による指定が利用できるからです。図 2.23 を見ながら、Text コンポーネントのプロパティ選定ダイアログの **[前景色]** に色名を入力して実行してみましょう。

16 進数表記の web カラーというのは、0xFD087A や#FAF のように、「0x または#」+「0 から 9 の数字および A から F のアルファベット文字を 3 文字または 6 文字」で色を表す方法です。web ページを作成するときによく用いられる色指定なので、そちらですでにこの指定方法をご存じの方には使いやすいでしょう。しかし、ご存じでない方は次に紹介する色空間における座標値を指定する方法を覚えた方が良いと思います。web カラーによる色指定については「 2.12.5:16 進数と色表現 」で解説していますので詳しくはそちらをご覧ください。

さて、色空間における座標値を指定する方法ですが、これは人間が知覚できる色が三次元空間の点として表現

black	aliceblue	darkcyan	lightyellow	coral
dimgray	lavender	teal	lightgoldenrodyellow	tomato
gray	lightsteelblue	darkslategray	lemonchiffon	orangered
darkgray	lightslategray	darkgreen	wheat	red
silver	slategray	green	burlywood	crimson
lightgray	steelblue	forestgreen	tan	mediumvioletred
gainsboro	royalblue	seagreen	khaki	deeppink
whitesmoke	midnightblue	mediumseagreen	yellow	hotpink
white	navy	mediumaquamarine	gold	palevioletred
snow	darkblue	darkseagreen	orange	pink
ghostwhite	mediumblue	aquamarine	sandybrown	lightpink
floralwhite	blue	palegreen	darkorange	thistle
linen	dodgerblue	lightgreen	goldenrod	magenta
antiquewhite	cornflowerblue	springgreen	peru	fuchsia
papayawhip	deepskyblue	mediumspringgreen	darkgoldenrod	violet
blanchedalmond	lightskyblue	lawngreen	chocolate	plum
bisque	skyblue	chartreuse	sienna	orchid
moccasin	lightblue	greenyellow	saddlebrown	mediumorchid
navajowhite	powderblue	lime	maroon	darkorchid
peachpuff	paleturquoise	limegreen	darkred	darkviolet
mistyrose	lightcyan	yellowgreen	brown	darkmagenta
lavenderblush	cyan	darkolivegreen	firebrick	purple
seashell	aqua	olivedrab	indianred	indigo
oldlace	turquoise	olive	rosybrown	darkslateblue
ivory	mediumturquoise	darkkhaki	darksalmon	blueviolet
honeydew	darkturquoise	palegoldenrod	lightcoral	mediumpurple
mintcream	lightseagreen	cornsilk	salmon	slateblue
azure	cadetblue	beige	lightsalmon	mediumslateblue

図 2.23　PsychoPy で使用できる色名 (web/x11 color name)

できることを利用しています。ちょっと数学的な話になりますが、空間の位置を表現する方法は何通りもあります。例えば二次元平面の水平方向に X 軸、垂直方向に Y 軸を引いて「原点から X 軸の方向に 10、Y 軸の方向に 10 進む」といった具合に平面上の位置を表現することができますが、同じ位置を「原点から 45 度の方向に $10\sqrt{2}$ 進む」と表現することもできます。前者を直交座標、後者を極座標と呼びますが、同じ位置でも直交座標と極座標では異なる数値で表されるわけです。これと同様に、色の表現も座標軸の取り方によって同一の色に対して複数の方法で表現することができます。PsychoPy では、RGB、HSV、LMS、DKL という 4 種類の表現をサポートしています。しかし、HSV は Builder からは使用できず、LMS と DKL は専用の装置を用いて実験に使用するモニターをキャリブレーション (調整) しないと使えませんので、本書では RGB による表現を使用します。この色表現を切り替えるのがプロパティ設定ダイアログの **[色空間]** です。 **[色空間]** の値が rgb に設定されていることを確認しておきましょう。なお、 **[色空間]** の値は web/X11 color name や web カラーで色を指定する時には無視されます。

ようやく色空間における座標値で色を指定する方法を説明する準備ができました。**[色空間]** を rgb に設定している場合、赤 (R)、緑 (G)、青 (B) の三種類の光の強度の組み合わせで色を指定することができます。RGB のそれぞれの成分の強度は-1.0 から 1.0 の実数で指定します。Text コンポーネントのプロパティ設定ダイアログを開いて **[前景色]** に

```
-1, -1, -1
```

と記入してください。半角文字 (日本語入力モードは OFF) で、カンマを忘れないでください。カンマの後ろの空白文字はあってもなくても構いません。実行すると黒色で文字が表示されるはずです。続いて以下の三つを順番に試してみましょう。

```
 1, -1, -1
-1,  1, -1
-1, -1,  1
```

上から順番に赤色、緑色、青色で文字が表示されたはずです。三つの数字が左から順番に R、G、B に対応しているのが理解していただけたでしょうか。さらに以下の値も試してみましょう。これらがどのような色になるかは実際に皆さんが確認してみてください。

```
 -0.3,  -0.3, -0.3
  0.2,   0.2,  0.2
-0.92, -0.46, 0.05
 0.09,  0.63, 0.13
```

なお、一般的なグラフィックソフトウェアでは RGB それぞれ 256 段階の整数で指定する表現方法が用いられていますが、この 256 段階表現の色を PsychoPy で使用するには-1.0 から 1.0 の実数に換算する必要があります。256 段階表現の場合、RGB 各成分の最小値は 0 で最大値は 255 ですから、値を 255.0 で割れば 0.0 から 1.0 の値が得られます。これを-1.0 から 1.0 に変換すればいいのですから、2 倍して 1.0 を引けば目的が達成されます。式で書けば以下の通りです。

```
2 × (256 段階表現の値 ÷ 255.0)-1.0
```

慣れないうちは狙った色を指定するのは難しいですが、 図 2.22 で触れたカラーピッカーがここで役に立ちます。 図 2.22 のボタンをクリックすると、 図 2.24 に示すカラーピッカーダイアログが表示されます。ダイアログ中央の RGB Channels という枠内にある **[R]**、**[G]**、**[B]** のスライダーを調整すると、それに対応する色でダイアログ左側の領域が塗りつぶされます。文章で説明するより触ってみた方が早いと思いますので、ぜひ自分で操作してみてください。

ダイアログ右側に並んでいる色見本をクリックすると、その色の RGB 値がスライダーに反映されます。色名と RGB 値がどのように対応しているかを確認したり、色名で表される色をほんの少し変更した色を指定したりするときに便利です。「この色を使いたい！」という色ができあがったら、ダイアログ左下の **[出力空間]** が PsychoPy RGB (rgb) になっていることを確認したうえで、右下にある OK ボタンをクリックしましょう。するとダイアログが閉じると同時に、**[前景色]** の欄に作成した色の RGB 値が自動的に挿入されます。色見本の色名を自動挿入することはできませんので、色名で指定したい場合は色名を覚えるかメモしたうえでダイアログを閉じ、手作業で色名を入力する必要があります。

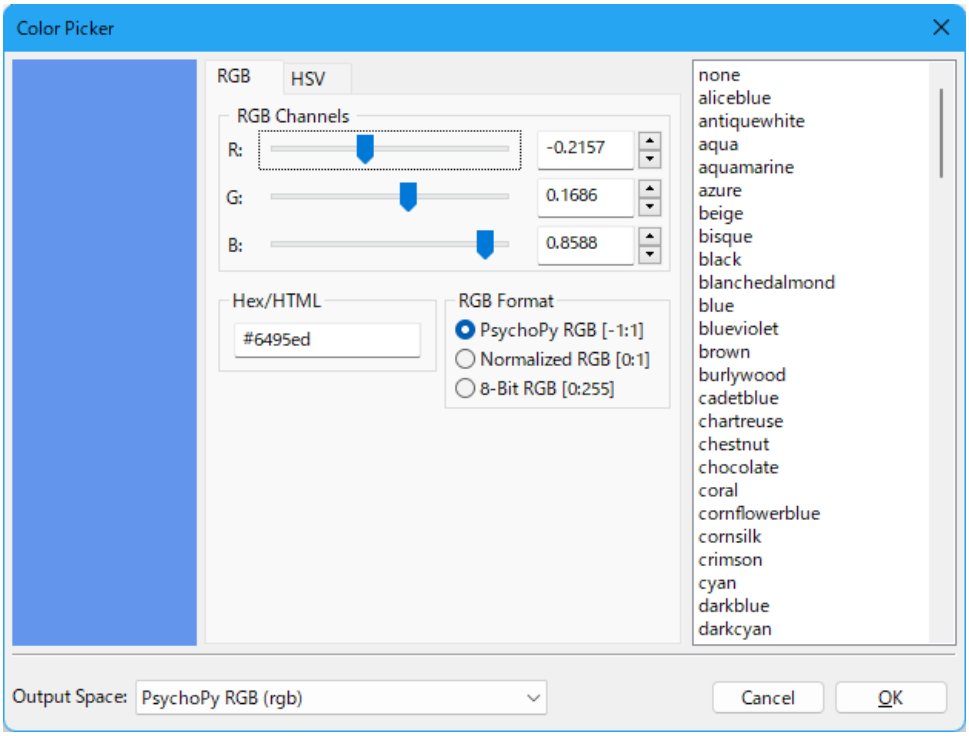


図 2.24　カラーピッカーダイアログ

最後に、Polygon コンポーネントの色指定について補足しておきます。Polygon コンポーネントには色指定に関して **[塗りつぶしの色]** と **[枠線の色]** という 2 つのプロパティがあり、それぞれ塗りつぶしと枠線の色に対応しています。これらの色に None という値を指定することによって、内部が塗りつぶされていない枠線だけの図形や、枠線がない図形を描画することができます。 **[塗りつぶしの色]** を None にすると内部が塗りつぶされていない枠線だけ、**[枠線の色]** を None にすると枠線がない図形になります。ぜひ覚えておいてください。

チェックリスト

- Text コンポーネントを用いて文字列を表示できる。
- 文字列を指定された位置に表示できる。
- 文字列を指定された大きさで表示できる。
- 文字列を上下反転、左右反転表示することができる。
- 文字列の自動折り返し幅を設定できる。どのような文字列では自動折り返しが起きないか説明できる。
- web/X11 color name による色指定で文字列の色を白、灰色、黒、赤、オレンジ色、黄色、黄緑色、緑、水色、青、ピンク、紫にすることができる。
- **[色空間]** を rgb に設定して、数値指定によって文字列の色を白、灰色、黒、赤、黄色、緑色、青色にすることができる。
- Polygon コンポーネントを用いて内部が塗りつぶされていない枠線だけの多角形を描画することができる。
- Polygon コンポーネントを用いて枠線がない多角形を描画することができる。

2.6 刺激の重ね順と透明度を理解しよう

刺激を色分けできるようになりましたので、刺激が重なってしまった時にどのような結果が得られるのかを解説できるようになりました。さっそく、刺激の重ねあわせについて解説しましょう。

Polygon コンポーネントひとつと Text コンポーネントひとつをルーチンペインに配置して、以下のように設定します。

- Polygon コンポーネント
  - **「基本」タブ**
    - **[形状]** を長方形にする
  - **「レイアウト」タブ**
    - **[サイズ [w, h] $]** を `(0.2, 0.2)` にする
  - **「外観」タブ**
    - **[塗りつぶしの色]** を `red` にする
  - 他のプロパティは初期値のままにする
- Text コンポーネント
  - **「基本」タブ**
    - **[文字列]** を `PsychoPy Builder による心理学実験` にする
  - **「書式」タブ**
    - **[文字の高さ $]** を `0.02` にする
  - 他のプロパティは初期値のままにする

どちらのコンポーネントを先にルーチンペインに配置したかによって、各コンポーネントのアイコンが並ぶ順番が異なります。先に配置したコンポーネントが上にあって、その下に配置した順番にアイコンが並びます。今までルーチンペイン上におけるアイコンの順番については触れませんでしたが、実はこの順番には大きな意味があります。図 2.25 をご覧ください。Builder では、ルーチンペインで上に配置されているコンポーネントから順にスクリーン上に表示します。ですから、ルーチンペイン上で下に配置されているコンポーネントほど重ね順は上になります。重ね順で上にあることを「手前にある」、下にあることを「奥にある」という言い方をすることもあります。

ルーチンペイン上での配置順を変更するには、変更したいコンポーネントのアイコン上へマウスカーソルを動かして、右クリックをしてメニューを表示させます。ここまではコンポーネントを削除する時の操作と同じです。削除する時にはメニューの「削除」という項目を選択しましたが、配置順を変更する時には「ひとつ上へ」、「ひとつ下へ」、「一番上へ」、「一番下へ」を選択します。

刺激の重ね順を解説したついでに、**[位置 [x, y] $]** などと同様に多くの視覚刺激用コンポーネントで使用できる **[不透明度 $]** を紹介します。**[不透明度 $]** は刺激の透明度を指定するプロパティで、0.0 から 1.0 の値をとります。0.0 は完全な透明で、スクリーン上では見えなくなってしまいます。1.0 は完全な不透明で、重ね順で下にある刺激は見えません。図 2.26 では、文字列の上に赤い正方形を重ねて、正方形の透明度を 1.0、0.75、

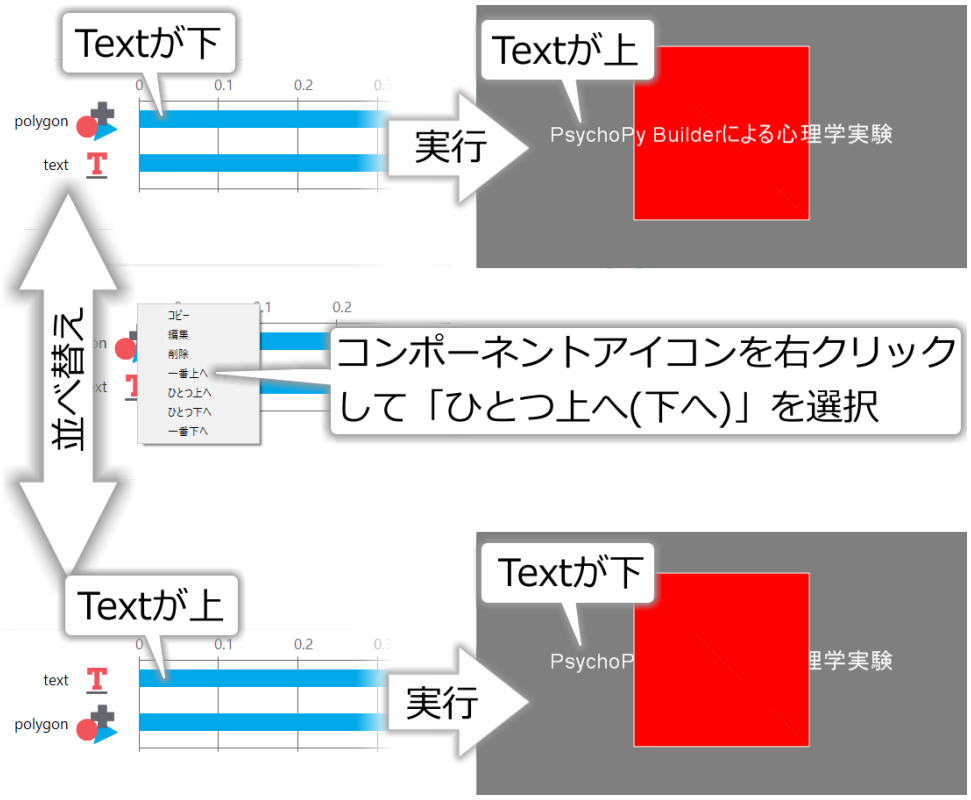


図 2.25　ルーチンペインにおける順序と刺激の重ねあわせの関係。ルーチンペインで上の方に配置されている刺激から順に表示されますので、スクリーン上での刺激の重ね順では上に配置されている刺激ほど下になります。

0.5、0.25、0.0 と変化させています。簡単に試すことができると思いますので、ぜひ各自でいろいろな値を試してみてください。

チェックリスト

- ルーチンペイン上における視覚刺激コンポーネントの順番とスクリーン上での重ね順の関係を説明できる。
- ルーチンペイン上におけるコンポーネントの順番を変更できる。
- 視覚刺激コンポーネントの透明度を設定して完全な透明、完全な不透明とその中間の透明度で刺激を表示させることができる。

2.7 刺激の提示開始と終了時刻の指定方法を理解しよう

この節では、刺激がいつ画面上に表示されて、いつ消えるかという時間的な側面を設定する方法について解説します。ここまで使用してきた Polygon コンポーネントと Text コンポーネントのプロパティ設定ダイアログの「基本」タブを見比べてみると、どちらにも **[開始]** と **[終了]** というプロパティが存在しているのがわかります (図 2.27)。これらが刺激の提示開始および終了を決めるプロパティです。解説に入る前に、ちょっと **[開始]** の上にある **[名前]** プロパティの使い方にも触れておきましょう（**[名前]** については次章で詳しく解説する予定です)。

まず、Polygon コンポーネントを二つルーチンペイン上に配置して、二つの正方形が左右に隙間なく並ぶよう

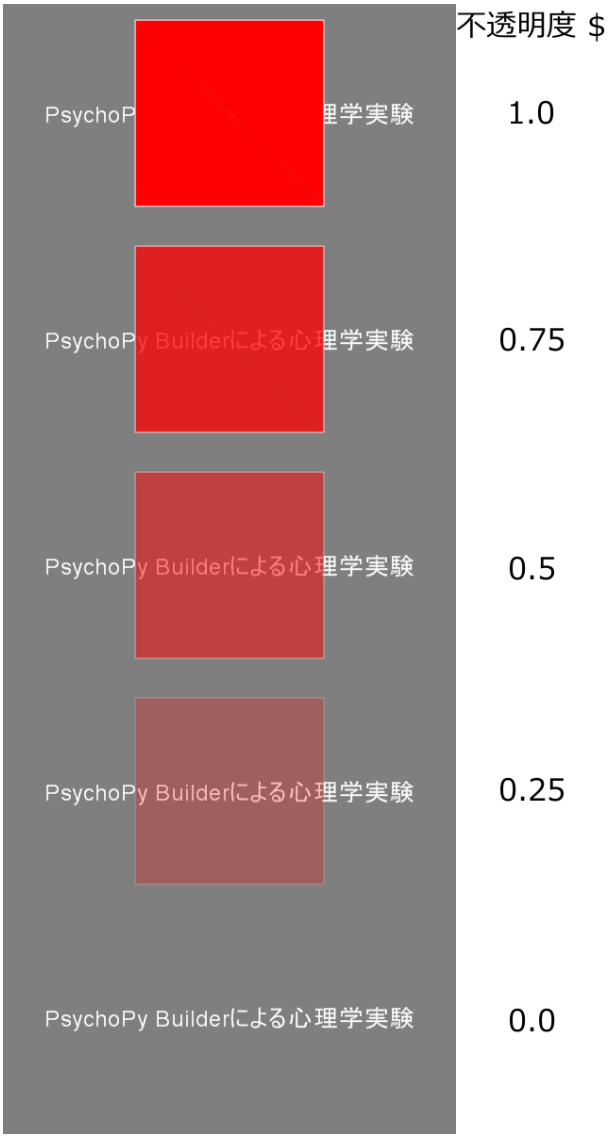


図 2.26　**[不透明度 $]** による透明度の指定。赤い正方形が文字列の上に重ねて、その赤い正方形の **[不透明度 $]** を段階的に変化させています

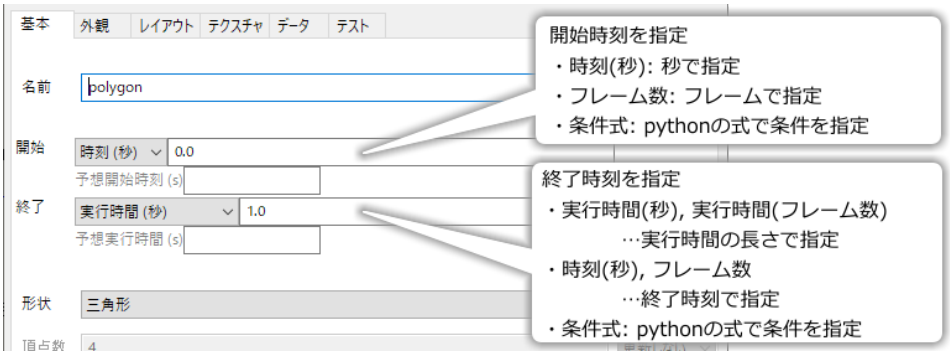


図 2.27　コンポーネントの開始、終了時刻を指定するプロパティ。「条件式」の使い方については 第 8 章 で触れます。

にしてみましょう。ここでは以下のように設定したとします。

- Polygon コンポーネントその 1 (赤)
  - –「基本」タブ
    - ＊ **[名前]** に red と入力
  - –「レイアウト」タブ
    - ＊ **[サイズ [w, h] $]** を (0.2, 0.2)、 **[位置 [x, y] $]** を (-0.1, 0) にする
  - –「外観」タブ
    - ＊ **[塗りつぶしの色]** を red にする
  - – 他のプロパティは初期値のままにする
- Polygon コンポーネントその 2 (緑)
  - –「基本」タブ
    - ＊ **[名前]** に green と入力
  - –「レイアウト」タブ
    - ＊ **[サイズ [w, h] $]** を (0.2, 0.2)、 **[位置 [x, y] $]** を (0.1, 0) にする
  - –「外観」タブ
    - ＊ **[塗りつぶしの色]** を green にする
  - – 他のプロパティは初期値のままにする
- 緑色の長方形が上に描画されるようにルーチンペイン上でのアイコンの順番を並べる。

[名前] を設定すると、 図 2.28 のようにルーチンペイン上で **[名前]** に設定した文字列が各コンポーネントのアイコンの左側に表示されます。同じ種類のコンポーネントが複数配置されている場合に区別しやすくてとても便利です。

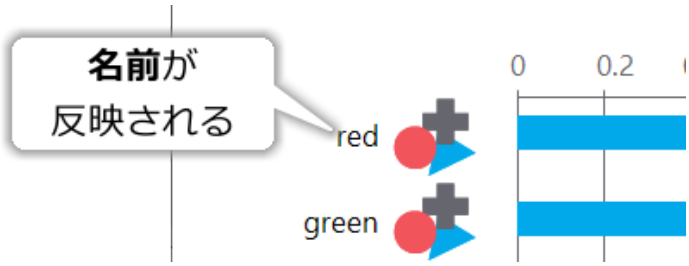


図 2.28 **[名前]** プロパティに文字列を入力すると、ルーチンペイン上でコンポーネントのアイコンの左側に入力した文字列が表示されます。

さて、この状態で実験を実行すると、赤と緑の正方形がスクリーンに 1 秒間表示されて終了するはずです。Builder の画面に戻ったら、ルーチンペイン上に配置した赤い正方形のプロパティ設定ダイアログを開き、 図 2.29 左のように **[開始]** を「時刻 (秒)」にして 0.5 と入力し、 **[終了]** を「実行時間 (秒)」にして 2 と入力してください。 **[開始]** と **[終了]** はそれぞれ初期状態で「時刻 (秒)」、「実行時間 (秒)」になっているはずですので、

変更していないのであればそれぞれ 0.5 と 2 を入力すれば大丈夫です。プロパティ設定ダイアログの OK をクリックしてダイアログを閉じると、ルーチンペイン上の表示が 図 2.29 右のように変化しているはずです。アイコンの横の青い横棒はコンポーネントが有効になる時間帯、視覚刺激の場合は画面上に表示されている時間帯を示しています。開始時刻に 0.5 秒を指定したので、青棒の左端は 0.5 の位置にあります。青棒の右端は終了時刻に対応していますが、こちらは少し説明が必要でしょう。 **[終了]** は「実行時間 (秒)」を指定して 2 と入力してありますので、刺激が画面上に表示されている時間は 2 秒です。刺激の表示開始時刻が 0.5 秒なのですから、終了時刻は 0.5 秒から 2 秒後の 2.5 秒でなければいけません。ルーチンペインの青棒の右端を確認すると、確かに右端は 2.5 秒の位置にあります。実験を実行してみると、最初に緑色の正方形のみが表示された後、一瞬 (0.5 秒) 遅れて赤い正方形が出現し、さらにすぐ後に緑色の正方形がスクリーンから消えます。赤い正方形は 2 秒間スクリーンに表示された後に消えて、その直後に実験が終了します。

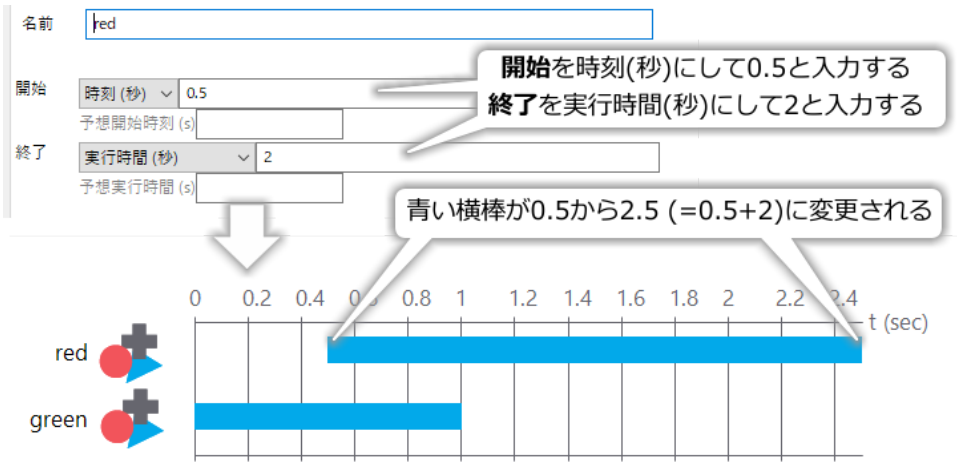


図 2.29　**[開始]** と **[終了]** の値を変更すると、ルーチンペイン上で青いバーの長さが変更されます。青いバーは実験実行時に、そのコンポーネントが有効となる (視覚刺激の場合は表示される) 時間帯を示しています。

基本的にこれだけで刺激の表示開始時刻と終了時刻を制御できますが、Builder では他の方法も提供されています。まず、 **[終了]** で「実行時間 (秒)」の他に「時刻 (秒)」を選択することができます。こちらを選択すると、終了時刻を直接入力して指定することができます。 図 2.29 の例で **[終了]** を「時刻 (秒)」に変更し、値に 2.5 を入力してみてください。ルーチンペインの青棒は 図 2.29 と同じになり、実行結果も同じになるはずです。ぜひ皆さん自身で手を動かして確認してみてください。

他には、刺激の表示開始、終了時刻を秒ではなくフレーム数で指定する方法があります。フレーム数で指定する場合も秒と同様に **[終了]** の項目で表示する時間の長さを指定するか、終了する時刻を直接指定するかを選択できます。「『フレーム数で指定』と言われてもよくわからない」という方は、「 2.12.6: 時刻指定における frame について 」をご覧ください。 図 2.27 で灰色の文字で描かれている **[予想開始時刻 (s)]** 、 **[予想実行時間 (s)]** という項目はフレーム数で指定する際に使う項目なので、そちらで合わせて説明してあります。

[開始] 、 **[終了]** ともに、秒による指定、フレーム数による指定に加えて「条件式」という選択肢がありますが、これは Python の条件式を直接記入する方法です。使いこなすには Python の文法を知っていなければいけませんので、ここでは一旦無視して 第 8 章 であらためて取り上げます。

最後に、非常に重要なテクニックをひとつ紹介しておきましょう。 **[終了]** の項目の数値を入力する欄を空白 (入力済みの数値を削除) してみてください。ルーチンペイン上でコンポーネントの有効時間帯を示す青い横棒が右側へ突き抜けてしまったはずです (図 2.30)。この状態になると、何らかの方法でルーチンが強制終了されない限り、この刺激は画面上に表示され続けます。この章で今まで製作してきたシンプルな「実験」では、

すべてのコンポーネントの終了時刻が決められていました (1.0 秒)。Builder の実験を実行した時には、ルーチン内に含まれるすべてのコンポーネントの終了時刻を経過したらルーチンが自動的に終了し、すべてのルーチンを実行すれば実験は自動的に終了します。 図 2.30 のように終了時刻が定められていないコンポーネントが存在すると、ルーチンが「永遠に」終了しません。現実には OS が再起動したり PC の電源が切れたりしていずれは終了してしまうでしょうが、そういう事態でもない限り刺激が表示され続けます。誤ってルーチンが終了しない状態に陥ってしまった時には、焦らずにキーボードの ESC キーを押してください。Builder の標準設定では、ESC キーが押されると直ちに処理中のルーチンを中断して実験を終了します。

終了時刻を定めないコンポーネントが定義できるようになっているのは、「実験参加者が反応するまで刺激を提示し続ける」といった実験手続を実現する為です。次章ではキーボードからの反応を取得する方法を学びますが、「キーボードが押されたらルーチンを終了する」という設定と、「ルーチンが終了するまで刺激を提示し続ける」という設定を組み合わせれば「実験参加者が反応するまで刺激を提示し続ける」ことが実現できるのです。詳しくは 第 3 章 で説明します。

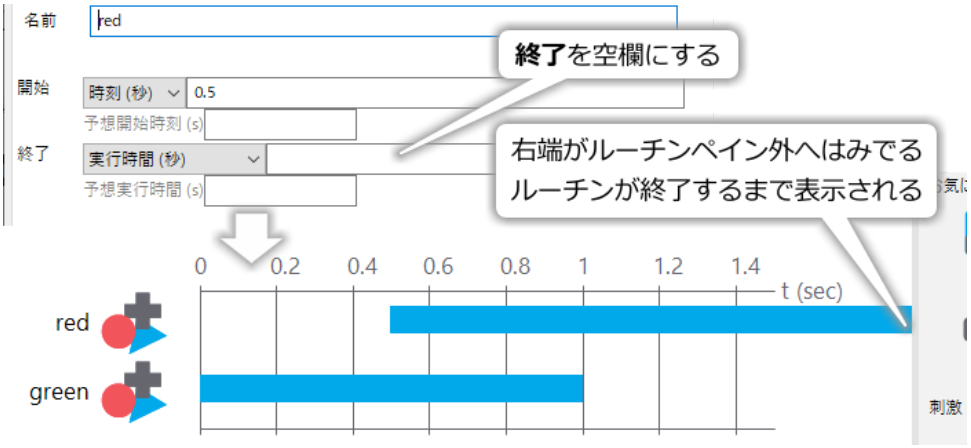


図 2.30　**[終了]** を空白にしておくと、ルーチンの終了までコンポーネントが有効になります。何らかの方法でルーチンを終了させない限り刺激は表示され続けます。標準設定では ESC キーを押すと強制的に実験を終了させることができます (ルーチンも強制終了されます)。

チェックリスト

- 刺激の表示開始時刻と表示時間を指定して表示させることができる。
- 刺激の表示開始時刻と表示終了時刻を指定して表示させることができる。
- 刺激の表示終了時刻を定めずに表示させることができる。
- 実行中の実験を強制的に終了させることができる。

2.8 Builder が作成するファイルを確認しよう

「刺激の位置や提示時間を指定する方法を覚える」というこの章の内容はほぼ終わりました。最後に実験の基本設定を行う方法を解説したいのですが、その前に Builder が作成するファイルとフォルダについて簡単に触れておきます。

ここまで作業の作業を進めた後で chapter2.psyexp を保存したフォルダを確認すると、data というフォルダと chapter2_lastrun.py というファイルができているはず。もし psyexp ファイルを chapter2 以外の名前で保存したのでしたら、_lastlan.py の前の部分が保存したファイル名に対応した文字列になっているはずです。

chapter2_lastrun.py は Builder が psyexp ファイルを「翻訳」して作成した Python のスクリプトです。メモ帳などのテキストエディタで開いてみると内容を確認することができます。Python を用いて実験をするとは元々このようなファイルを自分で書くということであり、その作業を人の代わりに行ってくれるのが PsychoPy Builder だというわけです。ただし、Builder が生成するスクリプトは人が書く場合に比べて少々冗長ですので、人が書けばもっと短いスクリプトで実現することも可能です。このファイルは実験を実行する度に自動的に作成されるので、実験終了後に削除してしまっても問題ありません。

data フォルダは、実験結果を記録したファイルが保存されるフォルダです。実験を一回実行する度に複数のファイルが作成されるので、もしここまで一気に作業してこられたのでしたら非常にたくさんのファイルが作成されているはずです。このフォルダ内のファイルには実験結果が記録されています。この章で作業した内容は特に記録する必要はありませんので、data フォルダごと削除してしまって構いません。ファイルの内容については 第 3 章 以降で詳しく見ていきます。

最後に、psyexp ファイルそのものについて少し補足しておきましょう。Builder で作成した実験の内容はすべてこのファイルに保存されていますので、実験が不要にならない限りこのファイルを削除してはいけません。psyexp ファイルの他に、第 3 章 で解説する条件ファイルも psyexp ファイルと一緒に保存しておく必要があります。刺激として画像ファイルや音声ファイルを使用する場合は、それらのファイルも忘れずに保存しておかなければいけません。なお、psyexp ファイルは XML 形式と呼ばれるデータ形式で保存されたファイルなので、メモ帳などのテキストエディタを使って開くと中身を見ることができます。

保存した psyexp ファイルは、 図 2.31 の「実験を開く」ボタンをクリックすると Builder で開くことができます。作成途中で保存した psyexp ファイルを開いたり、完成した psyexp ファイルを使って実験したりする時に使います。OS によっては psyexp ファイルのアイコンをダブルクリックするだけで自動的に Builder を起動してファイルを開くこともできます。作業を保存する時は「上書き保存」ボタン、別の名前で保存したいときには「名前を付けて保存」ボタンを使います。現在作成中の実験を置いておいて新たに実験を作成したい場合は「実験の新規作成」ボタンを使います。他にも「元に戻す」と「やり直す」ボタンも便利ですので一緒に覚えておくとよいでしょう。

図 2.31　ファイル操作に関するボタンと元に戻す・やり直すボタン。

チェックリスト

- foo_lastrun.py (foo は psyexp 実験ファイル名) の役割を説明することができる。
- data フォルダの役割を説明することができる。
- 実験結果を保存する必要がない場合、どのファイルを削除しても問題ないかを判断できる。
- 作製済みの psyexp ファイルを Builder で開くことができる。

- psyexp ファイルを別の名前で保存することができる。

2.9 実験の設定を変更しよう

これで刺激の描画方法の基礎を一通り解説しました。本格的な実験の作成に入る前に、実験の設定について解説しておきます。Builder ウィンドウ上部のツールバーの 図 2.32 に示したアイコンをクリックすると、実験設定ダイアログが開きます。このダイアログには「基本」、「データ」、「スクリーン」の三つのページがあり、非常に多くの項目が含まれています。「基本」から順番に見ていきましょう。

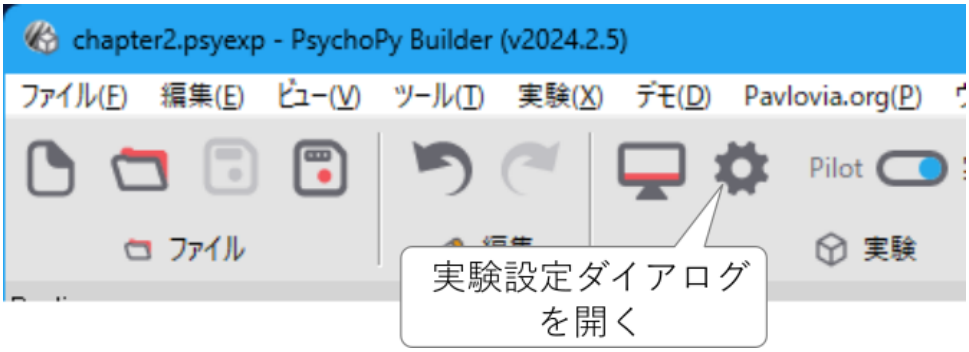


図 2.32　Builder ウィンドウ上部のリボンにある歯車のアイコンのボタンから実験設定ダイアログを開くことができます。

2.9.1 「基本」タブ

このタブでは **[実験の名前]** と **[実験情報ダイアログ]** が非常に重要です。**[実験情報ダイアログ]** は 第 4 章 以降で詳しく取り上げます。

[実験の名前]
実験の名前を入力します。実験結果の記録ファイルに反映されるため、データ整理の際に便利でしょう。ファイル名として使用できる文字列でなければいけません。日本語の文字の使用は避けた方が無難です。

[実行モード]
Builder のリボンの Run モードスイッチと同じです。実験設定ダイアログ上で変更して Builder に戻れば変更が反映されています。

[使用する PsychoPy のバージョン]
旧バージョンの PsychoPy で作成した実験が現在使用中のバージョンで動かないときに、旧バージョンで実行するように指定することができます。この機能を利用するためには、git というコマンドが使用できるように設定しておく必要があります。利用のためには他にもいろいろと条件があるので、初心者の方には利用をお勧めしません。

[ESC キーによる中断]
「 2.7: 刺激の提示開始と終了時刻の指定方法を理解しよう 」で触れた、ESC キーによる実験の強制終了を有効にするか無効にするかを指定します。実験中に実験参加者が誤って ESC キーを押してしまう恐れがある場合はチェックを外しておくべきですが、チェックを外してしまうと強制終了ができなくなりますので注意してください。実験が完成して、十分に動作確認をして問題がないことを確認してからチェックをはずすとよいでしょう。なお、フルスクリーンモードを使用していない場合は、実験実行中

に Runner のウィンドウを選択して実験中断ボタンをクリックしても強制終了することができます。

[Rush モード] この項目にチェックを入れると、実験を可能な限り高い優先度で実行しようとします。Windows にせよ MacOS にせよ、現代の OS では何も操作していなくてもいくつものプログラムが並行して動作しており、実験実行中に別のプログラムが時間を要する作業を始めると、実験の時間精度に影響する恐れがあります。並行して動作しているプログラムの間には優先順位があり、優先順位が高いものは低いものの割り込みを受けにくくなります。Rush モードを ON にすると、可能な限り実験プログラムの優先度を高めようとします。初期値では OFF になっていますが、実験の時間精度が安定しない場合は試してみる価値があります。

[実験情報ダイアログを表示] 実験実行時に表示される実験情報ダイアログの表示、非表示を指定します。チェックを外しておくと実験情報ダイアログが表示されません。

[実験情報ダイアログ] 実験情報ダイアログに表示する項目を設定します。詳しくは 第 4 章 を参照してください。

2.9.2 「スクリーン」タブ

このタブには **[フルスクリーンウィンドウ]**、**[ウィンドウの大きさ (pix) $]**、**[マウスカーソルを表示]**、**[単位]**、**[背景色]** など、画面に関する重要な設定項目がたくさんあります。

[モニター] 使用するモニターを指定します。PsychoPy ではモニターの設定に名前をつけて保存しておくことができますが、ここでは使用するモニター設定の名前を入力します。モニター設定の作成方法はこの章の「2.11: モニターを設定しよう」触れます。

[ウィンドウバックエンド] 実験を実行すると灰色の画面に切り替わりましたが、その画面の描画に使用するライブラリを指定します。初期値は pyglet で、通常は変更する必要ありません。

[スクリーン] 複数台のモニターが接続された PC を使用する場合、どのモニターを視覚刺激提示に使用するかを番号で指定します。モニター番号がわからない場合は、「スクリーン番号の表示」をクリックして各モニター上に番号を表示させて調べることができます。

[フルスクリーンウィンドウ] 刺激提示にモニターのスクリーンいっぱいに広がったウィンドウを用いるか否かを指定します。チェックが入っていると、スクリーン全体が Builder の実験ウィンドウで覆われて、他のアプリケーションやデスクトップは見えなくなります。この状態をフルスクリーンモードと呼びます。チェックを外すと、視覚刺激提示用に通常のアプリケーションのようなウィンドウが開いて、そのウィンドウ内に刺激が提示されます。一般論として、フルスクリーンモードの方が実験実行時の時間的な精度が高い傾向にあります。何らかの理由があって通常のアプリケーションウィンドウで実行したい場合を除いて、この項目はチェックしておくべきです。

[マウスカーソルを表示] この項目をチェックしておくと、フルスクリーンモードでの実験実行時にもマウスカーソルが表示されま

す。標準ではチェックされていません。マウスを用いて参加者の反応を記録する実験を実施する場合などに使います。 **「入力」タブで ioHub を選択していると、この項目のチェックを外しているにも関わらずマウスカーソルが表示されてしまう場合があります。**

[ウィンドウの大きさ (pix) $] フルスクリーンウィンドウを使用しない時に、刺激提示用ウィンドウの幅と高さを指定します。書き方は視覚刺激の大きさの指定と同様 [1920, 1080] といった具合に幅と高さの値をカンマで区切って書き、角括弧で囲みます。単位は pix です。フルスクリーンウィンドウ使用時にはこの項目は灰色に表示されていて編集できません。フルスクリーンウィンドウ使用時のスクリーンの解像度は OS による解像度の設定に従います。Pilot モード時のウィンドウの大きさは、「 2.10:Pilot モードのウィンドウサイズを設定しよう 」で示す方法で設定した大きさに従います。

[単位] 視覚刺激コンポーネントで用いられる標準の単位を指定します。具体的には、 表 2.2 に示した単位のうち「実験の設定に従う」を選択した際に使用される単位を指定します。個々のコンポーネントで「実験の設定に従う」以外の単位を選択した場合は、そちらが優先されます。単位の選択肢の中に「PsychoPy の設定に従う」という項目がありますが、これは PsychoPy 設定ダイアログ (「 2.10:Pilot モードのウィンドウサイズを設定しよう 」で触れます) で定義されている標準の単位に従うことを意味しています。筆者の率直な考えを述べると、「PsychoPy の設定に従う」を使う場面は (height が単位として導入された以降は特に) ほぼありません。しいて言えば、実験によって自分でコードを書く方法と Builder を使う方法を併用している人で、両方の方法で使用する単位を一括して変更する可能性がある場合は「PsychoPy の設定に従う」が役に立つかもしれませんが、特殊なケースだと思います。

[背景色] 視覚刺激提示画面の背景色を指定します。視覚刺激の色の指定方法と同様に、web/X11 color name や web カラー、色空間を指定した数値表現を使用することができます。

[ブレンドモード] 刺激を重ね書きした時の挙動を指定します。標準値は「平均」で、「 2.6: 刺激の重ね順と透明度を理解しよう 」で解説した通りに描画されます。「加算」にすると色が足し合わされます。「足し合わされる」といってもわかりにくいと思いますので、 **[不透明度 $]** 0.3 の赤、緑、青の円をブレンドモード「平均」と「加算」で重ね合わせた出力を 図 2.33 に示します。

「加算」の重ね合わせのほうが光の加法混色に近いですが、重ね合わせの結果、色が PsychoPy(正確には PsychoPy が描画に使用している OpenGL というライブラリ) が表現できる範囲を超えてしまった時には描画が破綻してしまいますので、実験製作者がよく考えて刺激の色を決定する必要があります。「平均」ではそのような破綻が起きることはありません。

[色空間] 背景色の指定に使用する色空間を指定します。

[背景画像] 画像ファイルを背景として使用したい場合、ここにそのファイルを指定します。画像ファイルの指定方法については 第 6 章 で詳しく解説します。画像ファイルのサイズと次の **[背景画像の伸縮]** の設定の組み合わせによってはスクリーンより画像が小さく表示されますが、その場合画像の周囲の領域は **[背景色]** で設定した色になります。

[背景画像の伸縮]

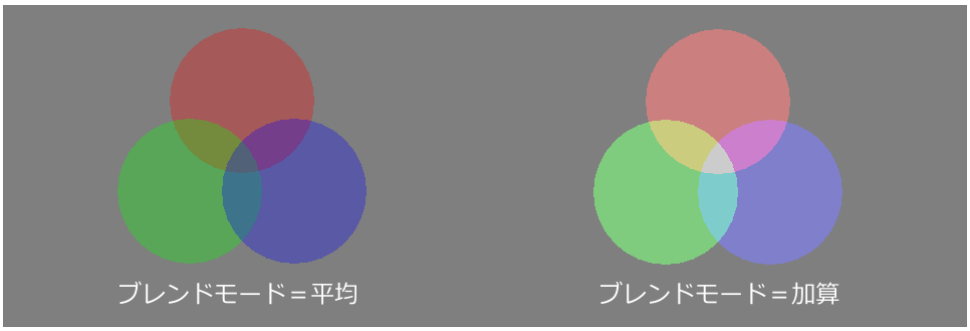


図 2.33　ブレンドモードの比較。

[背景画像] に画像ファイルが指定された場合に、画像のサイズをどのように調整するかを指定します。「なし」にすると画像ファイルのサイズそのまま、「cover」にすると画像がスクリーンに外接、「contain」にすると画像がスクリーンに内接するように拡大縮小されます。これらはいずれも元画像の縦横比を変更しません。「fill」にすると、スクリーンにぴったり一致するように必要に応じて縦横比を変更しながら画像を拡大縮小します。「scale-down」にすると、画像がスクリーンより大きい場合のみ画像がスクリーンに内接するように (つまり contains と同じ) 縮小されます。

[フレームレートの測定]　この項目がチェックされていると、実験開始時にフレームレートの測定 (1 秒間に何回画面を書き換えているか) をおこないます。実験を実行したときに Attempting to measure frame rate of screen, please wait... と表示されているのはこのフレームレートの測定をおこなっていることを示すメッセージです。フレームレートについては「 2.12.6: 時刻指定における frame について 」で詳しく解説しています。実験が始まる前に少し待たされるのを不満に思う方もいるかもしれませんが、 **この項目はチェックしたままにしておくことを強く勧めます。**

この項目のチェックの有無で次の項目の名称が変化します。

[フレームレート測定時のメッセージ] / [フレームレート]

[フレームレートの測定] がチェックされていれば **[フレームレート測定時のメッセージ]** 、チェックされていなければ **[フレームレート]** という項目になります。**[フレームレート測定時のメッセージ]** はフレームレートの測定中に画面に表示される文で、初期値は Attempting to measure frame rate of screen, please wait... です。文面を変更したい場合はここを編集してください。ただし、日本語の文字などはフォントによっては正しく表示されませんので、編集する場合でも英語のままにしておくことをお勧めします。

フレームレートの測定をおこなわない場合は **[フレームレート]** に適切なフレームレートを入力しなければいけません。たとえばモニターのスペック表で 60Hz と記されているなら 60 と入力します。ただし、実際にはさまざまな理由でスペック表のとおりに描画がおこなわれていない場合があり (だからこそ Builder の実験では最初に実測しようとする)、こちらで入力した値と実際の値がずれていたら時間精度に悪影響があることは覚悟しておいてください。

2.9.3 「オーディオ」タブ

音声関連の設定をします。すでに PsychoPy を長く使用していて、過去に使用していたライブラリと同じものを使い続けたい場合などには設定を変更する必要があるかもしれませんが、新たに使い始める人はまず変更せずに使用してみることをお勧めします。

オーディオの再生に関する設定を行います。詳しくは 第 10 章 を参照してください。

[オーディオライブラリ]
オーディオの再生に使用するライブラリを選択します。ptb は Matlab のパッケージとして有名な PsychToolbox を移植したもので、2025 年 2 月現在で標準のオーディオライブラリとされています。pyo、sounddevice、pygame は過去に使用されていたライブラリで互換性のために残されています。

[オーディオ遅延の優先度]
PsychToolbox のオーディオを使用する場合の再生遅延の設定をおこないます。0 から 4 まで 5 段階あり、数字が小さいほど幅広いハードウェアで再生できますが遅延が長くなります。

[ステレオを強制]
Sound コンポーネント (第 10 章) でステレオ再生するように設定します。とりあえずそのままにしておいて問題ないでしょう。

2.9.4 「オンライン」タブ

インターネット上でオンライン実験を行うための設定を行います。ローカルで実験を行う場合は関係ありません。

[出力パス]
サーバーへアップロードする HTML 形式の実験ファイルを出力するフォルダ名を指定します。 **通常、ここは空欄にしておいてください。**

[HTML 形式でエクスポート]
HTML 形式の実験ファイルを出力するタイミングを指定します。「同期時」ならサーバーと同期するとき、「保存時」なら保存するとき、「手作業で」なら手動で出力します。

[正常終了時の URL] 実
験が正常に終了した時に表示するページの URL を指定します。

[中断時の URL] 実
験が中断された時に表示するページの URL を指定します。

[終了メッセージ] 実
験が正常に終了した時に表示するメッセージを指定します。

[追加リソース] 実
験で使用する画像ファイルや音声ファイルのうち、Builder が自動で見つけられないものをここに追加しておくと Builder が認識してくれます (第 4 章 のテクニックで実行時に読み込むファイルを決定する場合などに便利です)。

2.9.5 「アイトラッキング」タブ

Builder に組み込みの各種市販アイトラッカーとの連携機能に関する設定をおこないます。使用するには対応するアイトラッカーが必要です。

[アイトラッカーデバイス] 使用するアイトラッカーを選択します。デフォルトは None(使用しない) です。選択したアイトラッカーに応じて必要な設定項目が表示されます。ここではマウスによってアイトラッカーの動作をシミュレートする MouseGaze を選択した場合の項目を解説します。

[Move ボタン] マウスカーソルの動きをどのように視線の動きに変換するかを指定します。CONTINUOUS ならマウスカーソルの位置がそのまま視線位置となります。LEFT_BUTTON、MID_BUTTON、RIGHT_BUTTON のいずれかを選択すると、選択したボタンをクリックしたときにその位置へ視線が移動します。

[Blink ボタン] 瞬目をシミュレートするボタンを選択します。

[サッカード閾値] サッカード検出のための閾値を視線の移動量 (単位:deg) で指定します。deg 単位が有効になるためにはモニターの設定でモニターの幅と観察距離が設定されていないといけない点に注意してください。

2.9.6 「入力」タブ

[キーボードバックエンド]
キーボードのキー押し検出に使うライブラリを選択します。PsychToolbox は時間精度の高さが特徴です。ioHub は PsychToolbox と比べると機能的にやや劣りますが、キーボードやマウス以外の入出力デバイスを用いるのなら ioHub の方がよいかもしれません。Pyglet は PsychToolbox、iohub には劣りますがこれらのライブラリを利用できない環境でも使うことができます。

なお、この項目を ioHub に設定している場合、「9.7: カーソルの位置を設定し、カーソルの表示 ON/OFF を制御しよう 」で紹介する方法でマウスカーソルを非表示にできない場合があります (2022.2.4/Win11 で確認)。

2.9.7 「データ」タブ

データの保存形式やログに関する設定をおこないます。データファイルについては 第 3 章 で解説します。通常は変更する必要はありません。

[データファイル名 $] 実験結果を記録したファイルの名前を決定する規則を Python の式で入力します。通常の用途では変更する必要はないはずです。

[データファイルの区切り文字]
データファイルで列を区切る文字を指定します。auto にするとファイル名から自動的に区切り文字を決定します。他にはコンマ、セミコロン、タブを指定できます。

[列の並び替え...]
データファイルで各パラメータが出力される列の順序を選択します。「追加順」は実験実行時に当該パ

ラメータが最初に記録されるタイミング順、「アルファベット順」は文字通りパラメータ名の順、「優先度」は次の **[列の優先度]** の設定に従って順序を決定します。

[列の優先度]

「列」にパラメータ名、「優先度」に数値を設定することによって列の並び替え順を制御することができます。priority.CRITICAL, priority.HIGH, priority.MEDIUM, priority.LOW, priority.EXCLUDE といった定数があらかじめ用意されています (それぞれ 30, 20, 10,0, -10 に対応していますが将来的に変更されるかもしれません)。筆者が確認した範囲では、同一の優先度を持つパラメータはアルファベット順に出力されます。

[xlsx 形式のデータを保存]

Excel の xlsx 形式で実験結果を記録します。詳しくは 第 3 章 を参照してください。

[ログの保存]

PsychoPy の動作状況をログファイルに記録します。

[CSV 形式のデータを保存 (summaries)]

CSV 形式で実験結果の要約を記録します。詳しくは 第 3 章 を参照してください。

[CSV 形式のデータを保存 (trial-by-trial)]

CSV 形式で実験の全試行の結果を記録します。詳しくは 第 3 章 を参照してください。

[pydat 形式のデータを保存]

Python の pydat 形式で実験結果を記録します。チェックを外すことはできません。

[hdf5 形式のデータを保存] ア

イトラッカーなどの、iohub を介して連携する一部の装置の測定データは hdf5 形式で記録されます。ここでは詳しい解説を省略します。

[ログレベル (ファイル)] ロ

グファイルに出力される内容を指定します。レベルには error から debug まで 6 段階あり、error が最も簡潔、debug が最も詳細です。初期値は info です。

[ログレベル (コンソール/アプリ)]

Runner の「標準出力」などに出力される内容を指定します。レベルの段階は **[ログレベル (ファイル)]** と同一で、初期値は warning です。

[時刻のフォーマット] ロ

グファイルに出力される時刻の表記方法を指定します。初期値は「実験開始時から」で、実験開始時刻を 0.0 とした経過時間で表記します。 ストップウォッチのようなものを想像してください。「実時間」を選択すると、PC の時計にしたがって 年月日 時分秒 の形式で表記します。

以上で実験設定ダイアログの概要の説明は終了です。とりあえず設定は変更せず初期値のまま使ってみることをお勧めしますが、

- 「スクリーン」タブの **[フルスクリーンウィンドウ]** のチェックを外して Pilot モード、実行モードの両方のモードで実験を実行してみる。
- 「スクリーン」タブの **[背景色]** を指定して背景色を変える。

- 「スクリーン」タブの **[マウスカーソルを表示]** のチェックを外した状態で「入力」タブの **[キーボードバックエンド]** を ioHub にして実行し、皆さんの実行環境でマウスカーソルが表示されないか確認する。表示されてしまう場合は **[キーボードバックエンド]** を PsychToolbox にして確認する。面倒だが **[マウスカーソルを表示]** をチェックした状態で ioHub、PsychToolbox での実行した場合も確認しておくとよい。

の 3 つは各自で実際に試してみてみるとよいでしょう。

チェックリスト

- 実験設定ダイアログを開くことができる。
- 実験開始時に実験情報ダイアログを表示させるか否かを設定することができる。
- 登録済みのモニターのうちどれを実験に使用するかを実験設定ダイアログで設定できる。
- 実験をフルスクリーンモードで実行するか否かを設定することができる。
- フルスクリーンモードを使用しない時に、視覚刺激提示ウィンドウの幅と高さを指定できる。
- 視覚提示ウィンドウの背景色を指定できる。
- 「実験の設定に従う」で参照される単位を指定することができる。
- ESC キーによる実験の強制終了を有効にするか無効にするかを指定することができる。
- 実験記録のファイルを保存するフォルダ名を指定することができる。
- フルスクリーンモード時にマウスカーソルを表示するか否かを指定することができる。

2.10 Pilot モードのウィンドウサイズを設定しよう

前節で確認した Builder の実験設定ダイアログに加え、PsychoPy には PsychoPy 設定ダイアログというものもあります。Builder の実験設定ダイアログは Builder 以外の方法で実験を作成する場合には無関係ですが、PsychoPy 設定ダイアログの内容は PsychoPy の動作全体に関わってきます。本書では PsychoPy 設定ダイアログの内容に触れませんが、Pilot モードのウィンドウサイズが標準では 800 × 600 しかなく、高解像度のモニターで見ると非常に小さくなってしまうので、この問題に関する項目だけ解説しておきます。

PsychoPy 設定ダイアログは 図 2.34 の左ように Builder のメニューの「ファイル」から開きます。ダイアログの「Pilot モード」を開くと「ウィンドウサイズを強制」という項目がありますので、1280, 720 のように幅、高さをカンマ区切りで入力してください。実験本番の時と同じサイズにするもよいでしょうし、やや小さめにしておいて背後にあるウィンドウにアクセスしやすいようにするのもよいでしょう。小さめにする場合、モニターの解像度が 1920 × 1080 (縦横比 16:9) なら 1280 × 720 にするなど、 **同じ縦横比にしておくと単位 height で実験を作成するときに視覚刺激がきちんと画面内におさまるか確認できる** のでお勧めです。各自の環境に応じて値を設定してください。

なお、「 1.2:PsychoPy をインストールしよう (Standalone インストーラー編) 」で触れた通り、PsychoPy 2024 系ではメニューなどが英語で表示される問題があり、「1.4.2: ロケールの変更 」に修正方法を示しましたが、この修正を適用してもなお PsychoPy 設定ダイアログの項目は英語で表示されてしまいます。その場合、「ウィンドウサイズを強制」は forcedWindowSize という名称で表示されます。「Pilot モード」のページには 800, 600

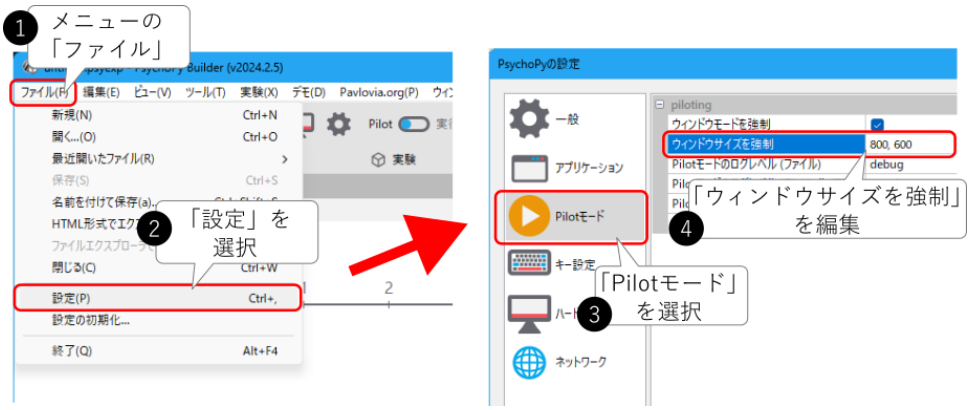


図 2.34　Pilot モードのウィンドウサイズの設定。Builder のメニューの「ファイル」から PsychoPy 設定ダイアログを開いて設定します。

のように 2 つの数値を並べて記入する項目はこれひとつしかありませんので (バージョン 2024.2.5 現在) すぐにわかると思います。

チェックリスト

- Pilot モード時のウィンドウサイズの設定を変更できる。

2.11 モニターを設定しよう

長くなりましたが、本章の最後の話題です。deg や cm を単位として使用できるように、あなたが使用しているモニターを PsychoPy に登録しておきましょう。

モニターを登録するには、ツールバーの 図 2.35 に示したモニターセンターダイアログを開くボタンをクリックします。メニューの「ツール」の「モニターセンター」からも開くことができます。開いたダイアログの左上に登録されたモニターの一覧が表示されており、その横の「新規…」ボタンで新たなモニターを登録、「保存」ボタンで変更の保存、「削除」ボタンで登録の削除を行います。登録モニター一覧の下にある日付のようなリストは、選択中のモニターに対するキャリブレーションデータの一覧を示しています。簡単に言えばモニターのキャリブレーションとは、PC 上では数値によって表されている色を、モニターが正確に表現できるように調整することです。キャリブレーションには専用のセンサーが必要なので、この本では扱いません。この本で作成する実験は、ブラウザでインターネット上のニュースの写真などを閲覧して、特に違和感を覚えない程度に色が表示できていれば問題なく実行できます。

モニターの寸法や観察距離を設定するには、モニターを選択して左下の「モニター情報」と書かれた枠内に数値を入力します。ここでは、新しいモニターを登録して設定を行うことにしましょう (図 2.36)。登録モニター一覧の右の「新規…」をクリックしてください。モニターの名前を登録するダイアログが表示されるので、My Monitor と入力しておきます。OK をクリックすると、モニター一覧に My Monitor という項目が追加されているはずです。My Monitor が選択されていることを確認して、左下の「モニター情報」という枠内の「スクリーンの観察距離 (cm)」、「解像度 (ピクセル; 水平, 垂直)」、「スクリーンの横幅 (cm)」に適切な値を入力します。 図 2.36 では 図 2.11 と同じ 1920 × 1080pix、幅 51.0cm のスクリーンを持つモニターを入力しています。皆さんは各自が使用しておられるモニターの数値を入力してください。PsychoPy では画素の縦横の長さは同一として計算しているので、幅だけを入力すれば高さは解像度と幅から自動的に計算されます。観察距離は 図 2.36 の例では 57.3cm としておきました。終了したら登録モニター一覧の「保存」をクリックして保存して、モニターセンターのダイアログを閉じてください。保存せずに閉じようとすると変更を保存するか尋

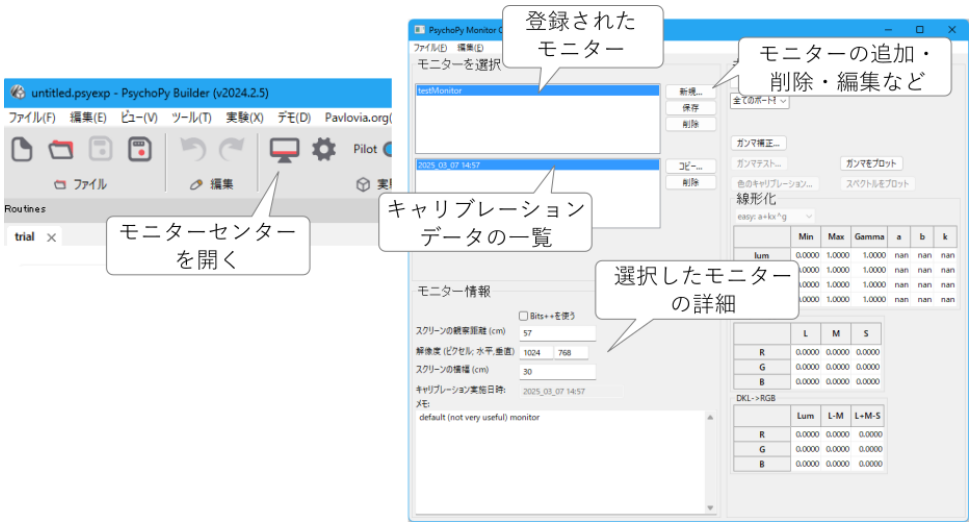


図 2.35　モニターセンターのダイアログ。モニターの登録や削除、設定の変更などができます。右半分はセンサーを用いたキャリブレーション時に使用します。

ねられるので、保存しておきましょう。

モニターの登録が終わったら、実験設定ダイアログを開いて「スクリーン」のページの「モニター」に My Monitor と入力しましょう。そして、ルーチンペインに Polygon コンポーネントを配置し、**[サイズ [w, h] $]** を [5, 5]、**[空間の単位]** を cm にして実行してみましょう。正しく設定されていれば、一辺の長さ 5cm の正方形のスクリーン上に表示されます。ぜひ定規で測って確認してください。刺激がすぐ消えてしまって測れないという方は、刺激の表示時間を長くしましょう。

確認ができたら、Polygon コンポーネントのプロパティ設定ダイアログを開いて **[空間の単位]** を deg にして実行しましょう。観察距離 57.3cm (≒ 180/π) の時には、1deg がほぼ 1cm となりますので、画面上ではやはり一辺約 5cm の正方形が表示されているはずです。それを確認したらモニターセンターへ戻って、My Monitor の観察距離を 30cm に変更してから実験を実行してみましょう。そうするとスクリーンに表示される正方形の一辺は 5cm より短くなったはずです。観察距離が短くなると刺激は網膜に大きく映るので、視角 5deg にするためには刺激を縮小しなければいけません。この縮小作業を PsychoPy が自動的に行ってくれたのです。さらに観察距離 30cm のままで Polygon コンポーネントを編集して **[空間の単位]** を cm に戻してみましょう。単位が cm の場合には、観察距離に関わらず常に一辺 5cm の正方形が表示されるはずです。

これで準備は完了です。次章ではいよいよ最初の実験を作成してみましょう。

チェックリスト

- モニターセンターを開くことができる。
- モニターセンターに新しいモニターを登録することができる。
- モニターの観察距離、解像度、スクリーン幅を登録することができる。

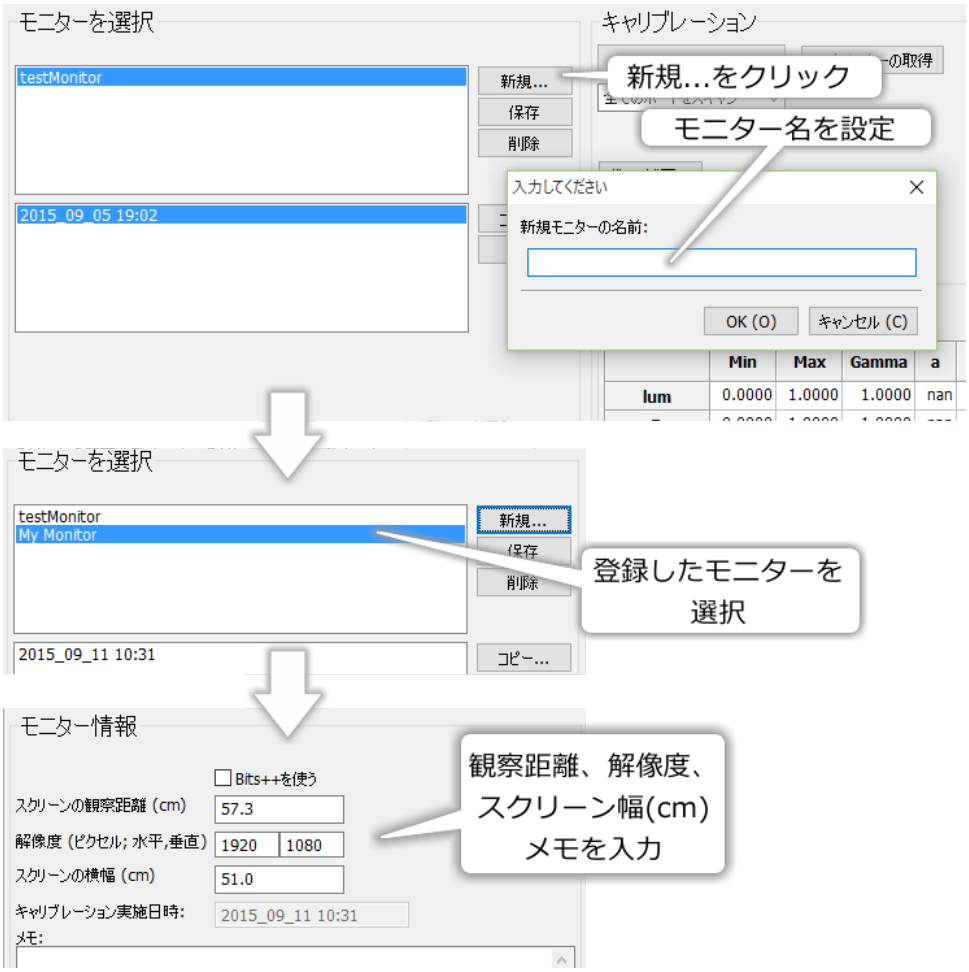


図 2.36　モニターの登録手順。

2.12 この章のトピックス

2.12.1 スクリーン左下の座標についての厳密な議論 (上級)

本文では、(1920, 1080) の解像度を持つモニターで右上の座標は (960, 540)、左下の座標は (-960, -540) と述べました。しかし、じっくり考えてみれば変です。水平方向の解像度が 10 しかない小さな小さなモニターを考えてみましょう。10 を半分にすると 5 ですから左端の座標を-5 として、-4、-3…と数えていくと、10 ピクセル目は 5 ではなく 4 です。したがって、このモニターの左端の X 座標が-5 なら右端の X 座標は 4 でなければいけません。同じように、水平解像度が 1920 のモニターで左端の X 座標を-960 にしたのなら、右端は 960 ではなく 959 でなければならないのです。

コンピュータ上で符号付きの整数を扱う場合、一般的には負の数の方が絶対値が 1 大きくなるように範囲を定めます。しかし、筆者の実行環境 (Windows10, Python x64, PsychoPy 3.0.5) で画面上に 1 × 1 ピクセルの刺激を描いて確認したところ、右上が (960, 540)、左下が (-959, -539) となり、正の方向に絶対値が 1 大きくなっていました。ただし、この結果はすべての実行環境、すべてのバージョンで保証されるとは限りませんので、参考程度にとどめてください。一般的な実験ではこの 1 ピクセルの差が問題となることはないと思われますので、本文ではこれ以降もスクリーンの中心を (0, 0)、解像度を 2 で割った値をスクリーン端の座標として記します。

2.12.2 PsychoPy における視角の計算について

視角による視覚刺激の位置や大きさの表現は、視知覚の実験などでは欠かせないものです。本文中で述べた通り、PsychoPy では視角による表現のために deg という単位が用意されていますが、その実装は近似計算です。PsychoPy Coder を使うと PsychoPy に刺激の位置などの単位を deg、pix、cm の間で相互に変換することができるのですが、それを利用して観察距離 55cm のモニターで 1.0deg を cm に変換すると 0.96cm という結果が得られます。これは 55cm × tan(π/180) の計算結果と等しいです。同様に PsychoPy に 10.0deg を cm に変換させると、9.60cm という結果が得られます。一方、55cm × tan(10 π/180) の計算結果は 9.70cm となり、9.60cm と一致しません。

これは、PsychoPy が deg を cm に変換する時に毎回三角関数の計算を行わずに、スクリーン中央から 1.0deg の位置を cm に変換した時の値を C として、X deg を cm に変換する時には C × X で近似計算しているために起きる現象です。X が小さいときはとてもよい近似値が得られるのですが、画面の中央から遠ざかるほど誤差が大きくなります (図 2.37)。先の例では画面中央から 10deg 離れた位置で 0.1cm しか異なりませんので、多くの実験では問題になることはないと思われます。

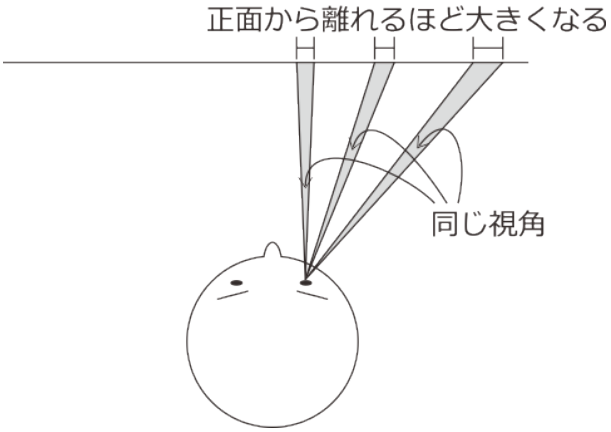


図 2.37　平面モニターを使う場合、視角を一定のまま刺激を端に移動させるとモニター画面上では刺激を大きくする必要があります。

しかし、極めて正確な刺激の描写が必要な実験や、最近増えてきた大型モニターを両端いっぱいまで使って刺激を描画するような実験では、この誤差が問題となる可能性があります。この問題に対応するために、PsychoPy 1.80 では deg より厳密な計算を行う degFlat、degFlatPos という単位が追加されました。degFlat では、図形の頂点座標を視角の定義通りに計算します。先ほどの観察距離 55cm で 10.0deg の例でも正しく 9.70cm という換算値が得られます。

defFlatPos は deg と degFlat の中間のような計算で、図形の頂点座標は deg と同様に計算し、刺激を配置するときの位置のみを正確に計算します。degFlat と degFlatPos の違いを図で示したのが 図 2.38 です。この例では deg, degFlat, degFlatPos を用いて傾いた正三角形を (0.0, 0.0) の位置から (1.0, 0.5) ずつ動かしながら (24.0, 12.0) の位置まで描画しています。白が deg、赤が degFlat、青が degFlatPos で描いたものです。deg で描いた白い三角形はすべて同じ形で等間隔に並んでいます。一方、degFlat で描いた赤い三角形は、右上に向かうにつれて間隔が広がり、三角形は大きくなり形が歪んでいます。三角形の頂点座標をすべて視角の定義に基づいて計算しているのでこのような結果となります。それに対して青で描かれた degFlatPos では、三角形の間隔こそ degFlat と同様に右上に向かうにつれて広がっていますが、三角形の大きさや形状は一定のままです。これが「頂点座標は deg と同様に計算し、配置するときの位置のみを正確に計算する」という意味です。

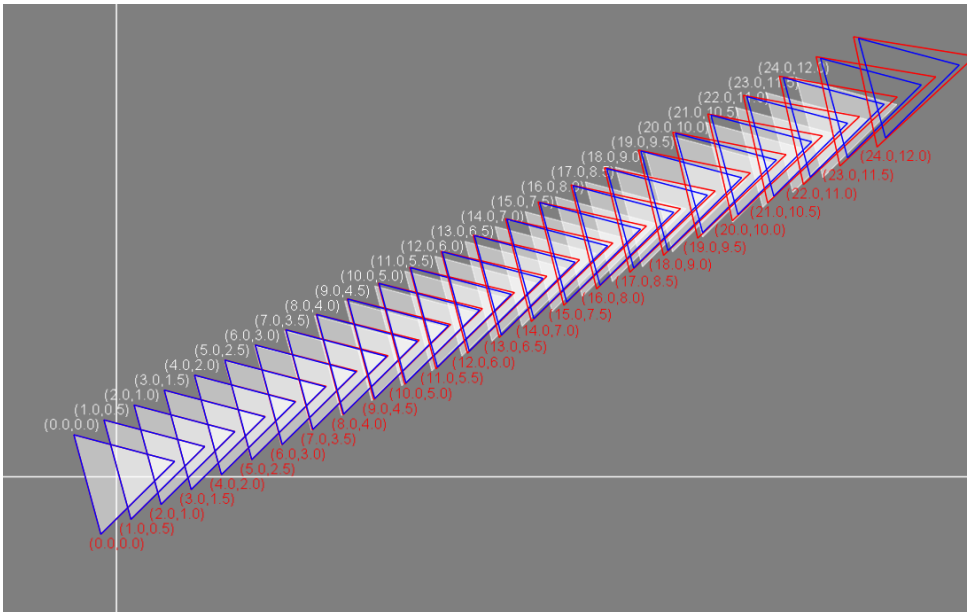


図 2.38　deg(白)、degFlat(赤)、degFlatPos(青) の違い。

2.12.3 Builder の設定ダイアログで用いられるフォント

PsychoPy Builder の標準のフォント設定では、実行環境によっては **[位置 [x, y] $]** のような $ が付くプロパティ値の小数点とカンマが非常に見分けにくいです。例えば、かつて Windows10 では 図 2.39 のようにカンマが非常に小さく表示されてしまい、小数点と見間違えることがありました。PsychoPy のウィンドウ上部のメニュー「ファイル」の「設定」を選び、表示されるダイアログの「Coder」タブをクリックして、「コード用フォント」や「コード用フォントサイズ」を変更すると見やすくなります。見分けにくくて苦労している人はぜひ試してください。

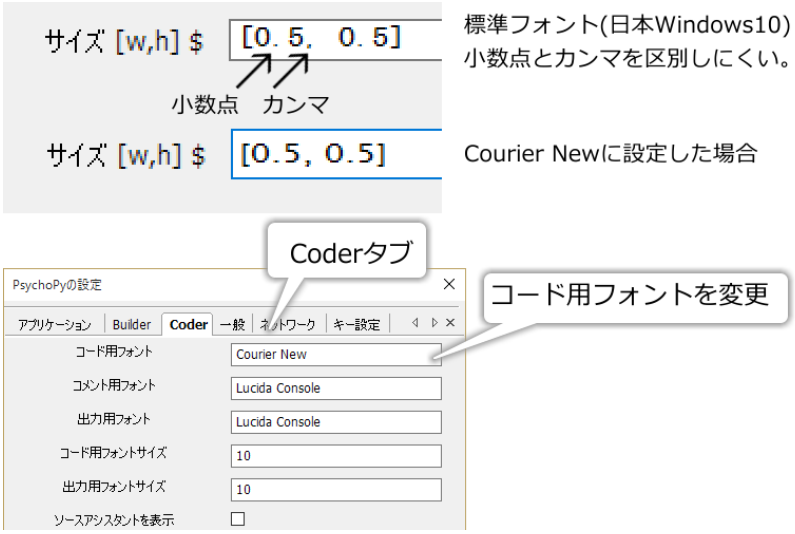


図 2.39　コード用フォントの変更。

2.12.4 Mac 上で日本語の文字が欠けてしまう場合の対策

MacOS では日本語フォントの仕様がしばしば変更されるようで、Text コンポーネントでの日本語の表示が乱れてしまう現象が度々生じています。ほとんどの場合、「書式」タブの **[フォント]** に適切なフォント名を指定すると解決します。残念ながら「こう書いたら必ず解決する」という表記方法はないのですが、「游ゴシック体」のような日本語でのフォント名か、「Hiragino Sans」のような英語でのフォント名を指定すると解決するケースが多いです。

2.12.5 16 進数と色表現

私たちが普段使い慣れている数は 10 進数でしょう。10 進数では 0、1、2、3、…と値が増加して、9 の次の整数は桁がひとつ上がり 1 の位は 0 に戻って「10（イチゼロ）」と表記されます。同様に考えると、例えば 8 進数とは 7 の次の整数になるときに桁がひとつ上がって 1 の位が 0 に戻る表記だということになります。0 から 10 までの整数を 8 進数で書くと 0、1、2、3、4、5、6、7、10（イチゼロ）、11（イチイチ）、12（イチニ）…となります。最後の 10（イチゼロ）、11（イチイチ）、12（イチニ）は 10 進数の 8、9、10 に対応しています。

頭がこんがらがってきますが、同様に 16 進数の表記を考えてみると困ったことが起きます。0、1、2、…、9 と来て、その次の整数を示す文字がアラビア数字には無いのです。仕方がないので、多くのプログラミング言語では 9 の次の整数を示す文字としてアルファベットの A、さらにその次の整数を表す整数として B、という具合にアルファベットを割り当てます。この調子でアルファベットを使っていくと、10 進数の 15 が F となり、10 進数の 16 が 10 となって 16 進数表記が完成します。Python では、「10」と書かれた時に 10 進数の 10 を表しているのか 16 進数の 16 を指しているのかを区別するために、16 進数表記の数には先頭に「0x」を付けることになっています。つまり、ただ「10」と書いてあれば 10 進数の 10 であり、「0x10」と書いてあれば 16 進数の 16 というわけです。

さて、なぜ web カラーによる色の表現に 16 進数が用いられるかということですが、web カラーでは赤、緑、青 (RGB: Red, Green, Blue) の光の強度をそれぞれ 0 から 255 の 256 段階で指定するからです。例えば RGB をそれぞれ 100 段階 (0 から 99) で変化させられる機械があるとすると、6 桁の 10 進数を用いて左から 1 桁目と 2 桁目を R、3 桁目と 4 桁目を G、5 桁目と 6 桁目に B を割り当てれば、この機械で使用できる色を 6 桁の 10 進数の数値としてわかりやすく表現することができます。この割り当て方法を 6 桁の 16 進数に適用すると、左側から 2 桁ずつ 16 進数の値を R、G、B に割り当てる形になります。2 桁の 16 進数というと 0x00 から 0xFF ですが、この範囲を 10 進数に換算すると 0 から 255 になりますので、RGB の各成分が 256 段階となる機械にとって 16 進数による表現は非常に相性がよいはずです。

具体的な数値で考えてみましょう。赤色は左の 2 桁の数値が 10 進数の 255 に対応する 16 進数 0xFF で、残りの桁は 0 となります。すなわち、0xFF0000 です。同様に、緑色は 0x00FF00、青色は 0x0000FF です。黄色の場合は、赤と緑の混色ですから赤に対応する桁と緑に対応する桁が 0xFF で青に対応する桁が 0 である値、すなわち 0xFFFF00 です。これらの色に対応する値を 10 進数で表記すると、赤が 16711680、緑は 65280、青は 255、黄色は 16776960 です。RGB の各成分を 256 段階で表現する機械においては、10 進数表記より 16 進数表記の方がはるかに直感的に RGB の強度が把握しやすいことがお分かりいただけるのではないかと思います。

なお、Python では数値が 16 進数表記であることを示すために 0x を先頭に付けますが、web ページを記述するのに用いられる HTML という言語では#FF0088 のように先頭に「#」を付けて 16 進数を示します。PsychoPy の色指定では「0x」も「#」も両方使用することができます。HTML では RGB のそれぞれに 2 桁ではなく 1 桁の 16 進数を割り当てて、#7FA のように 3 桁の 16 進数で色を表現することもできます。PsychoPy はこの

表現もサポートしています。3 桁の 16 進数が与えられた場合、PsychoPy は内部で例えば#7FA を#70F0A0 にするという具合に 0 を挿入することで 6 桁の表現に変換します。

2.12.6 時刻指定における frame について

ご存知の方も多いと思いますが、PC のモニターは 1 秒間に数十回も静止画をスクリーンに表示することによって滑らかな動きを表現しています。モニターがスクリーンを 1 秒間に書き換える回数をリフレッシュレートと呼びます。リフレッシュレートの単位は Hz です。一般的な PC 用モニターのリフレッシュレートは 60Hz 前後です。60Hz のモニターであれば、1 秒間に 60 回の書き換えを行います。高速に書き換えられる個々の静止画をフレームと呼びますが、このフレームという用語を用いると 60Hz のモニターは 1 秒間に 60 フレームを表示するという事もできます。フレームの書き換え間隔は一定ですので、1 フレームの表示時間は 1 秒÷ 60 回=0.0166... 秒、四捨五入して 16.7 ミリ秒です。

16.7 ミリ秒に 1 度しか書き換えが行われないということは、例えば Builder で刺激の表示時間を設定する際に **[終了]** で「実行時間 (秒)」を選択して 0.04 (=40 ミリ秒) を入力しても、40 ミリ秒は 16.7 ミリ秒の整数倍ではないので、実際に刺激が表示されている時間は 40 ミリ秒にはならないということです。Builder と 60Hz のモニターを用いて実際に動作確認してみると、「実行時間 (秒)」が 0.04 の時には 33.3 ミリ秒しか刺激は提示されていません (PsychoPy 1.79.01 で確認)。33.3 ミリ秒は 2 フレームに相当します。刺激提示開始時刻や終了時刻、提示時間は 1 ミリ秒 (もしくはそれ以下の) 単位で指定できますが、実質的にはフレーム単位でしか制御できていないのです (図 2.40)。言い換えると、刺激の提示時間としてフレームの表示時間の倍数を指定した時以外は、設定した時間と実際に表示されている時間の間には必ずズレが生じているのです。

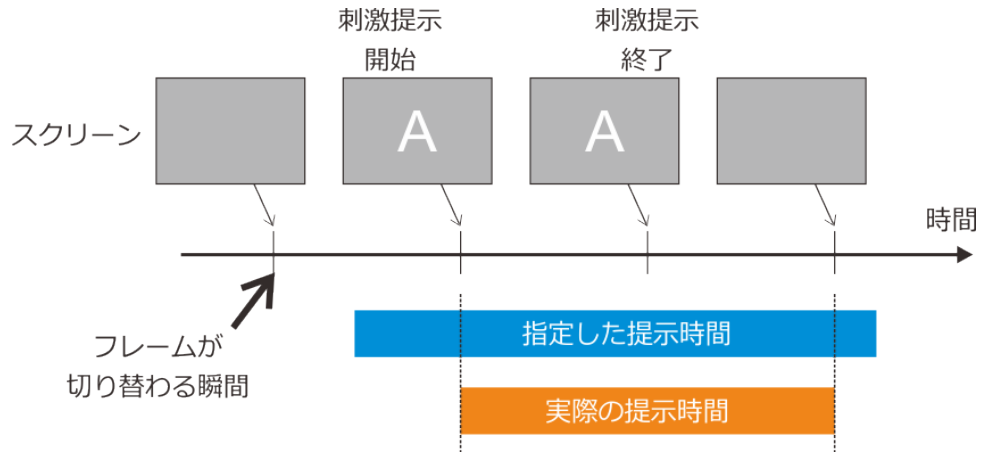


図 2.40　提示時間がフレーム表示時間の倍数になっていなければ指定した時間と実際の提示時間の間にずれが生じます。

どうせフレーム表示時間の倍数でしか正確に刺激提示時間を指定できないのでしたら、いっそのこと秒単位ではなくフレーム数で刺激提示時間を指定した方がわかりやすいという考え方もあるでしょう。フレーム数による指定を可能にするのがプロパティ設定ダイアログの **[終了]** で選択できる「実行時間 (フレーム数)」という選択肢です。「実行時間 (フレーム数)」を選択すると、指定された数のフレームを表示する間刺激を提示します。当然、正の整数を指定しなければ意味がありません。同様に **[開始]** 、 **[終了]** でともに選択できる「フレーム数」を用いるとルーチンが開始されてから何フレーム目に刺激提示を開始、終了するかを指定することができます。100 ミリ秒未満の短時間の刺激を提示する場合や、特に正確な提示時間の制御が必要な実験をする場合はフレームによる指定が有効です。

なお、フレームで刺激提示時間を指定すると、ルーチンペインの青い棒が表示されなくなってしまいます。Builder は実験に使用されるモニターのリフレッシュレートを知らないので、何秒から何秒まで刺激が提示さ

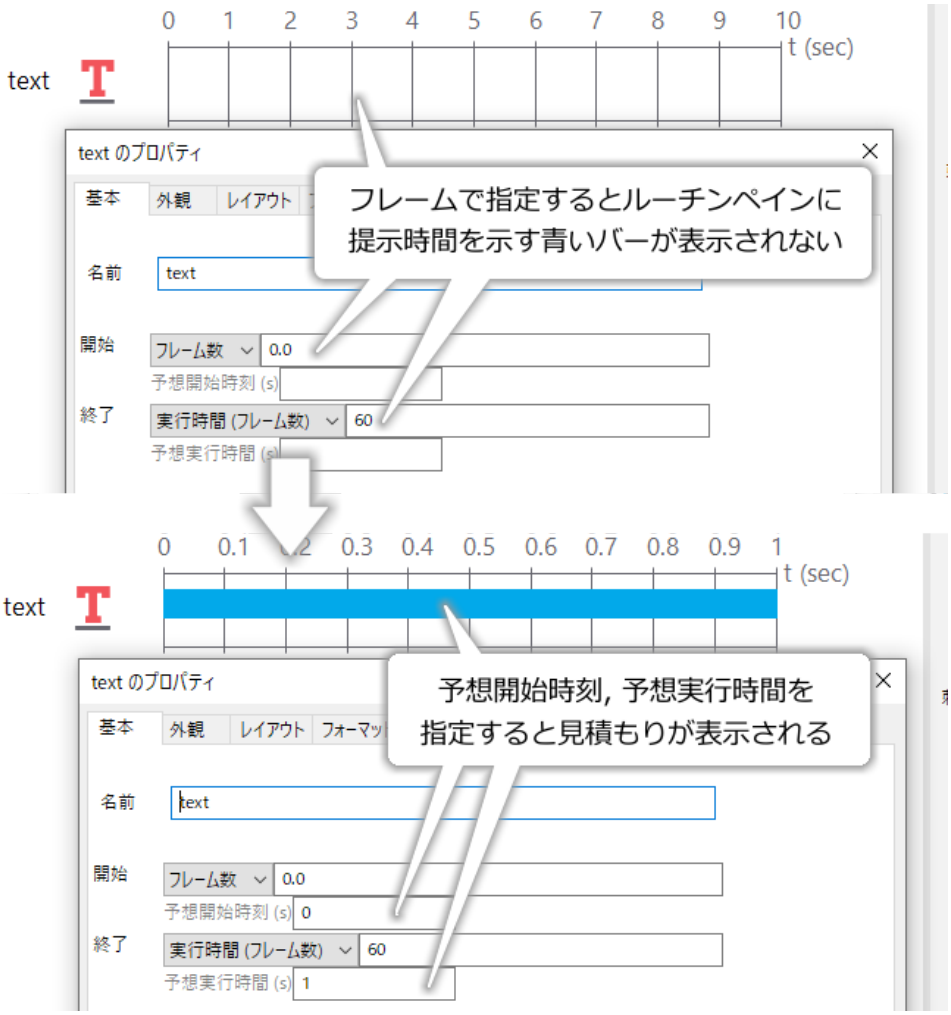


図 2.41　フレームで刺激の提示時間を指定するとルーチンペイン上で提示時間を示す青い棒が表示されません。**[予想開始時刻 (s)]**、**[予想実行時間 (s)]** に見積もり値を入力することでルーチンペイン上に提示時間を表示することができます。

れているかを計算することができないのです。このままではルーチンペインを見たときに刺激がどのような順番に提示されていくのか非常にわかりづらいので、Builder には「この時刻からこの時刻まで提示される」という目安を表示させる機能があります。Polygon コンポーネントや Text コンポーネントを配置してプロパティ設定ダイアログを確認してください。灰色の文字で **[開始]** の下に **[予想開始時刻 (s)]**、**[終了]** の下に **[予想実行時間 (s)]** と書かれた項目があります。ここに開始時刻と提示時間の見積もりをそれぞれ入力すると、見積もりに従って青い棒が表示されます (図 2.41)。見積もり値は、各自で使用しているモニターのリフレッシュレートと指定したフレーム数から計算する必要があります。ちなみに「時刻 (秒)」や「実行時間 (秒)」を選択している時にもこれらの見積もりを入力することができますが、混乱を招くだけで意味はありません。

蛇足ですが、リフレッシュレートと非常によく似た用語でフレームレートというものがあります。フレームレートの単位は frames per second の略で FPS です。「1 秒あたりのフレーム数」という意味ですから、リフレッシュレートと同じ意味のように思えます。しかし、フレームレートという用語は通常「PC がモニターに対して 1 秒間に表示するように要求したフレーム数」を指します。フレームレートが 100FPS に達しても、使用しているモニターのリフレッシュレートが 60Hz であれば 1 秒間に実際に表示されるフレーム数は 60 枚です。間違えやすいのでご注意ください。

第 3 章

最初の実験を作ってみよう―サイモン効果

3.1 実験の手続きを決めよう

この章では、最初の実験としてサイモン効果の実験を作ってみましょう。サイモン効果について詳しい解説は認知心理学の教科書などを参考にしていただくとして、ここでは実験の手続きを説明します。

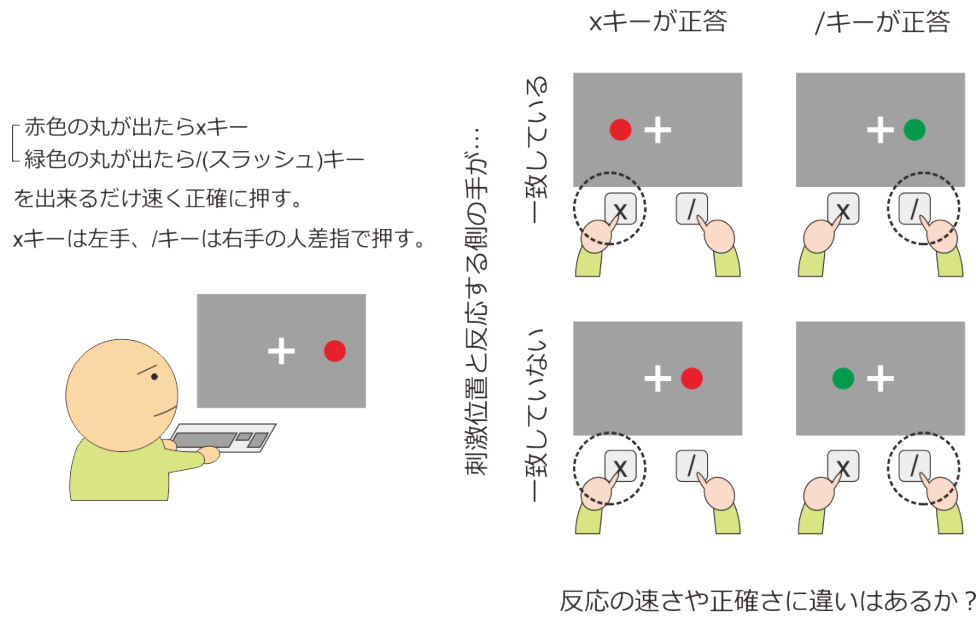


図 3.1　サイモン効果の実験概要。実験参加者は刺激の色に対応するキーをできるだけ速く正確に押すように求められます。スクリーン上の刺激の位置と反応するキーの位置関係が反応の速さや正確さに及ぼす影響を検討します。

図 3.1 をご覧ください。実験参加者に与えられた課題は、PC のスクリーン上に提示された刺激の色に応じて、PC のキーボードのキーを指定された指で押すことです。参加者はスクリーン中央の十字 (固視点) に視線を向けて、刺激が出現するのを待ちます。刺激は赤色または緑色の円で、固視点の右側または左側のどちらかに出現します。参加者は刺激の色が赤色であれば左手の人差し指で x のキー、緑色であれば右手人差し指で / (スラッシュ) のキーをできるだけ速く、間違わないように押さないといけません。左側に赤色の刺激が出現した場合と右側に緑色の刺激が出現した場合には刺激と反応する指が同じ側になりますが、それ以外の場合は刺激が出現した側と反対側の指で反応しないといけません。刺激が提示される位置と反応する指の関係が反応の速さや正確さにどのように影響するかを確認しようというわけです。

さて、実際に実験を作成するためには、図 3.1 の内容よりもっと詳細に手続きを決定する必要があります。図 3.2 は刺激提示スクリーンの時間的な変化を図示したものですが、このように書いてみると 図 3.1 の内容では x1 から x5 の値が決まっていないことがわかります。何かの文献に基づいて実験を作成しているのであれば文献に書かれている手続きを熟読して値を決めることになるでしょう。この章では、あまり内容を高度にし過ぎないためにあらかじめ以下のように値を決めることにします。

- x1 固視点の大きさは縦横 0.05(位置と大きさの単位はすべて height)。
- x2 試行が始まってから刺激が出現するまでの時間は 1.0 秒。
- x3 固視点の中心から刺激の中心までの距離は 0.7。
- x4 刺激の直径は 0.1。
- x5 刺激の位置 (2 か所) と色 (2 種類) の組み合わせで合計 4 種類の刺激に対して 25 試行ずつ、合計 100 試行。

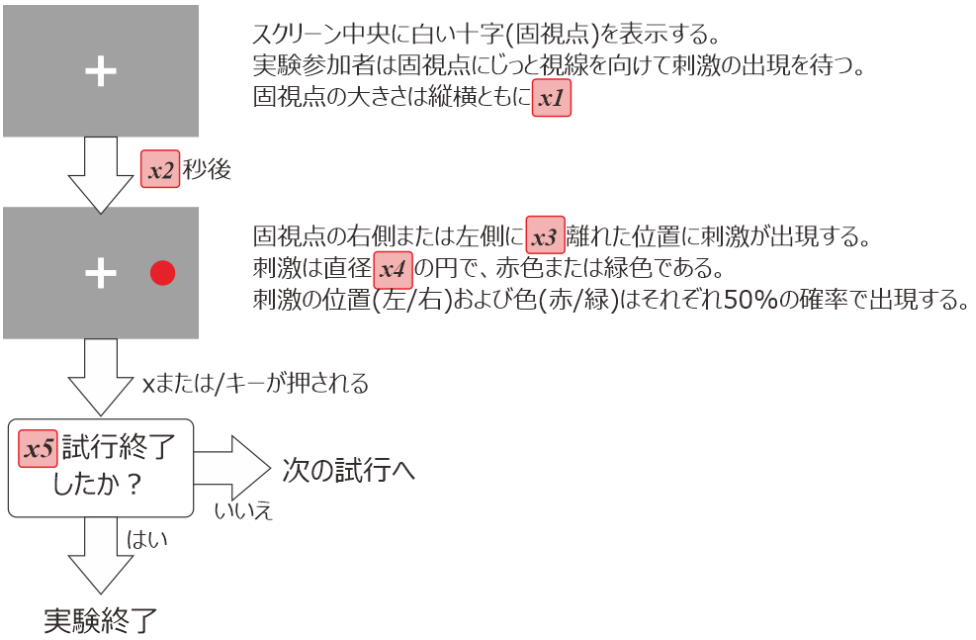


図 3.2　サイモン効果実験の手続き。実際に実験を作成するためには x1 から x5 の値を決定する必要があります。

刺激はスクリーンの左右端付近に出したいのですが、使用するスクリーンの縦横比によって左右端の X 座標が異なります。ここでは縦横比 3:2 のスクリーンでも刺激がおさまることを考えて固視点の中心から刺激の中心までの距離 (x3) を 0.7、刺激の直径 (x4) を 0.1 としました。 **Pilot モード時のスクリーン解像度を設定せず初期値 (800 × 600) のまま使用していると、この設定ではウィンドウの端にごくわずかだけしか刺激が見えません。** 実際に使用しているモニターの縦横比が 4:3 ならともかく、一般的なもっと横長のモニターを使用している方は、モニターの縦横比に合うように Pilot モードのウィンドウサイズを変更することをお勧めします (「2.10:Pilot モードのウィンドウサイズを設定しよう 」参照)。

実験を作成する前にもうひとつ確認して置きたいのは、この実験において「試行毎に変化する値」は何かという点です。 図 3.1 から明らかなように、刺激の色 (赤 or 緑) と刺激の位置 (右 or 左) は試行毎に変化します。刺激の色が変化するという事は、それに対応して正解となるキー (x or /) も変化することに注意しましょう。

さて、それではいよいよ実験の作成を始めましょう。なお、今回作成する実験の psyexp ファイル名は、第 3

章の実験ということで exp03.psyexp とすることにします。

3.2 実験のためのフォルダを準備して psyexp ファイルを保存しよう

Builder の初期設定では、起動時に前回に作成した実験が自動的に開きます。本書を解説順序に従って作業しているなら、前章で作成した実験が開かれていると思います。その場合は、Builder のメニューの「ファイル」から「新規」を選んで新たな実験を作成しましょう。新たな Builder のウィンドウが開いて、まっさらな新しい実験が作成されます。新しい実験を作成したら、前章の実験が開いた Builder のウィンドウは閉じてしまって構いません。もし新しい実験を作成する前に Builder のウィンドウを閉じてしまった場合は、Runner や Coder ののメニューの「ウィンドウ」から「Builder を表示」を選択すれば新たな実験で Builder ウィンドウを開くことができます。

新しい実験を作成したら、まず psyexp ファイルを保存しておきたいのですが、その前にひとつしておきたい作業があります。実験設定ダイアログを開いて、「入力」タブの **[キーボードバックエンド]** を PsychToolbox にしておいてください。初期状態では ioHub になっているはずですが、本書の範囲では ioHub の長所を活かせる実験は扱いません。ioHub は実行環境によって実験終了時に異常に時間がかかるなどの問題が生じるので、PsychToolbox にしておいた方が無難です (2025 年 2 月現在、PsychoPy の開発状況を確認すると、初期状態を PsychToolbox にする変更が検討されているようなので、将来的にはこの作業は不要になるかもしれません)。

[キーボードバックエンド] の設定を終えたら、次は本章の実験のためのフォルダを作成してください。デスクトップ、ドキュメントファルダ、USB メモリなど、ユーザーが自由にファイルを作成できる場所なら好みに応じてどこでも結構です。Builder では **ひとつの実験に関するファイルはすべてひとつのフォルダにまとめる** とファイル管理がとても楽になります。授業などで「PsychoPy 実習」などといったフォルダを作成して、授業関連のファイルをすべてまとめておこう考えている方は、その「PsychoPy 実習」フォルダの中に exp03 などこの章の実験のためのフォルダ (例えば exp03 といった名前) を作成してください。次章以降でも、新しい実験を作成する度にその実験専用のフォルダを作るようにしてください。

この章の実験のためのフォルダを作成して、そこへ新しい実験を exp03.psyexp として保存し終えたら、次の作業へ進みましょう。

チェックリスト

- 新たに実験を作成するときは、その実験のためのフォルダを用意してその中に psyexp ファイルを保存する習慣をつける。
- **[キーボードバックエンド]** は ioHub の機能が必要な実験でなければ PsychToolbox にする。

3.3 視覚刺激を配置しよう

第 2 章 では視覚刺激の大きさや位置、色の指定などについて学んだのですから、まずは刺激の作成から始めましょう。まず、刺激の円は Polygon コンポーネントで **[形状]** を円すれば描画できます。刺激の位置と色は試行毎に変化しますが、まだその方法は解説していませんのでひとまず (0.7,0) の位置に赤色で表示することにしましょう。Polygon コンポーネントをひとつルーチンに配置し、以下のように設定してください。

- **「基本」タブ**
  - **[名前]** に `stimulus` と入力する。

  - **[形状]** を円にする。

•**「外観」タブ**

  - **[塗りつぶしの色]** に `red`、**[枠線の色]** に `None` と入力する。

•**「レイアウト」タブ**

  - **[サイズ [w, h] $]** に `(0.1, 0.1)` と入力する。
  - **[位置 [x, y] $]** に `(0.7, 0)` と入力する。

- PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログを開いて **[単位]** を height にしておく。

続いて固視点ですが、幸いなことに Polygon コンポーネントは十字を描くこともできます。新たに Polygon コンポーネントを 1 つルーチンに配置し、以下のように設定してください。

•**「基本」タブ**

  - **[名前]** に `cross` と入力する。
  - **[形状]** を十字にする。

•**「レイアウト」タブ**

  - **[サイズ [w, h] $]** に `(0.05, 0.05)` と入力する。

続いて提示時間の設定について考えましょう。試行開始時には固視点のみがスクリーン上に提示されていて、試行開始 1.0 秒後に刺激が提示されるのですから、固視点 (cross) の **[開始]** は「時刻 (秒)」で 0.0、刺激 (stimulus) の **[開始]** は「時刻 (秒)」で 1.0 にすればよいでしょう。固視点、刺激ともにキーが押されるまでずっと提示されるのですから、**[終了]** はどちらも空白にしておきましょう。

さて、ここまでの作業を終えると、ルーチンペインには 図 3.3 上のように 2 つの Polygon コンポーネントが並んでいるはずです。コンポーネントの順番はこの通りでなくても構いません。そして実験を実行すると 図 3.3 下のようにスクリーンに刺激が提示されることを確認してください。ここまでの記述で分からないことがあったら 第 2 章 を復習してください。ここまで確認できればいよいよキーボードを用いて実験参加者の反応を検出する方法を学びましょう。

チェックリスト

- Polygon コンポーネントを用いて円をスクリーン上に提示することができる。
- Polygon コンポーネントを用いて十字をスクリーン上に提示することができる。

3.4 キーボードで反応を検出しよう

Builder からキーボードを利用するには、コンポーネントペインの Response というカテゴリに含まれる Keyboard コンポーネントをルーチンペインに配置します。Polygon や Text コンポーネントを配置する時と同じように、コンポーネントペインの Keyboard コンポーネントのアイコン (図 3.4) をクリックしましょう。すると Keyboard コンポーネントのプロパティ設定ダイアログが開きます。

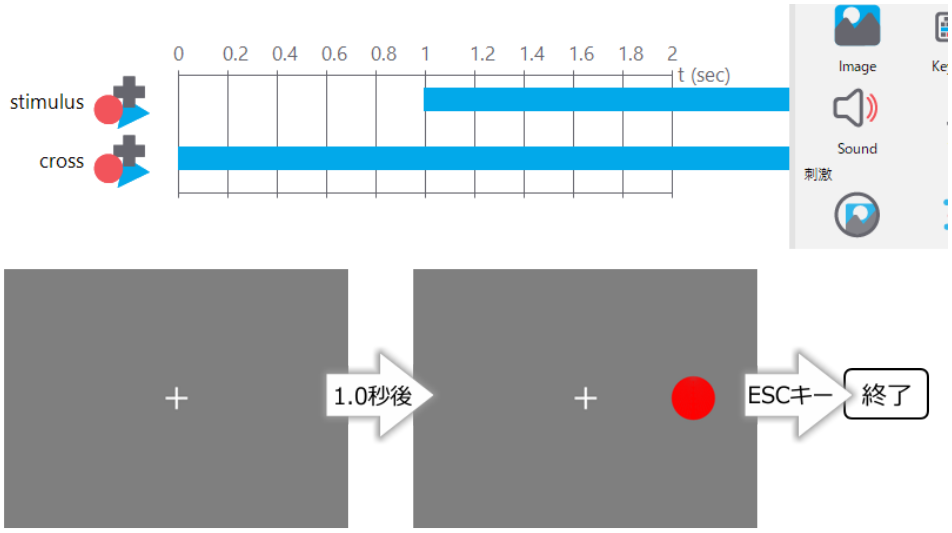


図 3.3　視覚刺激用の Polygon コンポーネントの配置と実行確認。

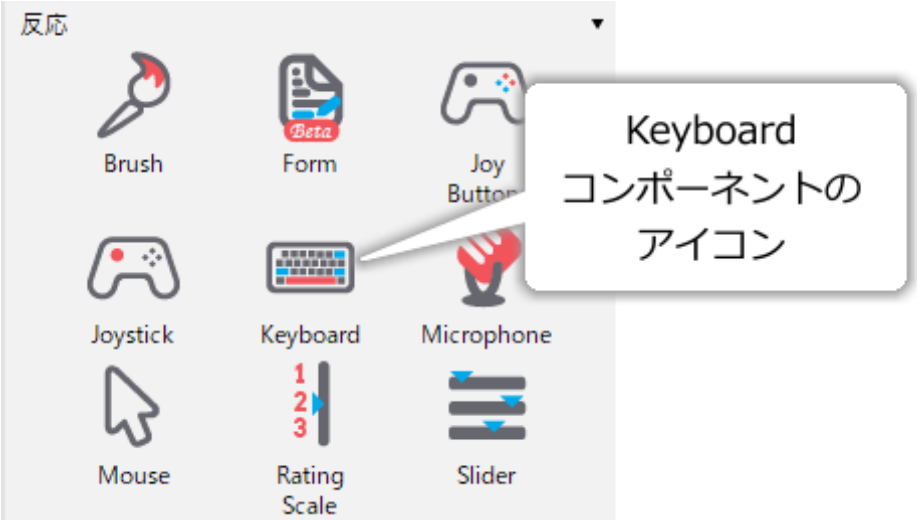


図 3.4　Keyboard コンポーネントのアイコン

「基本」タブの **[名前]**、**[開始]** や **[終了]** は、第 2 章 で扱ったコンポーネントと同様に、名前をつけたりコンポーネントが有効となる時間を指定したりするために使用します。視覚刺激の場合、「有効となる時間」はスクリーン上に表示されている時間に対応しましたが、Keyboard コンポーネントの場合はキー押し反応を受け付ける時間に対応します。例えば今回の実験の場合、刺激がスクリーン上に出現する前に押されたキーは無視するべきですから、試行開始 1.0 秒経過するまで Keyboard コンポーネントは無効にするべきです。また、今回は参加者が反応するまで待ち続けますが、実験によっては 3 秒以内に反応できなかったら強制的に次の試行に移るといったこともあるでしょう。 **[開始]** と **[終了]** を適切に設定することによってこういった実験を実現することができます。

[終了] の下に **[Routine を終了]** という項目がありますが、ここにチェックが入っていれば、キーが押されると他のコンポーネントがまだ終了していなくても強制的にルーチンを終了します。今回はこの機能を利用して「参加者がキーを押すまで刺激を提示し続ける」という動作を実現します。仮に「30 秒間スクリーンを注視し、刺激が無作為な時間間隔で複数回スクリーン上に出現する度にできるだけ速くキーを押す」といった実験の場合はキーが押されてもルーチンを中断してはいけません。このような実験の場合は **[Routine を終了]**

のチェックを外しておきます。

[検出するキー $] には、Builder がキー押しを監視するキーの名前を列挙します。初期状態では'y', 'n', 'left', 'right', 'space' と入力されています。これは順番にアルファベットの Y キー、N キー、カーソルキーの「←」キー、「→」キー、スペースキーを表すキー名です。必ず半角文字で入力すること、キー名の前後にシングルクォーテーションまたはダブルクォーテーション記号でキー名を囲むこと、キー名の間はカンマで区切ることが重要なポイントです。ここに書かれたキー以外が押された場合は、Keyboard コンポーネントが有効な時間帯であっても無視されます。参加者が反応に使用するキーのみを書くことによって、参加者が誤って無効なキーを押してしまってルーチンが終了してしまうことを防止できます。心理学実験ではあまり無いことだと思いますが、すべてのキーを有効する必要がある場合は **[検出するキー $]** を空白にしてください。

表 3.1 に PsychoPy が認識するキー名の一覧を示します。一般的なキーボードでは Shift キーを押しながら 1 キーを押すと！記号を入力できますが、PsychoPy は「！というキーが押された」とは判断せず、「Shift キーとと 1 キーが押された」と判断しますので注意してください。

表 3.1: 一般的な日本語キーボードで使用できるキー名

キー名	対応するキー	キー名	対応するキー
f1	F1	**minus**	-
f2	F2	**asciicircum**	^
f3	F3	**backslash**	\
f4	F4	**bracketleft**	[
f5	F5	**bracketright**	]
f6	F6	**semicolon**	;
f7	F7	**colon**	:
f8	F8	**comma**	,
f9	F9	**period**	.
f10	F10	**slash**	/
f11	F11	**at**	@
f12	F12	**return**	Enter キー
num_0	0 (テンキー)	**1**	1
num_1	1 (テンキー)	**2**	2
num_2	2 (テンキー)	**3**	3
num_3	3 (テンキー)	**4**	4
num_4	4 (テンキー)	**5**	5
num_5	5 (テンキー)	**6**	6
num_6	6 (テンキー)	**7**	7
num_7	7 (テンキー)	**8**	8
num_8	8 (テンキー)	**9**	9
num_9	9 (テンキー)	**0**	0
num_add	+ (テンキー)	**a**	A
num_subtract	- (テンキー)	**b**	B
num_multiply	* (テンキー)	**c**	C
num_divide	/ (テンキー)	**d**	D

次のページに続く

表 3.1 – 前のページからの続き

キー名	対応するキー	キー名	対応するキー
num_decimal	. (テンキー)	**e**	E
left	←	**f**	F
down	↓	**g**	G
right	→	**h**	H
up	↑	**i**	I
escape	ESC キー	**j**	J
pageup	PageUp キー	**k**	K
pagedown	PageDown キー	**l**	L
end	End キー	**m**	M
home	Home キー	**n**	N
delete	Del キー	**o**	O
insert	Ins キー	**p**	P
backspace	バックスペースキー	**q**	Q
tab	タブキー	**r**	R
lshift	左シフトキー	**s**	S
rshift	右シフトキー	**t**	T
lctrl	左 Ctrl キー	**u**	U
rctrl	右 Ctrl キー	**v**	V
lalt	左 Alt キー	**w**	W
ralt	右 Alt キー	**x**	X
		y	Y
		z	Z

もうひとつ注意しないといけない点は、キーボードによっては特殊キーの左右を区別しないなどのクセがある点です。例えば 表 3.1 では左シフトキーと右シフトキーにそれぞれ lshift、rshift という名前が割り当てられていますが、使用しているキーボードが左右どちらのキーを押しても lshift というキー名を返すように作られている場合があります。自分が使用する予定のキーボードがどのようなキー名を返すのか確認したい、Windows 用キーボードの Windows キーや Mac 用キーボードの Command キーがどのような名前を返すのか確認したいといった場合は、キー名を表示させる Builder の「実験」を作成すると便利です。作成にはずっと先の章で取り上げるテクニックが必要になりますので、本文から分離して「 3.12.1: 自分のキーボードで使えるキー名を確かめる 」に記しておきました。参考にしてください。

説明も長くなってきましたので、その他の項目は後で解説することにして、まずは動作を確認してみましょう。Keyboard コンポーネントのプロパティ設定ダイアログで以下のように設定してください。

- **「基本」タブ**
  - **[開始]** を「時刻 (秒)」にして `1.0` と入力する。
  - **[終了]** を空白にする。
  - **[Routine を終了]** にチェックが入っていることを確認する。
  - **[検出するキー $]** に `'x', 'slash'` と入力する。

- その他の項目は初期設定のままにする。

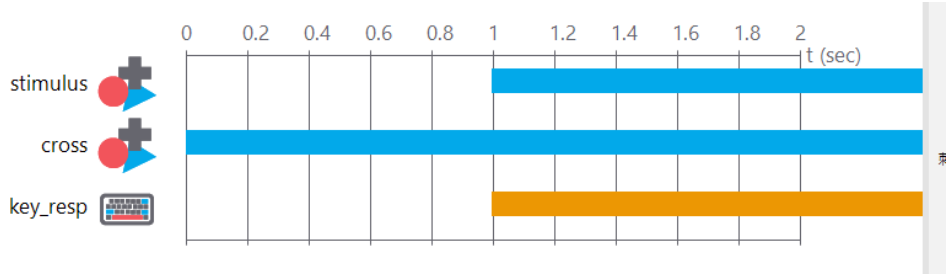


図 3.5　Keyboard コンポーネント配置後のルーチン。反応に関するコンポーネントの有効な時間はオレンジ色のバーで表示されます。

これらの設定が済めばルーチンペインは 図 3.5 のようになるはずです。Keyboard コンポーネントの有効な時間を示すバーが青ではなくオレンジである点に注意してください。反応計測に用いるコンポーネントはこのようにオレンジ色で表示されます。これで、x または / キーが押されたことを検出してルーチンを終了させることができるようになりました。以下のことを必ず確認しておいてください。

- 刺激 (赤い円) が出現した後に x キーまたは / キーを押すと刺激提示が終了すること。
- 刺激 (赤い円) が出現する前に x キーまたは / キーを押しても刺激提示が終了しないこと。
- x キー、 / キー、ESC キー以外のキーを押しても効果がないこと（ただし Windows キーなど OS により特殊な役割を持つキーは除く）。

キー押しが検出できたら次はそれをファイルに記録する方法をマスターしたいところですが、それはひとまず置いておいて、次節では刺激の位置や色を変更しながら試行を繰り返す方法を学びます。

チェックリスト

- ルーチン実行中にキー押しが有効となる時間帯を指定できる。
- キーが押されると直ちにルーチンを中断するように設定することができる。
- 有効とするキーを指定できる。
- すべてのキーを有効にするように設定することができる。

3.5 条件ファイルを作成しよう

今までは「実験」と言いつつただ静止画像が一回スクリーンに表示されて終了するだけでしたが、本節からいよいよ実験らしくなってきます。Builder では、条件ファイルと呼ばれるファイルから試行毎の刺激の色や反応に使用できるキーなどの値を読み込んで設定する機能があります。この機能を用いることによって、今までは不可能だった「刺激の色や位置を変えながら試行を繰り返す」ことが可能になります。条件ファイルの作成方法は何通りかあるのですが、ここでは Excel を用いた方法を解説します。第 1 章で述べたとおり、Excel をお持ちでない方は LibreOffice Calc を代わりに用いることができます。

では、条件ファイルの作成を始めましょう。Builder を一旦離れて Excel を起動してください。Builder で利用できる条件ファイルを Excel で作成するには、Excel のシートの各列に刺激の色や位置等のパラメータを割り当てて、実験で使用するパラメータの組み合わせを 1 条件 1 行で列挙します。 図 3.6 を参考にしながら具体的に手順を追っていきましょう。

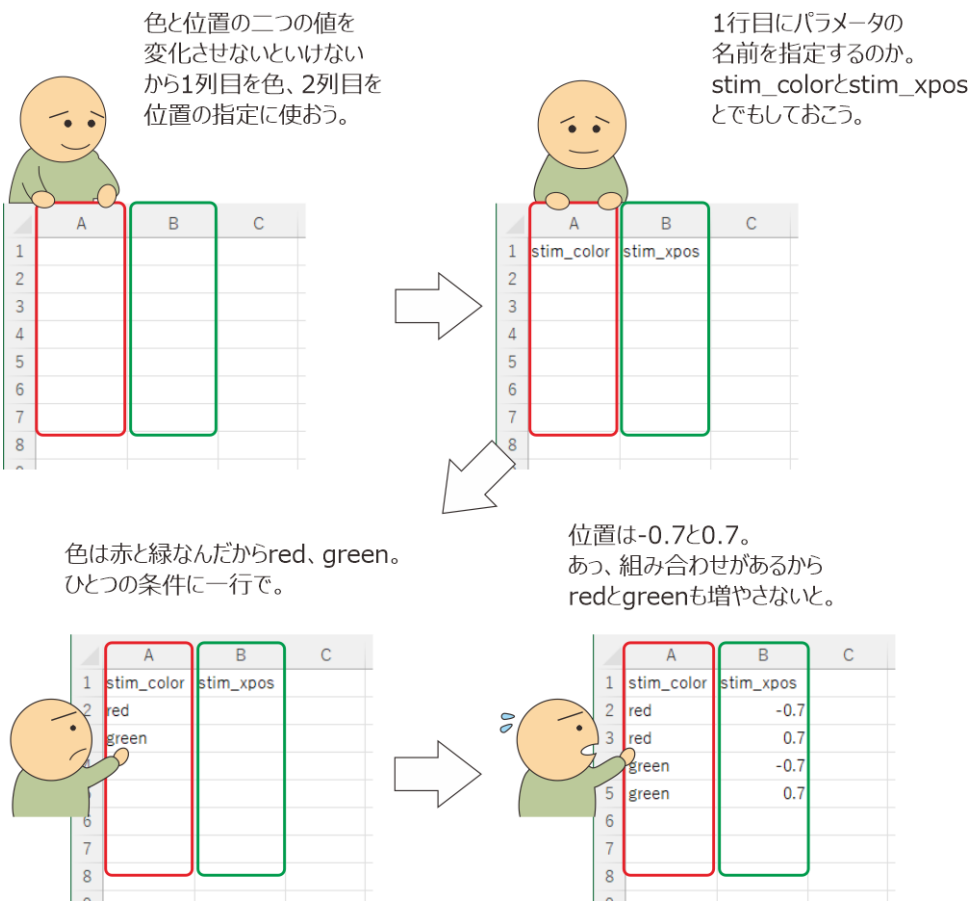


図 3.6　条件ファイルの作成。

今回の実験では試行毎に刺激の「色」と「位置」が変化します。このように変化していくものそれぞれに Excel の列を割り当てていきます。例えば「色」を 1 列目、「位置」を 2 列目に割り当てるとしましょう。このように、条件ファイルを通じて変化させていく値をパラメータと呼びます。この用語は以後頻出しますので覚えておいてください。続いて決めないといけないのは、それぞれのパラメータの「名前」です。人にものを頼む時には「コレをアレに載せといてよ」と言っても通じるかも知れませんが、コンピュータに作業を頼むときには「コレ」や「アレ」を曖昧さなしに示す必要があります。そこで用いられるのが名前です。「刺激の色」と言った時に「刺激」とは具体的にどのコンポーネントを指しているのか、「値を設定する」と言った時に「値」とはどこに記されているものか、といったことをコンピュータにもわかるように示すために、私たちがコンピュータに指示を出す時には、操作や参照の対象となるものすべてに固有の名前をつけなければいけません。「固有の名前」とは、言い換えると異なるものに同じ名前を割り当ててはいけないということです。第 2 章 で Polygon コンポーネントを複数配置した時に、一つ目の Polygon コンポーネントの **[名前]** の初期値が「polygon」、二つ目の Polygon コンポーネントの **[名前]** の初期値が「polygon_2」になっていたのに気付かれた方がおられると思いますが、これは Builder が二つの Polygon コンポーネントを明瞭に **[名前]** で区別できるように自動的に異なる名前を割り振ってくれた結果なのです。

Builder で使用できる名前は、「Python の変数として使用できる名前」のうち、PsyhcoPy や Builder でまだ使用されていないものに限られます。現在の PsychoPy のベースとなっている Python3 では日本語の文字や一部の記号も名前に使用できるのですが、Python の文法規則の詳細を知らなければ理解しがたい制限も多いので、以下のルールを満たすものを使用するのが無難でしょう。

- 1 文字目が英文字 (A-Z、a-z) であること。アンダーバー (_) も使用できるがアンダーバーから始まる名

前は避けた方がよい (Python の文法上特別な意味を持つので)。英文字の大文字と小文字は区別される。

- 2 文字目以降が英文字 (A-Z、a-z)、数字 (0-9) またはアンダーバー (_) であること。英文字の大文字と小文字は区別される。
- Python の正常な動作のために確保されているキーワード (if, else, while, for, from, global など) や PsychoPy のモジュール名 (core, data, visual, event など)、Builder の予約語 (expInfo, expName, buttons など) は使用できない。以後、記述の簡略化のためにこれらを合わせて「予約語」と呼ぶ。

一番目と二番目の条件を満たすのはそれほど難しくありませんが、最後の「予約語を使用してはいけない」という条件は厄介です。予約語は Python や Builder が正常に動作するようにすでに使用している名前のことであり、かなりの数の名前が予約済みなので覚えるのはあまり現実的ではありません (「 13.4: 予約語 」参照)。予約語を避けるのに有効なのが、英小文字から始まる接頭語 + アンダーバーという名前を用いるという方法です。例えば自分で決める名前すべて先頭に my_ を付けたり、刺激には s_、反応には r_を付けたりするといったルールを決めておけば、ほぼ確実に予約語を避けることができます。Builder を起動中であれば「 3.12.2:Builder で使用できない名前を判別する 」で述べる方法を用いて Builder で使用できる名前を判別することができます。

本題に戻りましょう。今回は刺激の色と位置を変化させる必要がありますので、二つの名前を自分で決める必要があります。上記の名前に使える文字列の制限に加えて、作成中のルーチンペインにすでに配置済みのコンポーネントの **[名前]** と重複しないように名前を決める必要があります。ここでは色に対応する名前として stim_color、位置に対応する名前として stim_xpos という名前を使うことにしましょう。Excel の 1 行目にこれらの名前を入力してください。

名前を決めたら、次はそれぞれのパラメータの値を列挙していきます。値は Builder が理解できるものでなければいけません。例えば「赤」とか「緑」とか日本語で入力しても Builder は理解できませんので、第 2 章 で覚えたように red や green とか#FF0000、#00AA00 などと書かねばなりません。刺激の位置も「右」、「左」ではなく刺激の **[位置 [x, y] $]** にそのまま記入できる値を書く必要があります。

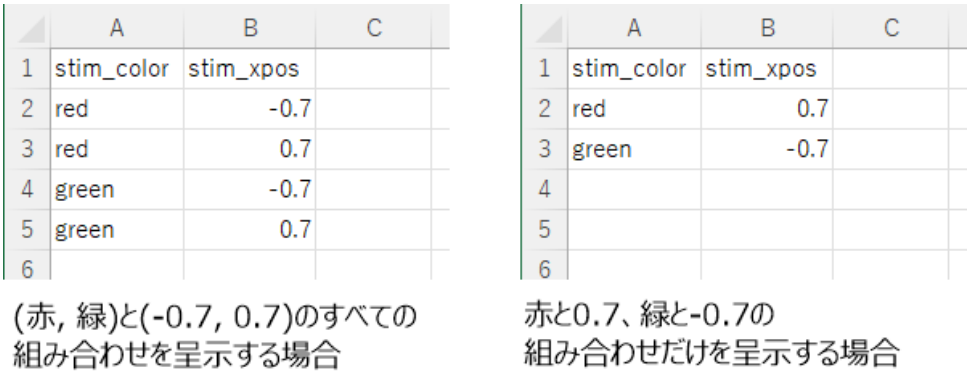


図 3.7　パラメータの組み合わせを全て提示するか、特定の組み合わせだけを提示するかによって条件ファイルの書き方が異なります。

これらの注意点を念頭に置いて、まず刺激の色を入力しましょう。刺激の色は赤または緑ですから、stim_color には red と green を挙げる必要があります。これらの値を、stim_color と入力したセルの下へ、空白セルを間に挟まずに並べていきます。続いて刺激の位置ですが、今回の実験では位置の Y 座標は変化しないので X 座標の値だけを列挙することにしましょう。固視点から刺激への距離を 0.7 に決めているのですから、左は-0.7、右は 0.7 です。stim_color と同じように stim_xpos の列に-0.7、0.7 を縦に並べればよいのですが、今回の実験

では「赤」の刺激が「右」と「左」、「緑」の刺激が「右」と「左」で合計 4 通りの条件があることに注意しなければいけません。条件ファイルでは 1 行が一つの実験条件に対応しますので、これら 4 通りの条件がすべて列挙されなければいけません。どうすればよいかというと、stim_color の列を red, red, green, green といった具合に red と green が 2 回ずつ出現するように書き換えて、stim_xpos に-0.7、0.7、-0.7、0.7 と書けばよいのです (図 3.7 左)。これですべての条件が挙げられたことになります。この並べ方は Builder で実験を作成する際には非常に頻繁に用いますのでよく覚えておいてください。なお、「赤色の時は必ず右に、緑色の時は必ず左に刺激が出現する」という実験の場合には 2 通りしか組み合わせがありませんので、 図 3.7 右のように red と 0.7、green と-0.7 がそれぞれ同一の行に並ぶように書きます。

今回の実験のパラメータには、刺激の位置と色の他にも「正解となる反応」がありますが、まずは実際に刺激が次々と提示される様子を確認することにしましょう。他のセルに不必要な値が入力されていないのを確認してから、作成したシートを Excel の「ブック (*.xlsx)」形式で保存してください。以後、この形式で保存されたファイルを xlsx ファイルと呼びます。LibreOffice Calc を使っている場合は標準では xlsx ファイルとして保存されませんので保存ダイアログの「ファイルの種類」から xlsx 形式を選択することを忘れないようにしてください (図 3.8)。**保存場所は必ず psyexp ファイル (ここまで解説通りに作業しているなら exp03.psyexp という名前のはずです) と同じフォルダにしてください。**保存する xlsx ファイル名は、第 3 章の実験用の条件ファイルという事で exp03_cnd.xlsx としておきます。

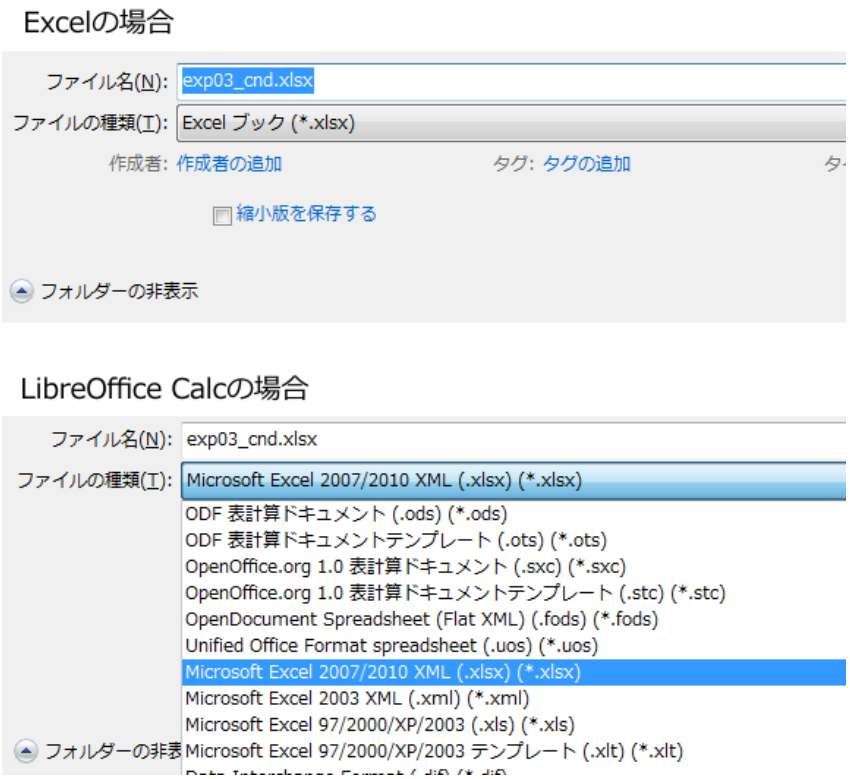


図 3.8　条件ファイルを Excel のブック (*.xlsx) 形式で保存します。Excel の場合は標準で xlsx ファイルとして保存されますが、LibreOffice Calc の場合は [ファイルの種類] から xlsx 形式を選択する必要があります。

もし自分が psyexp ファイルを保存している場所がわからない場合は、Builder に戻って「ファイル」メニューの「ファイルエクスプローラで開く」を選択してください。 OS 標準のファイルエクスプローラ (Windows ならエクスプローラー、MacOS なら Finder など) で現在編集中の psyexp ファイルがあるフォルダが開きます (図 3.9) 。フォルダの位置を確認して Excel から保存するときに該当フォルダを開くのもいいですし、いっ

たんデスクトップなどの見つけやすい場所に xlsx ファイルを保存しておいて、後から psyexp ファイルと同じフォルダへ移動させてもいいでしょう。後から移動する場合は、xlsx ファイルを Excel で開いたままだと「ファイルが使用中なので移動できない」といったエラーが出てしまうので、いったん閉じてから移動してください。コピーなら Excel を閉じなくてもできますが、複数の場所に xlsx ファイルがあるとどちらを編集しているのかわからなくなりがちなのでお勧めしません。

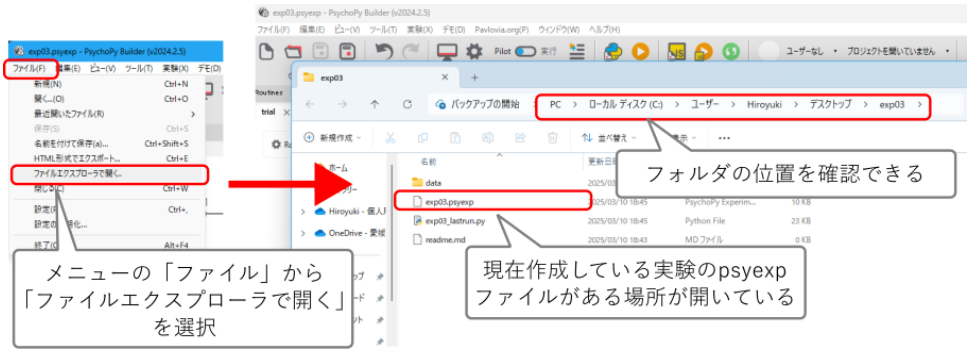


図 3.9　「ファイル」メニューの「ファイルエクスプローラで開く」を使うと現在編集中の psyexp ファイルがあるフォルダを簡単に開くことができます。

これで条件ファイルの準備ができました。Builder に戻ってこの条件ファイルを読み込むように実験を拡張しましょう。

チェックリスト

- Excel または LibreOffice Calc を用いて、実験に用いるパラメータを列挙した条件ファイルを作成することができる。
- 実験のパラメータを表現するために PsychoPy で使用できる名前を決めることができる。
- psyexp ファイルを保存しているフォルダを Builder から開くことができる。

3.6 繰り返しを設定しよう

Excel は一旦終了して、Builder に戻ってください。一度 Builder を終了してしまった人は exp03.psyexp を開きましょう。開いたら、ルーチンペインの下のフローペインを見てください。 図 3.10 の左上のように、左から右に矢印が描かれていて、矢印の上に trial と書かれた四角いアイコンが描かれています。フローペインは実験の流れ (フロー) を示しており、左から右へ向かって実験が進行します。trial と描かれた四角はルーチンを示しており、trial ルーチンは実験を作成すると自動的に作成されるルーチンです。ここまでの作業で「ルーチンにコンポーネントを配置する」という作業を何度も繰り返しましたが、これらの作業はいずれも trial ルーチンに対して行っていたのです。

さて、この trial ルーチンを繰り返し実行するためには、フローにループを挿入する必要があります。ループを挿入するには、フローペインの左端の「ループを挿入」を左クリックします。クリックした後にフロー上にマウスカーソルを動かすと、ループの始点または終点を挿入できる点にカーソルが重なった時にその場所に黒い円が表示されます (図 3.10 左下)。黒い円が表示されている時にその位置をクリックすると、始点または終点として指定することができます。通常は始点と終点の二カ所を指定する必要がありますが、今回はルーチンが一つしかなくて始点が決まると終点は自動的に決まってしまうため、一カ所をクリックするだけで始点と終点が確定します。

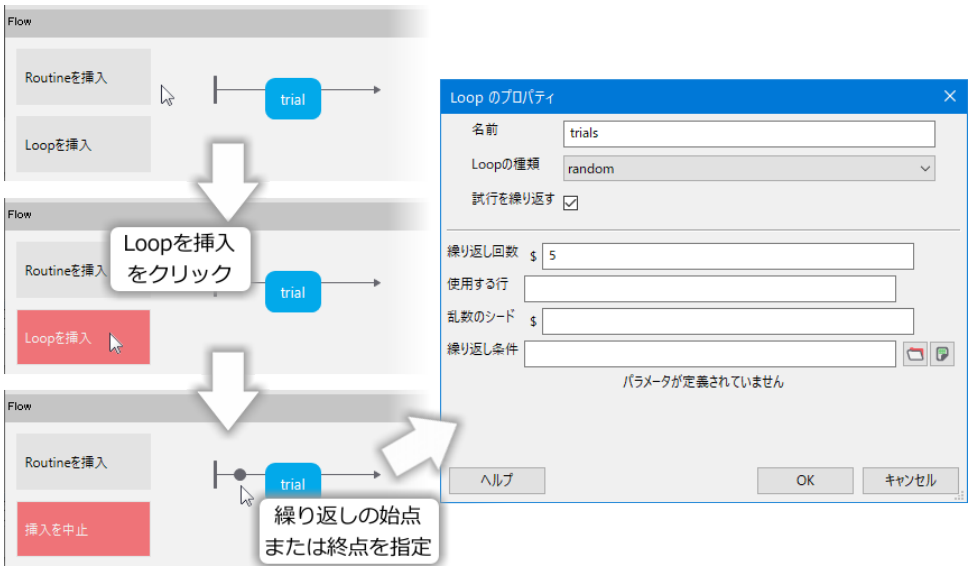


図 3.10　繰り返しの挿入。

ループの始点と終点が確定すると、ループの設定ダイアログが表示されます。ここではループの名前や種類、ループ回数、条件ファイルなどを指定します。名前はコンポーネントの名前と同様に、Builder 上での表示をわかりやすくするためなどに使えます。問題は **[Loop の種類]** ですが、本章では指定できる種類の内 sequential、random、fullRandom の三種類を解説します。残りの 2 つは階段法と呼ばれる特殊な手続きを実現するためのものなので、トピックとしておきました。「 3.12.10:staircase と interleaved staircases による階段法の手続き (上級) 」をご覧ください。

図 3.11 は、exp03_cnd.xlsx で指定した 4 条件の条件ファイルを sequential、random、fullRandom で繰り返した場合の実行例を示しています。ループ回数を指定する **[繰り返し回数 $]** には 3 が入力されていると仮定しています。sequential では、条件ファイルに書かれた条件が、そのままの順番で 2 行目 (1 行目はパラメータ名) から実行されます。最後の行まで進むとまた 2 行目に戻って繰り返し、 **[繰り返し回数 $]** で指定された回数の繰り返しが終了すると終了します。random と fullRandom では無作為に順番が並べ替えられますが、random では「条件ファイルに書かれた条件を無作為に並び替えて実行する」という作業を **[繰り返し回数 $]** 回繰り返します。従って、それぞれのループでは条件ファイルに書かれた条件が必ず 1 回ずつ実行されます。 図 3.11 の random の例で 1 回目、2 回目、3 回目の繰り返しの全てにおいて条件ファイルに書かれた 4 つの条件が 1 回ずつ実行されていることを確認してください。これに対して、fullRandom では全試行を終えた時点で条件ファイルに書かれた条件がそれぞれ **[繰り返し回数 $]** 回繰り返されます。 図 3.11 の random と fullRandom をよく見比べてその違いを理解してください。一般論として、random では同一条件の試行が続くことはあまりありません (n 回目の繰り返しの最後の試行と n+1 回目の繰り返しの最初の試行が一致した場合のみ) が、fullRandom では同一条件の試行が続くことがしばしばあります。これらのループタイプを使い分けることが Builder を使いこなすコツです。

[乱数のシード $] は、繰り返し順序の無作為にするための処理を操作するための項目です。 **[乱数のシード $]** を空白にしておくと、PsychoPy が実験を実行する度に異なる繰り返し順序を決定してくれます。しかし、例えば 498202 という整数を指定しておくと、この値を変更しない限り、一見無作為な並び替えに見えますが毎回必ず同じ順序の繰り返しが行われます。この整数のことを乱数のシードと呼びます。シードを実験者が自分で指定してその記録をとっておけば、以前行った実験と全く同一の繰り返し順序を再開できます。研究の目的によってはこのような再現性が必要になるかも知れませんので、各自の判断で選択してください。なお、乱数のシードについてもう少し詳しく知りたい方は「 3.12.3: 無作為化と疑似乱数 」をご覧ください。

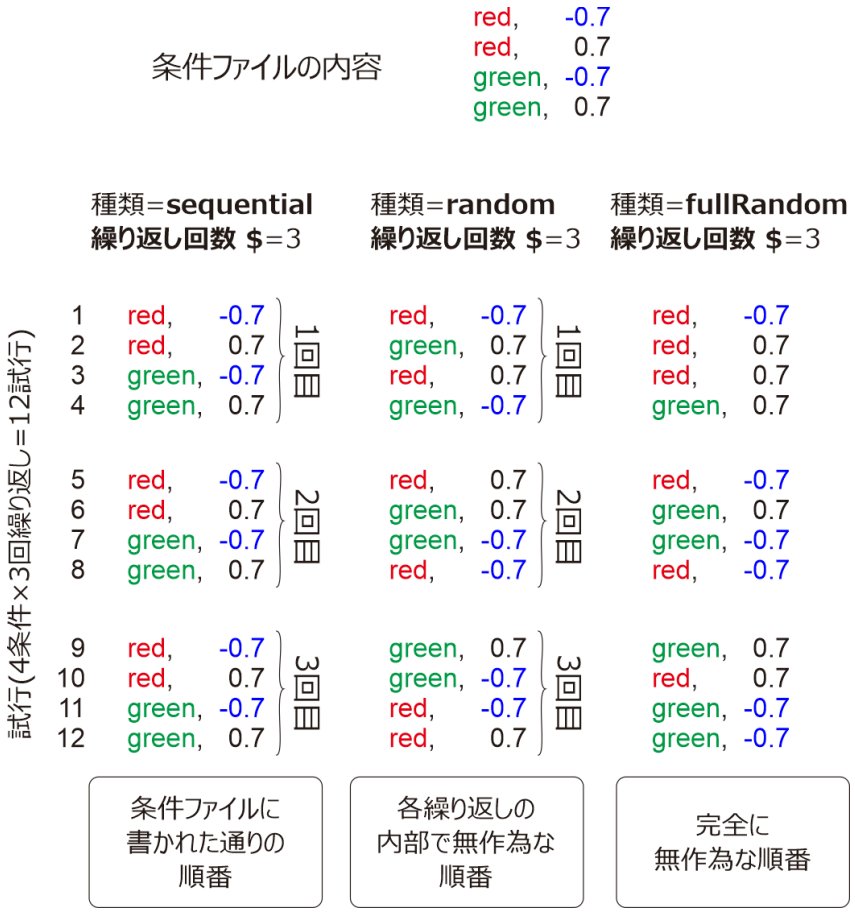


図 3.11　3 種類のループタイプの違い。

[条件] には、条件ファイル名を入力します。 図 3.12 に示すように入力欄の右側にある 2 つのボタンがあります。左側のボタンをクリックするとファイル選択ダイアログが開くので、先ほど作成した条件ファイルを選択しましょう。特に難しい操作ではないと思いますが、ひとつだけ注意してほしい事があります。 **[繰り返し条件] を入力する前に少なくとも一度、実験 (psyexp ファイル) を保存してください** 。Builder は psyexp ファイルが保存された場所を基準に条件ファイルを探しますので、一度も psyexp ファイルが保存されていない状態で条件ファイルを指定してしまうと、入力結果がおかしくなってしまう場合があります。この章を読みながら作業している方はすでに exp03.psyexp という名前で実験を一度保存しているはずですので問題は生じませんが、今後自分で実験を作成する時には気を付けてください。

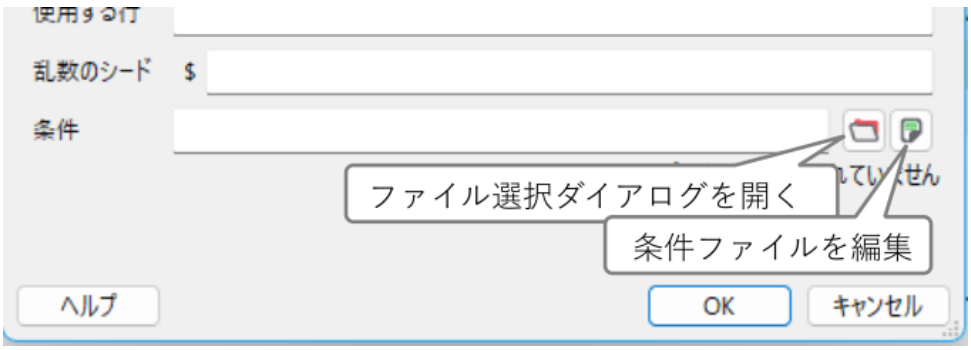


図 3.12　**[条件]** の入力欄の右側のボタン。ファイルを設定済みの状態で右端のボタンをクリックすると Excel など xlsx 編集アプリケーションが起動して条件ファイルの内容を確認、修正できます。

[条件] に条件ファイルを設定済みの状態で 図 3.12 の右側のボタンをクリックすると、Excel などのアプリケーションで条件ファイルを開いてくれます。条件ファイルの詳細を確認したい時や、内容を修正したい場合に便利なので覚えておいてください。 **[条件]** が空欄の時にこのボタンをクリックすると、内容が空っぽの xlsx ファイルが作成されます。内容を入力して保存しても自動的に **[条件]** に設定してくれないので、手作業で設定する必要があります。ご注意ください。

あと **[試行を繰り返す]** というチェックボックスと **[使用する行 $]** という項目がありますが、 **[試行を繰り返す]** については次章で解説します。 **[使用する行 $]** については「 3.12.4:Loop のプロパティ設定ダイアログの **[使用する行 $]** について 」をご覧ください。

さて、解説はこのくらいにして実験の作成を進めましょう。今回の実験では **[Loop の種類]** に fullRandom を用いることにして、 **[繰り返し回数 $]** には 25 を入力しましょう。 **[乱数のシード $]** は空白で良いでしょう。条件ファイルに 4 つの条件が定義されているので、それぞれを 25 回ずつ繰り返すと 25 × 4=100 試行となり、最初の計画と一致します。そして **[条件]** の項目の右の「ファイルを選択」ボタンを押して条件ファイル (exp03_cnd.xlsx) を選択してください。正常に読み込むことができれば **[条件]** に exp03_cnd.xlsx と条件ファイル名が表示されます。もし C:/Users/User/Documents/exp03_cnd.xlsx とか ../../Documents/exp03_cnd.xlsx のように **条件ファイル名の前に別の文字列がついている場合は、psyexp ファイルをまだ一度も保存していない、条件ファイルが psyexp ファイルとは別のフォルダになるなどの間違いを犯している可能性が高いので、よく確認してもう一度ファイルを選び直してください。**

正しく条件ファイルを選択できていることを確認したら、続いて 図 3.13 左のように **[条件]** の下に条件ファイルの概要が表示されているのを確認してください。図 3.13 では「2 パラメータ, 4 条件」と書かれていますが、「4 条件」が条件ファイルの行数 (先頭行を除く)、「2 パラメータ」が列数に対応しています。次の行に [stim_color, stim_xpos] と書かれているのは条件ファイル先頭行に入力したパラメータ名です。パラメータ名の順番は Builder が扱いやすいように自動的に並べ替えるので、パラメータ名に過不足がないことだけを確認してください。exp03_cnd.xlsx で定義したパラメータ名と一致していることが確認できたら、ダイアログの OK をクリックしてダイアログを閉じましょう。すると 図 3.13 右のようにフローペインにループ (左側へ戻る矢印) が表示されます。ループの矢印上に trials と書かれた白い四角が表示されていますが、これはループのプロパティ設定ダイアログの **[名前]** で設定したループの名前を示しています。

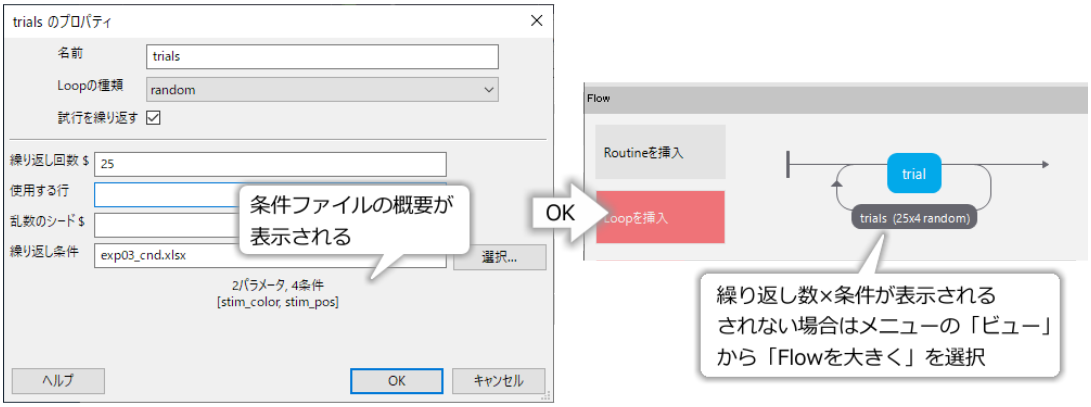


図 3.13　**[条件]** に条件ファイルを指定すると、条件ファイルの概要が表示されます。概要が表示されない場合は条件ファイルの読み込みに失敗していますので条件ファイルの内容を確認しましょう。

一度設定したループを再設定したい場合は、ループの名前が書かれた白い四角にマウスカーソルを重ねて左ボタンをダブルクリックしてください。削除したい場合は、同じくマウスカーソルを重ねた状態で右クリックをしてください。「削除」という項目だけがあるメニューが表示されるので、「削除」を選択しましょう (図 3.14

)。

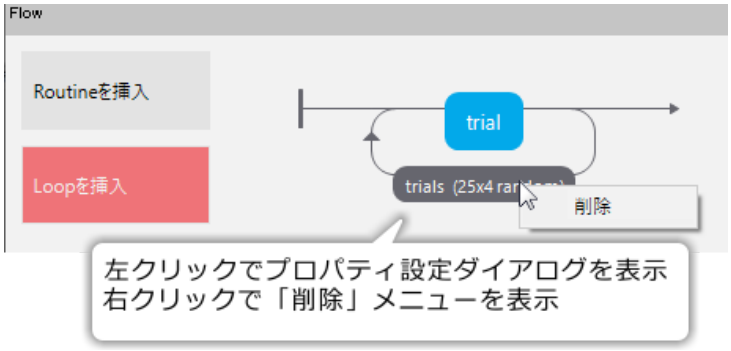


図 3.14　ループの再設定と削除

さて、これでいよいよ赤や緑の刺激が左右に表示されるようになりました、と言いたいところですが、まだもう一つ作業が残っています。この節の作業によって、Builder は条件ファイルから条件を読み取って、パラメータを変化させながらルーチンを繰り返し実行できるようになりました。しかし、肝心のルーチンが条件ファイルから読み取られたパラメータを利用できるようになっていません。次の節では、読み取ったパラメータを利用する方法を解説します。

チェックリスト

- フローペインでループの挿入を開始できる。
- ループを挿入する際、フローペインの矢印上に表示される黒い円の意味を説明できる。
- ループのプロパティ設定ダイアログで条件ファイルを指定し、表示された条件ファイルの概要が適切か確認できる。
- **[Loop の種類]** の sequential、random、fullRandom の違いを説明できる。
- **[繰り返し回数 $]** の値、条件ファイルに含まれる条件数、ルーチンが繰り返される回数の関係を説明できる。
- **[乱数のシード $]** に値を設定するとどのような効果が得られるか説明できる。
- 条件ファイルを **[条件]** に指定する前に psyexp ファイルを保存すべき理由を説明できる。

3.7 パラメータを利用して刺激を変化させよう

手始めに、条件ファイルから読み取ったパラメータを使って刺激の位置を左右に変化させてみましょう。ルーチンペインに戻って、stimulus のプロパティ設定ダイアログを開いてください。 **[位置 [x, y] $]** の欄には (0.7, 0) と入力されているはずですが、これを [stim_xpos, 0] に書き換えてください。そして、入力欄の右側にある「更新しない」と書かれたプルダウンメニューをクリックして「繰り返し毎に更新」を選択しましょう (図 3.15)。

このようにプロパティ設定ダイアログの項目に、条件ファイル先頭行のパラメータ名を書くと、Builder が実験を実行する時にパラメータの値を自動的に更新してくれます。右側のメニューを「繰り返し毎に更新」にし忘れるという失敗は、ある程度慣れてきたユーザーでも時々うっかりしてしまいがちなのですが、その場合は実験を実行して該当するルーチンまで進行すると突然実験が終了して Runner の画面に戻ってしまいます。こ

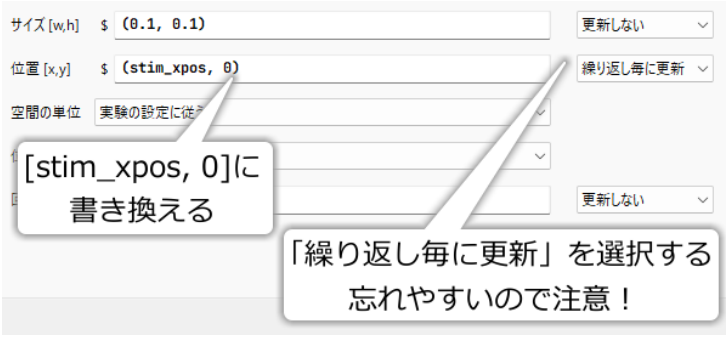


図 3.15　条件ファイルからパラメータを読み込んで自動的にプロパティを更新するように設定します。

の時、Runner の「標準出力」の欄を見ると、図 3.16 のように「stim_xpos なんて値は知らないよ」というメッセージが表示されているはずです。「繰り返し毎に更新」に設定していないと Builder が条件ファイルから読み込んだ値を stim_xpos に代入するという作業が行われないので、stim_xpos という名前の変数が存在していないというのがこのメッセージの意味なのですが、かなり分かりにくいですね。Builder がもう少しわかりやすいメッセージを出してくれたらと思うのですが、技術的にとても難しいことなので人間ががんばって解釈してあげる必要があります。

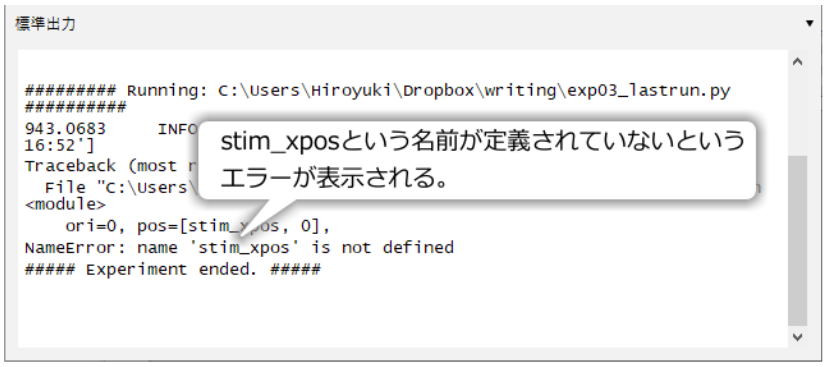


図 3.16　条件ファイルからパラメータを読み込むつもりなのに「更新しない」のまま実行してしまった時に表示されるエラー。

変更作業が終わったら、実験を実行して刺激の位置が左右に変化することを確認してください。これでようやく実験らしくなってきました。そして、エラーで実験が止まることを体験しておくために、ぜひ一度「更新しない」に戻して実行してみてください (体験した後はまた「繰り返し毎に更新」に戻すことを忘れずに！)。

続いて刺激の色も条件に応じて更新するようにしましょう。基本的には位置の更新と同じなのですが、ここにも一つ注意しないといけない点があります。今まで敢えて説明していなかったのですが、プロパティ設定ダイアログの項目には$が最後についているものとついていないものがあることに皆さんは気付いておられるでしょうか。例えば **[位置 [x, y] $]** や **[回転角度 $]** には$が付いていますし、 **[塗りつぶしの色]** や **[文字列]** には$が付いていません。この$記号は、難しい言い方をしますと「入力した値が文字列として評価されるか ($なし)、Python の式として評価されるか ($あり)」を表しています。

具体的な例を挙げましょう。Text コンポーネントの **[文字列]** に Target と入力されているとします。これは画面上に Target という文字列を表示したいということでしょうか。それとも、条件ファイルに Target という名前のパラメータが定義されていて、そのパラメータの値を代入したいということでしょうか。条件ファイルという便利なものを導入した代償に、「各コンポーネントのプロパティに文字列が入力されている場合、それはデータとしての文字列なのか、パラメータの名前なのか」を区別する方法を考えなければいけなくなってしま

いました。ここで登場するのが$です。Builder では、プロパティに入力された値に$が含まれていなければ、その値は文字列であるとみなされます (図 3.17)。従って、先ほど挙げた「**[文字列]** に Target と入力されている」例では Target という文字列をスクリーン上に提示したいのだと解釈されます。もし$Target と書かれていれば、Target という名前のパラメータの値を条件ファイルから読み込むのだと解釈されます。当然、条件ファイル内で Target という名前のパラメータが定義されていなければエラーになります。

図 3.17　$を付けることによって入力した値が名前を表しているのか文字列を表しているのかを区別します。

ここで、読者の皆さんの中には「ちょっと待てよ、じゃあ 図 3.15 の時はなんで$[stim_xpos, 0] じゃなくて [stim_xpos, 0] でパラメータの値を読み込めたの？」と疑問を持った方もいるかも知れません。確かに 図 3.17 の規則に従うのであれば、 図 3.15 は$[stim_xpos, 0] と書かないといけないはずです。ここで注目してほしいのが **[位置 [x, y] $]** というプロパティ名の最後の$記号です。この$は本来入力欄に書くべきだった$がプロパティ名に引っ越してきたと思えばよいでしょう。考えてみれば、 **[位置 [x, y] $]** や **[回転角度 $]** と言ったプロパティには文字列が指定される可能性はありません。例えば論文の手続きに「刺激を X 度回転した」と書いてあったら、常識で考えて X は変数であって文中のどこかで「X は 30 または 60」とかいう具合に数値が指定されているはずでしょう。それと同じことで、位置や回転角度のように数値を指定すべきプロパティに文字列が書いてあったら、常識的に考えてそれは名前であるはずです。だから、最初からプロパティ名に$を含ませてしまって、いちいち$を入力しなくてよいようになっているのです (図 3.18)。

以上の解説が済んだところで、ようやく条件ファイルを使って刺激の色を変化させる方法を説明することができるようになりました。刺激の色は **[塗りつぶしの色]** と **[枠線の色]** で指定します。これらのプロパティに条件ファイルで定義したパラメータ stim_color の値を代入したいのですが、 **[塗りつぶしの色]** にも **[枠線の色]** にも$はついていません。ではどのように書いたら良いか、もうお分かりですね。正解を 図 3.19 に示します。「繰り返し毎に更新」を選択するのを忘れないようにしてください。設定ができたら、実験を実行してみましょう。今度は刺激の位置も色もきちんと条件ファイルに従って変化するはずです。

これでようやく刺激が完成しました。次は実験記録を保存したファイルを確認しますが、その前に色の指定に関して少し補足しておきます。第 2 章 で色の指定方法について学んだ時に、色空間内の座標による指定では 1.0, 0.0, 0.75 のように RGB 成分をカンマ区切りで書く方法を紹介しました。**[塗りつぶしの色]** 、**[枠線の色]** 、**[背景色]** など、色を表すプロパティにはいずれも$がついていないので、本節の解説に従うと Builder は「1.0, 0.0, 0.75 という名前の色」と解釈してしまうはずです。実は、以前のバージョンの Builder ではまさにこのような解釈をしてしまって「1.0, 0.0, 0.75 などという変な名前の色は知らない」とエラーになってしまっていました。もし皆さんが先輩などが作成した実験を引き継ぐなどしたら、RGB 成分による色指定として$(1.0, 0.0, 0.75) とか$[1.0, 0.0, 0.75] といった具合に$を付けて ( ) か [ ] で囲んだ表記を見かけるかもしれませんが、以

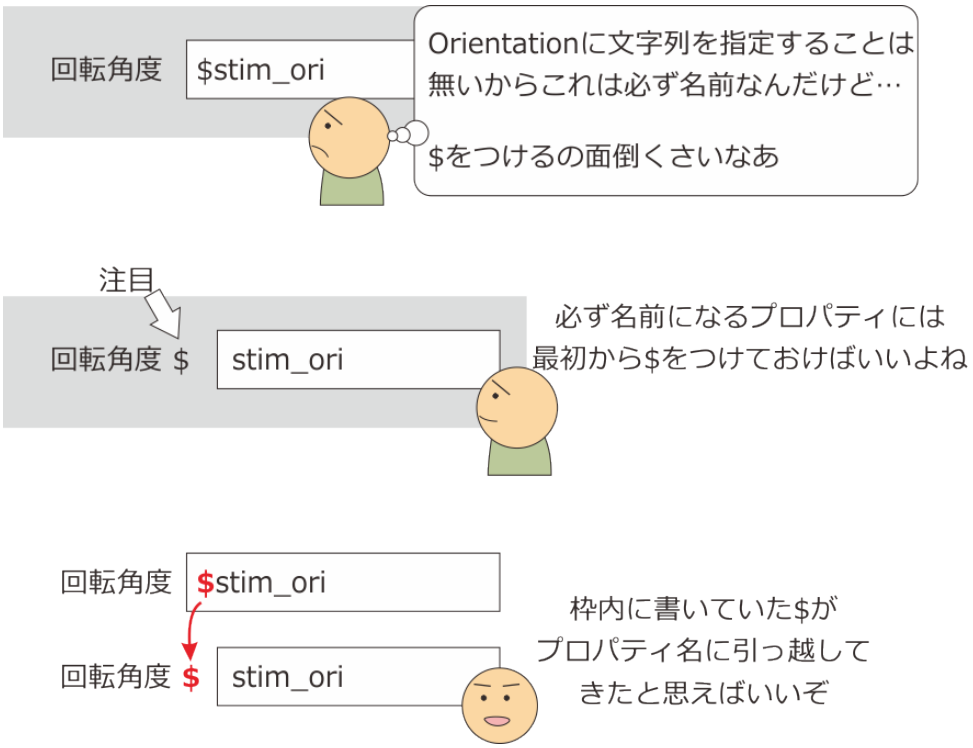


図 3.18　プロパティ名に$が付いているものは、文字列が指定されることがあり得ないプロパティなので、$を入力しなくても名前として解釈されます。

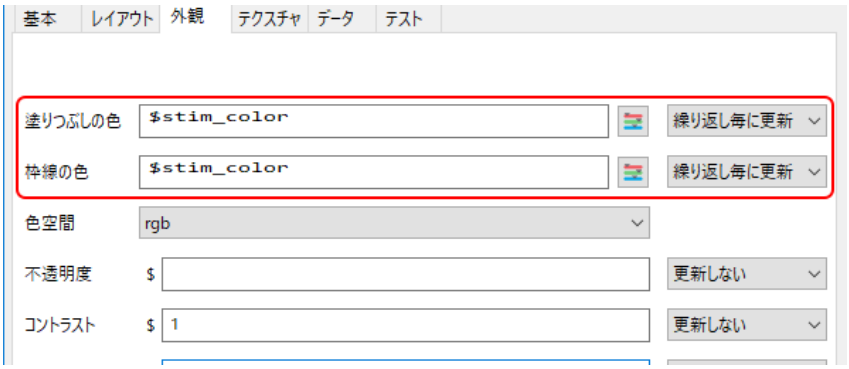


図 3.19　条件ファイルを用いて刺激の色を繰り返し毎に更新する時の設定例。「高度」タブですので注意してください。

前のバージョンで RGB 成分による指定をするためには、この書き方が必要だったのです。現在のバージョンの PsychoPy でもこの表記は有効なので、古い実験を現在の PsychoPy で実行する際に修正する必要はありませんが、「昔はこういう書き方が必要だったんだ」ということは知っておいていただくと混乱しなくて済むでしょう。現在の PscyhoPy は 1.0, 0.0, 0.75 のような指定があると「これは RGB 成分として解釈できないか？」とトライしてみて、解釈できなかった時だけエラーを返すようになっています。

あともうひとつ「$記号そのものを文字列として表示させたい場合はどうしたらいいの？」という問題があるのですが、かなり脱線しないといけませんので追加のトピックとしておきました。興味がある方は「 3.12.5:$を含む文字列を提示する 」をご覧ください。

チェックリスト

- 条件ファイルで定義したパラメータを用いてコンポーネントのプロパティ値を更新できるように設

定できる。

- コンポーネントのプロパティ名の最後に$が付いているものと付いてないものの違いを説明できる。

3.8 実験記録ファイルの内容を確認しよう

製作中の実験もかなり体裁が整ってきたので、ここで一度実験記録ファイルの内容について解説しておきましょう。以下の手順 (図 3.20) で設定を変更してから実験を実行してください。ちょっと面倒ですが、100 試行すべて行ってください。

- data フォルダ内に作成されているファイルをすべて削除する。
- 実験設定ダイアログを開いて、hdf5 形式以外のすべての実験記録ファイルを出力するように設定する。すなわち、「xlsx 形式のデータを保存」、「CSV 形式のデータを保存 (summaries)」、「CSV 形式のデータを保存 (trial-by-trial)」にチェックを付ける (「pydat 形式のデータを保存」のチェックはチェックが付いている状態から変更できない)。
- 実験を実行し、実験情報ダイアログの participant に foo と入力して実行する。

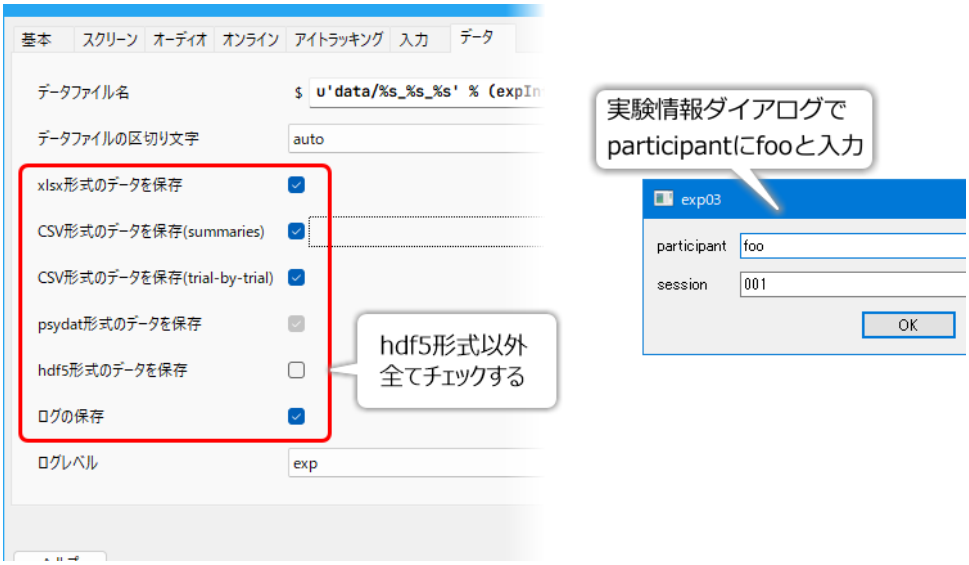


図 3.20　実験設定ダイアログを開いてすべての記録ファイルを保存するようにチェックして実験を最後まで実行してください。実験開始時の実験情報ダイアログの participant に foo と入力してください。

実験が終了したら、data フォルダの内容を確認してください。 表 3.2 に示したファイルができているはずです。ファイル名の先頭部分には participant に入力した文字列が代入されており、続いて実験を実行した時刻が続きます。participant に入力する文字列を工夫すると、データの整理が非常に楽になります。participant を空白のまま実験を実行すると、アンダースコア (_) の後に実行時刻が続くファイル名になります。participant にファイル名として使用できない文字列を指定しないように注意してください。

表 3.2　Builder が作成する実験記録ファイル (foo は実験情報ダイアログの participant に入力した文字列、yyyy_mm_dd_hhMM は年 (西暦)、月、日、時、分)

ファイル名	概要
foo_yyyy_mn_dd_hhMM.xlsx	条件毎に結果をまとめた Excel ファイルを保存する。実験設定ダイアログの「xlsx 形式のデータを保存」をチェックすると作成される。
foo_yyyy_mn_dd_hhMMtrials.csv	条件毎に結果をまとめた CSV ファイルを保存する。実験設定ダイアログの「CSV 形式のデータを保存 (summaries)」をチェックすると作成される。
foo_yyyy_mn_dd_hhMM.csv	試行毎に結果をまとめた CSV ファイルを保存する。実験設定ダイアログの「CSV 形式のデータを保存 (trial-by-trial)」をチェックすると作成される。
foo_yyyy_mn_dd_hhMM.psydat	Python の cPickle というモジュールを用いて出力された実験記録。自分で分析用 Python スクリプトを書く場合に用いる。実験設定ダイアログの「pydat 形式のデータを保存」をチェックすると作成される。
foo_yyyy_mn_dd_hhMM.log	実験中の PsychoPy の動作記録が保存されている。実験設定ダイアログの「ログレベル $」の選択肢によって内容が変化する。PsychoPy 自体の動作解析や開発などに用いる。通常のデータ分析向きではない。実験設定ダイアログの「ログの保存」をチェックすると作成される。

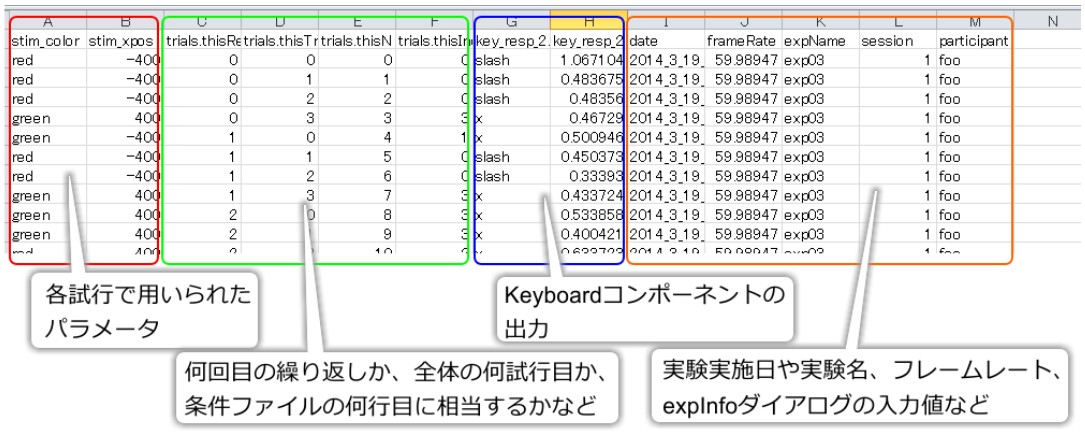
作成されるファイルのうち、拡張子が.log と.psydat のファイルはとりあえず無視してよいでしょう。.psydat ファイルは「データ分析は自作の Python スクリプトをでする」という人しか使わないでしょうし、.log ファイルは通常のデータ分析では必要のないファイルです。ただ、「実験が正常に動作していたのか？」という事が問題になった時には.log ファイルがあると役に立ちますので、学会や論文で公表する可能性がある実験の.log ファイルは保存しておいた方がよいでしょう。

データ分析の際に基本となるファイルは、foo_yyyy_mn_dd_hhMM.csv です。拡張子の.csv は、このファイルの保存形式がカンマ区切りのテキストファイル (comma-separated values; CSV) であることを意味しています。Excel も LibreOffice Calc も CSV ファイルを開くことができますので、以下の解説ではこれらのアプリケーションで CSV ファイルを開くという前提で進めます。

foo_yyyy_mn_dd_hhMM.csv を Excel で開くと (LibreOffice Calc でも同様、以下省略)、シートの 1 行が 1 回の試行に対応する形で実験記録が保存されていることを確認できます。このファイルは、実験設定ダイアログで「CSV 形式のデータを保存 (trial-by-trial)」をチェックしていると作成されるので、以下 trial-by-trial 記録ファイルと呼ぶことにします。1 行目は各列の見出しで、2 行目が第 1 試行、3 行目が第 2 試行という具合に 101 行目 (第 100 試行) までデータがあることを確認してください。列数は条件ファイル数に含まれるパラメータ数やルーチンに配置されたコンポーネントによって変化しますが、一般的に以下の内容が含まれます (図 3.21)。

各試行で用いられたパラメータ　　条

件ファイル内で定義されているパラメータの内、どの値が使用されたかが示されています。この情報を



stim_color	stim_xpos	trials.thisRe	trials.thisTr	trials.thisN	trials.thisIn	key_resp_2.	key_resp_2	date	frameRate	expName	session	participant
red	-400	0	0	0	0	slash	1.067104	2014_3_19_	59.98947	exp03	1	foo
red	-400	0	1	1	0	slash	0.483675	2014_3_19_	59.98947	exp03	1	foo
red	-400	0	2	2	0	slash	0.48356	2014_3_19_	59.98947	exp03	1	foo
green	400	0	3	3	3	x	0.46729	2014_3_19_	59.98947	exp03	1	foo
green	-400	1	0	4	1	x	0.500946	2014_3_19_	59.98947	exp03	1	foo
red	-400	1	1	5	0	slash	0.450373	2014_3_19_	59.98947	exp03	1	foo
red	-400	1	2	6	0	slash	0.33393	2014_3_19_	59.98947	exp03	1	foo
green	400	1	3	7	3	x	0.433724	2014_3_19_	59.98947	exp03	1	foo
green	400	2	0	8	3	x	0.533858	2014_3_19_	59.98947	exp03	1	foo
green	400	2	1	9	3	x	0.400421	2014_3_19_	59.98947	exp03	1	foo

図 3.21　trial-by-trial 記録ファイルの内容。「各試行で用いられたパラメータ」は条件ファイルを使用した場合のみ、「Keyboard コンポーネントの出力」は Keyborad コンポーネントを使用した場合のみ出力されます。

含む列の数は条件ファイルの列数に対応しています。実行した実験で条件ファイルが使用されていない場合は出力されません。

繰り返しに関する情報　そ

の行の試行が何回目の繰り返しの何試行目にあたるか、全体を通して数えて何試行目にあたるかといった情報が示されています。 図 3.21 の trials.thisRepN は何回目の繰り返しか、trials.thisTrialN はその繰り返しにおける何試行目か、trials.thisN は全体を通して何試行目か、trials.thisIndex はその試行で用いられたパラメータが条件ファイルの何個目の条件に相当するかを示しています。Python では「順番は 0 番目から数える」という規則がありますので、第 1 試行が 0 と表記されていることに注意してください。また、列見出しの trials の部分は Builder におけるループの名前に対応しています。

刺激の開始・終了に関する情報　各

試行で刺激が (描画) 開始・終了された時刻が、実験開始を 0 秒として秒単位で出力されています。何か実験中に異常があって描画タイミングがおかしくなったりしていなかったかをこれらの値で確認することができます。終了時刻が設定されていない刺激では終了時刻が None となります。この情報が不要な場合は「 3.12.7: コンポーネントの開始・終了時刻をデータファイルに出力しないようにする 」を参照してください。

キー押しに関する情報　実

験に Keyboard コンポーネントが含まれていなければ出力されません。key_resp_2.keys は押されたキーのキー名、key_resp_2.rt は Keyboard コンポーネントが有効になってからキーが押されるまでの時間を示しています。単位は秒です。列見出しの key_resp_2 の部分は Builder における Keyboard コンポーネントの名前に対応しています。なお、時間が小数点以下ものすごい桁数まで出力されていますが、計測に詳しい方は「一体この値はどの程度精度があるのか」と不安になるかもしれません。PsychoPy Builder が出力する時間の精度については「 3.12.6:PsychoPy の時間計測の精度について (上級) 」をご覧ください。

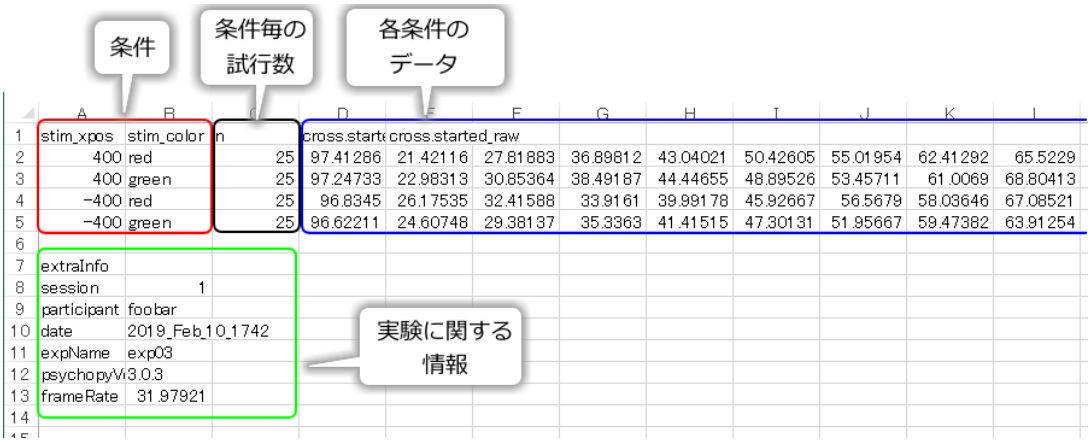
キー押し検出開始・終了に関する情報　実

験に Keyboard コンポーネントが含まれていなければ出力されません。Keyboard コンポーネントがキー押し検出を開始・終了した時刻が、実験開始を 0 秒として秒単位で出力されています。終了時刻が設定されていない場合は終了時刻が None となります。その他の実験に関する情報実験実施日や実験名、実

験情報ダイアログで入力した値、使用した PsychoPy のバージョン、フレームレートなどが示されています。フレームレートとは、1 秒あたりに描画したフレーム数のことで、モニターのリフレッシュレートと大きく異なる場合はスクリーンの描画に問題が生じている可能性があります。リフレッシュレートとフレームについては「2.12.6: 時刻指定における frame について」をご覧ください。特にこの値がモニターのリフレッシュレートより大幅に小さい場合は、スクリーンの描画に遅延が生じていて刺激が適切な時刻に表示されていない恐れがあります。

今回の実験では、刺激が提示される位置と反応する指の関係が反応の速さや正確さにどのように影響するかを調べるのが目的ですから、刺激の位置と色の組み合わせ別に、正答率と反応時間を計算すればよいでしょう。すでに自分自身でいろいろなアプリケーションを用いて実験を行っている方は各自で好みの方法で分析していただければよいですが、こういった分析が初めての方はとりあえず stim_color と stim_xpos でデータを並べ替えしてみましょう。するとそれぞれの条件に対応する試行が集まりますので、ワークシート関数を用いて平均値を求めたり、条件毎に x キーと slash キーが押された回数を数えれば計算することができますが、平均値の計算と比べれば手間がかかります。実は、Builder には参加者が押したキーが正答であったか否かを判定して記録する機能があり、これを使うと正答率の計算は楽になります。次節ではこの方法を学びます。

続いて「xlsx 形式のデータを保存」をチェックすることで出力される拡張子が xlsx のファイルと、「CSV 形式のデータを保存 (summaries)」をチェックすることで出力される拡張子が csv のファイルについても確認しておきましょう。これらはいずれも条件ファイルで定義された実験条件ごとに結果を 1 行に要約したもので、内容はほぼ同じです。拡張子が xlsx のファイルを開いてみてください。 図 3.22 のような内容になっているはずです。



	A	B	C	D	E	F	G	H	I	J	K	L
1	stim_xpos	stim_color	n	cross.start	cross.started_raw							
2	400	red	25	97.41286	21.42116	27.81883	36.89812	43.04021	50.42605	55.01954	62.41292	65.5229
3	400	green	25	97.24733	22.98313	30.85364	38.49187	44.44655	48.89526	53.45711	61.0069	68.80413
4	-400	red	25	96.8345	26.17535	32.41588	33.9161	39.99178	45.92667	56.5679	58.03646	67.08521
5	-400	green	25	96.62211	24.60748	29.38137	35.3363	41.41515	47.30131	51.95667	59.47382	63.91254
6												
7	extraInfo											
8	session	1										
9	participant	foobar										
10	date	2019_Feb_10_1742										
11	expName	exp03										
12	psychopyVi	3.0.3										
13	frameRate	31.97921										
14												

図 3.22　xlsx 記録ファイルの内容。データが 1 条件 1 行でまとめられています。

ひとつの実験条件が 1 行にまとめられるため、本章の例では 4 行となります。条件ファイルの行数と同じですね。各行の最初に条件ファイルに記述されているパラメータが示されています。パラメータの次には n という見出しの列があり、各条件に対応する試行が何試行あったかが記されています。 図 3.22 ではすべて 25 となっています。一般に、n の値はループの **[繰り返し回数 $]** に一致します。さらに続いて、各条件のデータが数十列にわたって出力されています。これらの行の下には、実験の実施時刻やフレームレート、実験情報ダイアログの内容といった実験に関する情報が記されています。この記録形式は **[繰り返し回数 $]** が増えると非常に列数が多くなるという問題があるのですが、条件毎の正答率や平均反応時間といった情報が出力されるので、trial-by-trial 記録ファイルから自分で計算する手間が省けるという利点があります。章末の「3.12.8:「CSV 形式のデータを保存 (summaries)」・「xlsx 形式のデータを保存」で作成される記録ファイルの形式」に詳しく解説しておきますので、興味がある方はそちらをご覧ください。どちらを使うかは好みの問題ですが、本書では trial-by-trial 記録ファイルを使用します。

チェックリスト

- expInfo の participant に設定した文字列が記録ファイル名にどのように反映されるかを説明できる。
- data フォルダに作成される拡張子 log と psydat のファイルに何が記録されているか、概要を説明することができる。
- CSV ファイルの CSV とは何の略であり、何を意味しているか説明することができる。
- trial-by-trial 記録ファイルから、各試行で用いられたパラメータの値を読み取ることができる。
- trial-by-trial 記録ファイルから、各試行において押されたキーのキー名と反応時間を読み取ることができる。
- trial-by-trial 記録ファイルに記録されている反応時間の計測開始時点 (0.0 秒になる時点) がどのように決まるか答えることができる。
- trial-by-trial 記録ファイルから実験情報ダイアログで入力した値を読み取ることができる。
- trial-by-trial 記録ファイルから実験実行時のフレームレートを読み取ることができる。
- 分析時に未変更の記録ファイルを保存しておいた方がよい理由を説明できる。

3.9 反応の正誤を記録しよう

Builder の画面に戻って、exp03.psyexp を開いてください。そして Keyboard コンポーネントのプロパティ設定ダイアログを開きましょう。 図 3.23 に示す通り **[正答を記録]** という項目がありますが、この項目をチェックすると **[正答]** という項目に入力できるようになります。ここに正答となるキー名を入力しておくと、Builder が押されたキーが正答キーと一致するか否かを判定して記録ファイルに保存してくれます。x キーが正解ならば'x'、 / キーが正解であれば'slash' です。GO/NOGO 課題のように「押さないのが正解」の場合は None と入力します。None の前後にクォーテーションマークが付いていないことに気を付けてください。

多くの心理学実験では、条件によって正答となる反応が変化します。従って、**[正答]** に入力するキー名は条件毎に変化させる必要があります。「条件毎に変化する」となると、条件ファイルの出番です。刺激が赤色の時に x キー、緑色の時に / キーを押すのが正答ですが、この関係は作成済みの条件ファイルに含まれている値からはわからないので、正答のキー名を示すパラメータ列を条件ファイルに追加する必要があります。刺激の位置と色を変化させた時にはまず条件ファイルの作成から始めましたが、どのようなパラメータ名を使用するかすでに決めているのであれば、Builder 側の作業から始めてしまっても問題はありません。correct_ans という列を条件ファイルに追加して、そこへ正答キー名を入力することにしましょう。

Keyboard コンポーネントのプロパティ設定ダイアログの **[正答]** に「correct_ans から値を代入して繰り返し毎に更新する」ように設定しないといけないのですが、ここで条件ファイルの参照について皆さんがきちんと理解しているかテストです。 **[正答]** に入力する値は correct_ans ですか、それとも$correct_ans ですか？ 答えは$付きの$correct_ans です。なぜ$が必要かをきちんと説明できない方は、もう一度$記号について復習しておきましょう。繰り返し毎の更新でもう一つ注意すべき点として「「繰り返し毎に更新」を忘れずに選択する」というものがありましたが、 **[正答]** の右側にはそもそもプルダウンメニューがありません。この項目に関しては、「繰り返し毎に更新」を選ばなくても更新が行われます。

[正答] の設定が終わったら、今度は Excel を開いて条件ファイルを編集しましょう。correct_ans と 1 行目に書かれた列を追加して、正答キー名を入力してください。正答キーは刺激の色と対応しているのですから、

図 3.23　反応の正誤を記録するよう設定する手順。

stim_color が red の行は x、green の行は slash が正答キーです (図 3.24)。入力したら条件ファイルを保存しましょう。条件ファイルの名前や保存場所を変更していなければ、Builder で条件ファイル名を設定し直す必要はありません。試しに Builder に戻ってフローペインのループを左クリックしてループのプロパティ設定ダイアログを表示させると、相変わらずパラメータは stim_color と stim_xpos の二つと表示されています。しかし、実行するときちんと新しく追加したパラメータも読み込まれません。正しく表示されないと気持ち悪いという方は、もう一度条件ファイルを設定し直してください。正しく stim_color、stim_xpos と correct_ans の 3 パラメータが表示されるはずです。

	A	B	C
1	stim_color	stim_xpos	correct_ans
2	red	-0.7	x
3	green	-0.7	slash
4	red	0.7	x
5	green	0.7	slash
6			

図 3.24　条件ファイルに correct_ans の列を追加し、正答キー名を入力します。

Keyboard コンポーネントと条件ファイルの修正が終わったら、実験を実行してみましょう。きちんと 100 試行終わるまで実行してください。終了したら、trial-by-trial 記録ファイルの内容を確認してみましょう (図 3.25)。trial-by-trial 記録ファイルには key_resp_2.corr という列が追加されています。この列の値が 1 の試行は正答で、0 の試行は誤答です。ということは、その平均値を計算すれば正答率 (0.0=全問不正解、1.0=全問正解) を計算できるというわけです。

以上で、サイモン効果を題材としたこの章の実験は「一応」完成しました。「一応」というのは、確かに最初に計画した目標は一通り達成しているのですが、実際にこの psyexp ファイルを使って参加者を募って実験す

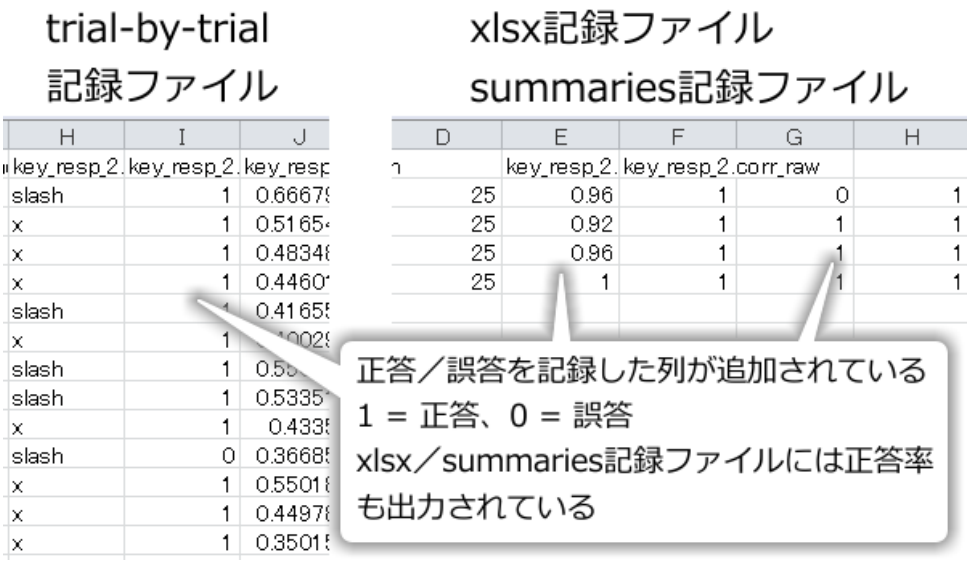


図 3.25　**[正答]** を設定すると記録ファイルに正答／誤答が記録されます。

るとなるといくつか不都合な点があるからです。例えば、実験を実行して実験情報ダイアログに値を入力すると、直ちに最初の試行が始まってしまいます。実験者が実験情報ダイアログに入力する場合、入力したらすぐに実験参加者にキーボードを渡さないと最初の試行の刺激に反応できないでしょう。また、実験参加者に対する教示も実験開始前にスクリーン上に表示しておきたいものです。終了時もいきなり Builder のウィンドウに戻るのではなく、何か一言実験参加者へのお礼や指示があると気が利いていて良いでしょう。次の節では、こういった気配りを実験に組み込む方法を解説します。

チェックリスト

- 正答／誤答を記録するように Keyboard コンポーネントを設定することができる。
- キーを押さないことが正答となるように設定することができる。
- 正答となるキー名を条件ファイルから読み込んで繰り返し毎に更新することができる。
- trial-by-trial 記録ファイルから、各試行の正答／誤答を読み取ることができる。

3.10 ルーチンを追加して教示などを表示しよう

この節では、exp03.psyexp の実験開始時と終了時に教示などを表示させます。一口で教示を表示すると言っても「イラストで教示するのか、文章で表示するのかなど」いろいろと選択肢がありますが、ここでは文章で表示することにしましょう。

文章を表示するとなると Text コンポーネントの出番ですが、ここまで解説してきた方法だけではうまく表示することができません。というのも、今まで通りルーチンペイン上に Text コンポーネントを配置してしまうと、繰り返しの度に表示されてしまうので「実験開始時と終了時」ではなくて試行毎の表示になってしまうのです。繰り返しが始まる前や後に何かを表示したいときには、新しいルーチンを追加する必要があります。図 3.26 はフローペインを使って新しいルーチンを作成してフローに挿入する手順を示しています。まず、フローペイン左端の Insert Routine をクリックします。すると (new) と trial という二つの項目があるポップアップメニューが表示されます。「繰り返しを設定しよう」の節で少しだけ触れましたが、PsychoPy で実験を新規作成した時に最初から表示されているルーチンは trial という名前がついています。ポップアップメニューの

trial という項目を選択すると、trial ルーチンがもう一つフローに挿入されます。同じルーチンを何個も挿入してどうするのかと思われるかもしれませんが、その例は練習問題で触れることにしましょう。

新しくルーチンを作成してフローに挿入する場合は、(new) を選択します。すると挿入するルーチン名を入力するダイアログが表示されますので、わかりやすい名前を決めて **[新しい Routine の名前]** 入力しましょう。名前はパラメータ名に使用できる文字列と同様の規則を満たすものでなければいけません。また、すでにパラメータ名やコンポーネントの **[名前]** プロパティで使用している名前とも重複してはいけません。ここでは実験開始前に教示を表示するためのルーチンということで instruction という名前を付けておきます。**[Routine のテンプレート]** は、さまざまな用途を想定してすでにコンポーネントをいくつか配置済みのルーチン (テンプレートと呼びます) を挿入するもので、初期状態で Basic, Misc, Online, Trial と分類された十数個のテンプレートが登録されています。便利な機能ではありますが、テンプレートを適用した後に手作業で設定しないといけない項目があったり、テンプレートの機能を活かすためにその前のルーチンで適切な前処理をしておかないといけなかったりするなど、テンプレートの内容を熟知していないと使いこなすのは難しく、初心者向きではありません。そのため、 **本書ではコンポーネントが何も配置されていない「Basic:blank」というテンプレートを使うという前提で解説していきます。**ある程度 Builder に習熟してきたら、自分が実験でよく使う手続きをテンプレートとして登録するとよいでしょう。登録方法は「3.12.9: 独自のルーチンテンプレートを登録する方法 (上級)」をご覧ください。

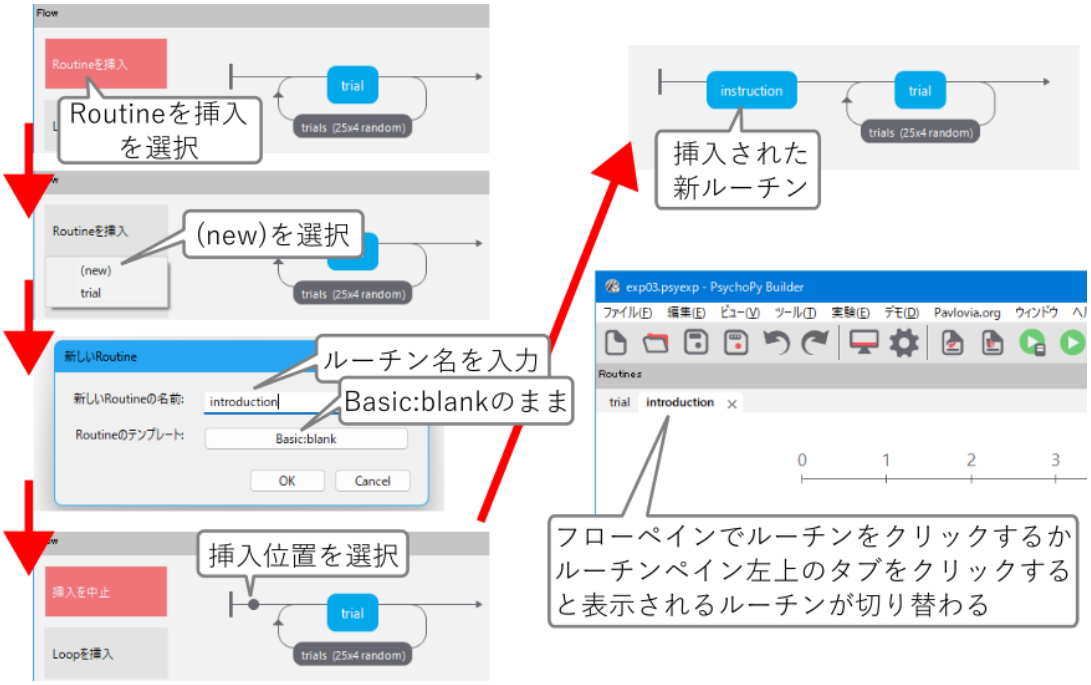


図 3.26　フローペインから新しいルーチンの追加を追加します。

名前を決定すると、新しいルーチンをフローのどの位置に挿入するかを指定しなければいけません。フロー上でマウスカーソルを動かすと、マウスカーソルが挿入可能な位置に重なった時に黒い丸が表示されますので、ループの前を選んで挿入してください。すると 図 3.26 の右上のように instruction と書かれた四角いアイコンがループの前に挿入されます。同時に、ルーチンペインの左上に instruction と書かれたタブが出現しているはずです。フローの instruction と書かれたアイコンか、ルーチンペイン左上の instruction と書かれたタブをクリックすれば、まだコンポーネントが配置されていない空白のルーチンが表示されるはずです。この空白のルーチンが新たに挿入された instruction ルーチンです。

instruction ルーチンの編集を始める前に、操作方法の確認を兼ねてもうひとつ実験終了時のメッセージを表示

するためのルーチンを追加しておきましょう。ルーチン名は thanks とします。ここでは三つの点を確認しておいてください (図 3.27)。

- フローペインの左端の Insert Routine を使ってフローにルーチンを挿入する時にポップアップメニューに instruction ルーチンが含まれていること。
- フロー上の挿入済みルーチンの上にマウスカーソルを重ねて右クリックするとポップアップメニューが表示され、とルーチンをフローから削除したり名前を変更したりできること。
- 上記の方法でフローからルーチンを削除しても、該当するルーチンはルーチンペインに残ったままであること。

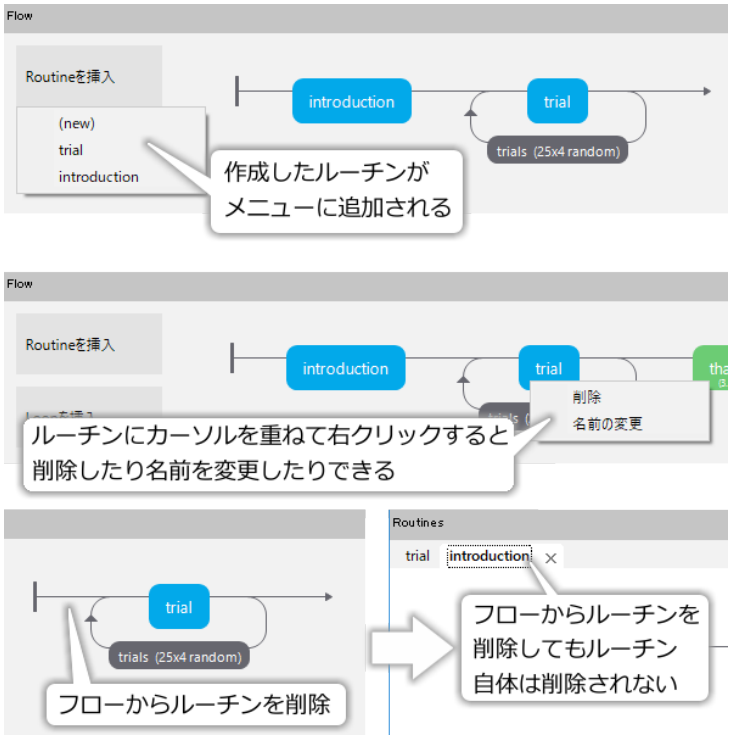


図 3.27　フローペインにおけるさまざまな操作。フローからルーチンを削除してもルーチンそのものは削除されません。

特に、フローからルーチンを削除してもルーチンそのものはルーチンペインに残っているという点は覚えておいてください。バージョン 2020.2.6 現在で Builder はフローにおけるルーチンの並び替えをサポートしていませんので、ルーチンの順番を変えたいときには一旦ルーチンをフローから削除して再挿入する必要があります。その際、一旦フロー上から目的のルーチンがなくなってしまっても全く問題ありません。

さて、instruction と thanks のルーチンを追加してそれぞれループの前と後ろに挿入ら、ルーチンの編集に入りましょう。まず instruction ルーチンでは、画面中央に以下のような教示を画面上に提示することにしましょう。

画面の中央に白い十字が表示されたあと、すこし遅れて赤色または緑色の円が表示されます。
刺激の色を見て、以下のキーをできるだけ速く間違わずに押してください。

赤色：左手人差し指でキーボードの x キー
緑色：右手人差し指でスラッシュ（ / ）キー

(次のページに続く)

(前のページからの続き)

```
スラッシュキーを押すと課題が始まります。
100 問終了するまで休憩できないので注意してください。
```

実験参加者の中には「スラッシュキー」と言われてもどのキーかわからない方もいるでしょうから、キーの確認も兼ねてスラッシュキーを押して実験を開始することにしてみましたが、スペースキーなどの実際の反応で使用しないキーを割り当てるのもよいでしょう。上記の文章を表示するための Text コンポーネントの設定については、ここまで作業してきた皆さんでしたらヒントなしでできると期待します。

ひとつ注意すべき点は、ルーチン終了時刻の指定です。教示に「スラッシュキーを押すと課題が始まります。」と書いてあるのですから、Keyboard コンポーネントを使用して、スラッシュキーが押されたらルーチンを終了するように設定すればよいでしょう。もちろん Text コンポーネントと Keyboard コンポーネントの **[終了]** は空白にしておきます。ここまではすでに解説済みのテクニックですが、このままだと記録ファイルに「課題を開始する時に押したスラッシュキー」まで保存されてしまいます (ただし 第 7 章 参照)。分析に必要な反応が記録されなくなってしまうわけではないので致命的な問題ではありませんが、分析と無関係なキー押しが記録されていると分析の際に思わぬ手間がかかる恐れがあります。

キー押しを記録したくない場合には、Keyboard コンポーネントのプロパティ設定ダイアログを開いて、**[記録]** を「なし」に設定しましょう (図 3.28)。その Keyboard コンポーネントで検出されたキー押しは一切記録されません。「その Keyboard コンポーネントで」と断ったとおり、ひとつの実験に複数の Keyboard コンポーネントを配置している場合、個々の Keyboard コンポーネントで独立に **[記録]** の設定を行うことができます。今回の実験の場合、instruction ルーチンに配置する Keyboard コンポーネント **[記録]** を「なし」にして、trial ルーチンに配置する Keyboard コンポーネントは「最後のキー」のままにしておけばよいでしょう。これで instruction ルーチンは完成です。

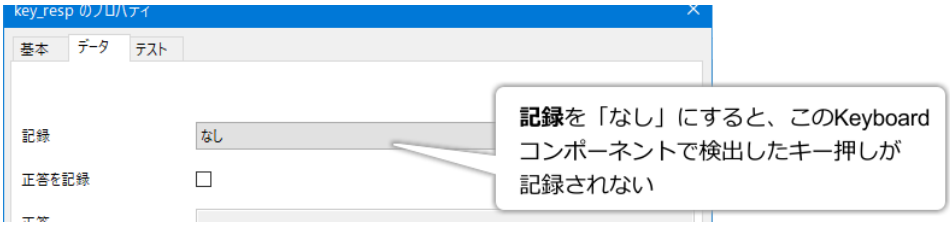


図 3.28　Keyboard コンポーネントで **[記録]** を「なし」に設定すると、その Keyboard コンポーネントで検出したキー押しが記録ファイルに出力されません。

続いて thanks ルーチンですが、ここでは Text コンポーネントを使用して「終了しました。ご協力ありがとうございました。」というメッセージを 3 秒間表示することにしましょう。Text コンポーネント以外のコンポーネントを配置する必要はありません。特に難しい点はないはずですが、ひとつ解説しておきたい点があります。Text コンポーネントの **[終了]** の設定を終えた後にフローペインを見ると、thanks ルーチンだけ緑色で表示されていて、小さく 3.00s と表示されているはずです (表示されていない場合はメニューの「ビュー」から「Flow を大きく」を選択してみてください)。対して、trial や instruction ルーチンは青色で表示されています (図 3.29)。緑色のルーチンは、中に含まれるすべてのコンポーネントの終了時刻が確定していて、ルーチン開始から終了までに要する時間が計算できることを示しています。青いルーチンは、キーが押されるまで終了しないルーチンのように、ルーチン開始から終了までの時間が実行時にならないと確定しないことを示しています。

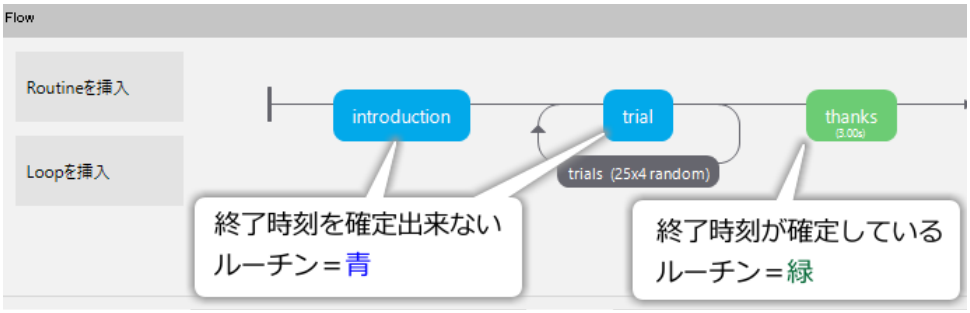


図 3.29　青いルーチンは終了時刻が確定していないことを、緑色のルーチンは終了時刻が確定していることを表しています。緑色のルーチンにはルーチンの実行時間の見積もりが表示されています。

以上でこの節の目標は達成しましたが、ルーチンの新規作成やフローへの挿入について解説したついでに、Builder ウィンドウ上部のメニューの「実験」という項目について補足しておきます (図 3.30 上)。メニューの「実験」からは、ルーチンの新規作成やコピー＆ペースト、フローへのルーチンとループの挿入の操作ができます。これらの操作のうち、ルーチンの新規作成とフローへのルーチンとループの挿入はフローペイン左端の「Routine を挿入」や「Loop を挿入」を用いる方法とほぼ同じです。ルーチンのコピー＆ペーストは、内容がわずかに異なるルーチンを複数作る必要がある時に便利な機能です。

試しにルーチンペインに trial ルーチンを表示している状態で、「実験」メニューから「Routine のコピー」を選択してください。続いてもう一度「実験」メニューを開いて「Routine の貼り付け」を選択しましょう。すると新しいルーチンの名前を入力するダイアログが表示されるので、他のルーチンやコンポーネントなどと名前が重複しないように新しい名前を入力してください。すると、trial と同じコンポーネントが配置された新しいルーチンが作成されます。ただし、同じコンポーネントが配置されていると言っても、各コンポーネントの名前は Builder がコピー元のルーチンに含まれるコンポーネントの名前と重複しないように自動的にアンダーバー + 数字が付けられるので注意してください (図 3.30 下)。

ルーチンの新規作成、コピー＆ペーストときたら、削除の方法についても知りたいところです。削除はルーチンペイン左上のタブから行います。現在選択中のタブにはルーチン名の右に×マークが表示されています。この×を左クリックすると、そのタブのルーチンが削除されます (図 3.31)。誤って削除してしまった場合はすぐにツールバーの元に戻すボタンをクリックしましょう。

以上で本章の解説を終えたいとと思います。簡単な実験であれば本章の内容だけで十分に実現できるはずです。締めくくりとして、本章で解説したテクニックを応用する練習問題を出しておきます。本章の内容をマスターできたと思った方は是非挑戦してください。

チェックリスト

- ルーチンを新たに作成してフローに追加することができる。
- 既存のルーチンをフローに追加することができる。
- フローからルーチンを削除することができる。
- フローペインの赤いルーチンと緑のルーチンが何に対応しているか説明することができる。
- 特定の Keyboard コンポーネントが検出したキー押しを記録しないように設定することができる。
- 既存のルーチンと同一の内容を持つ新しいルーチンをコピー＆ペーストの機能を使って作成するこ

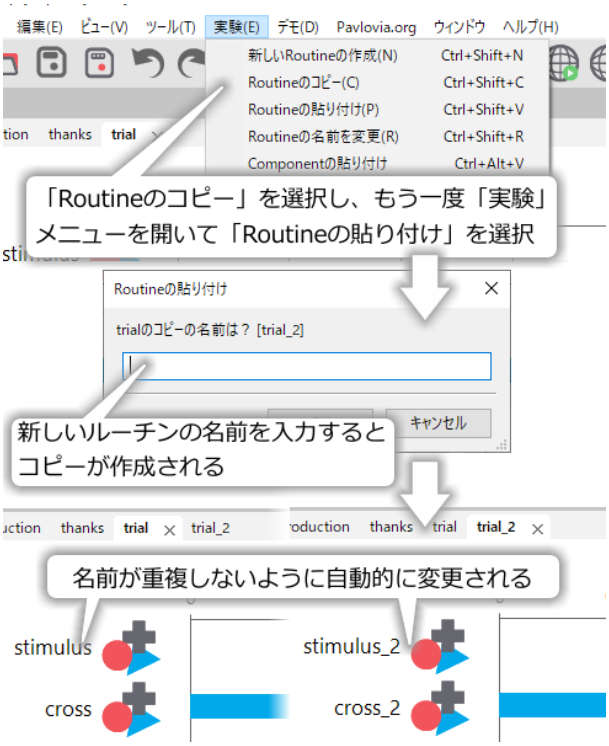


図 3.30　ルーチンのコピー＆ペースト。コンポーネントの名前が重複しないように、ペーストされたルーチン内のコンポーネント名には元のコンポーネント名にアンダーバー＋数字が自動的につけられます。

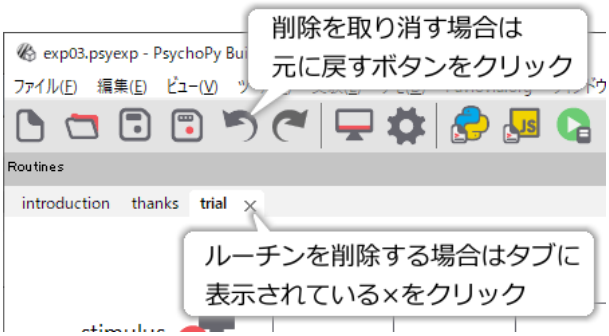


図 3.31　ルーチンを削除するにはタブに表示されている×をクリックします。誤って削除してしまったなどの理由で削除を取り消したい場合は元に戻るボタンを使います。

とができる。

3.11 練習問題：練習試行を追加しよう

先ほどは instruction ルーチンを終えて実験を開始する際に、実験参加者にスラッシュキーを確認させることも兼ねてスラッシュキーで instruction ルーチンを終了するようにしました。しかし、本来であれば本試行と同様の手続きの練習試行を数回体験させてから実験を開始するべきです。練習と本試行を別々の psyexp ファイルとして作成するのも良いのですが、ここでは練習問題ということで単一の psyexp ファイルで練習試行と本試行を実施するように改造してみましょう。

図 3.32 はこの練習問題で作成する実験の流れです。本試行と終了メッセージは本章で作成した実験から変更する必要はありません。教示はもうひとつルーチンを追加する必要があるでしょう。この練習問題のポイント

は、いかに手間をかけずに練習試行を作成するかです。ヒントは「フローの中に同一のルーチンを複数回挿入できる」という点と、「練習試行と本試行はループ回数が違うだけで条件は同じ」という点の二点です。これでもまだピンと来なければ、 図 3.32 の一番下のフローも参考にしてください。

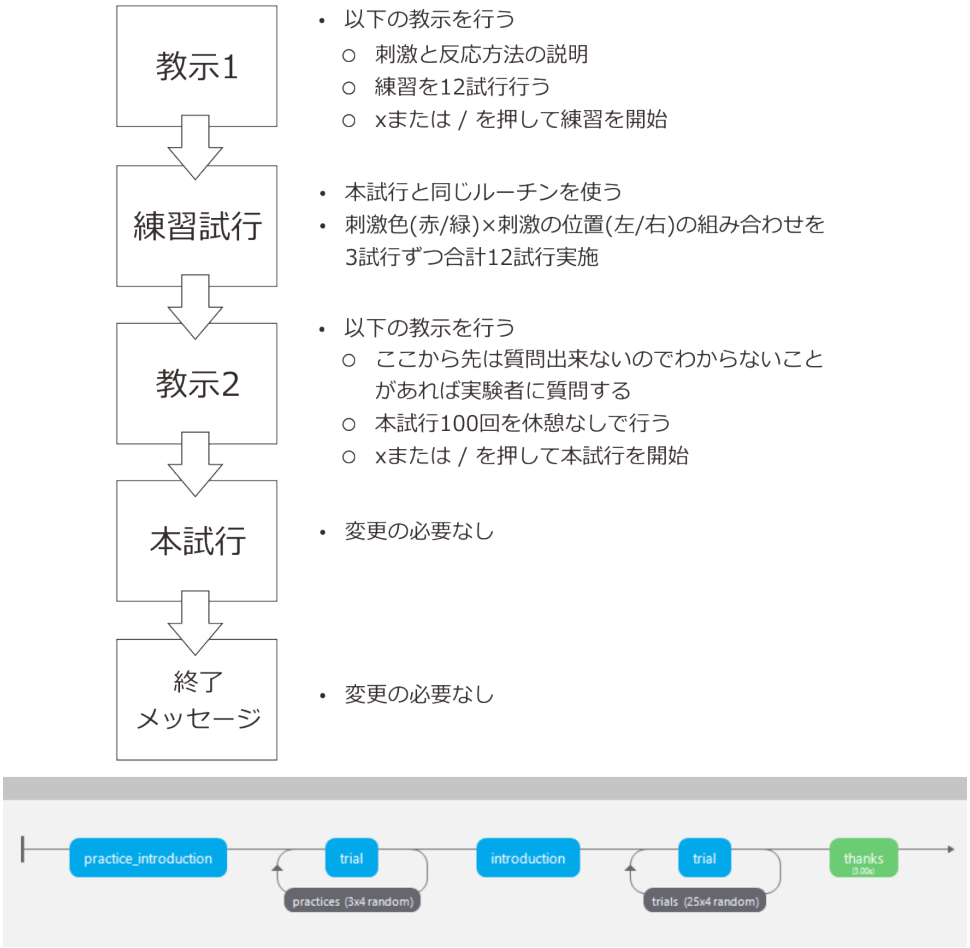


図 3.32　練習問題。本試行を開始する前に 12 試行の練習を行うように改造してみましょう。一番下は作成例のフローを示しています。

3.12 この章のトピックス

3.12.1 自分のキーボードで使えるキー名を確かめる

Builder を起動し、ルーチンペインに Keyboard コンポーネントと Text コンポーネントを配置して以下のように設定してください。

- Keyboard コンポーネント
  - **[名前]** を `test_keyboard` とする。
  - **[終了]** を空白にする。
  - **[Routine を終了]** のチェックを外す。
  - **[検出するキー $]** を空白にする。
- Text コンポーネント

  - **[終了]** を空白にする。
  - **[文字列]** に `$test_keyboard.keys` と入力する。$記号やピリオドには重要な意味があるので忘れずにつける。間に空白文字を含んではいけない (第 4 章 、第 6 章 参照)。
  - **[文字列]** の横にある「更新しない」と書かれたプルダウンメニューを開いて「フレーム毎に更新する」を選択する。
  - 一度実行してみて、字が小さすぎたなら **[文字の高さ $]** を調節する。

- 実験設定ダイアログ
  - 「スクリーン」タブの **[フルスクリーンウィンドウ]** のチェックを外す。
  - 「基本」タブの **[ESC キーによる中断]** のチェックを外す。
  - 念のため「基本」タブの **[実行モード]** を Piloting にする。

以上の設定を行ったら実験を保存して実行し、適当にキーボードのキーを押してみてください。押したキーに応じてスクリーン中央に文字列が表示されるはずです。これが押したキーに対応するキー名です。**ESC キーによる実験の中断を無効にしているので、Runner から中断するしかありません。**フルスクリーンウィンドウを無効にしているうえに、Pilot モードで実行しているので、Runner にアクセスできないということはないはずです。万一両方とも忘れてしまった場合 (つまりフルスクリーンウィンドウかつ実行モードで実行した場合)、Runner から終了するのはかなり困難です。OS の機能で強制的にウィンドウを切り替えたりプロセスを kill したりといったことができればよいですが、どうしてもうまくいかず PC を再起動するしかなくなる可能性もありますので、くれぐれもフルスクリーンモードの無効化と Pilot モードでの実行を忘れないようにしてください。

3.12.2 Builder で使用できない名前を判別する

Builder を起動し、適当なコンポーネントをルーチンに配置して、 **[名前]** に名前として使用したい文字列を入力してみましょう。文字列が Python の予約語や PsychoPy のモジュール名、Builder の予約語に一致する場合は、 図 3.33 のようにプロパティ設定ダイアログの下部に赤色でエラーメッセージが表示されます。ダイアログ左下の OK ボタンをクリックできなくなりますので、意図せずに予約語を名前に使用してしまった場合でもまず気づくでしょう。

この方法でチェックできるのは予約語などと一致する場合と、自分で定義した他のコンポーネントの **[名前]** や条件ファイルのパラメータ名などと一致している場合です。ただし、 第 4 章 で紹介する、読み込む条件ファイルを実行時に切り替える方法を用いている場合は、Builder は名前の重複を判断できません。実験作成者が管理する必要があります。

3.12.3 無作為化と疑似乱数

予測不可能な順序に並べられた数の列のことを乱数 (または乱数列) と呼びます。乱数には、一様乱数や正規乱数など、値の出現確率の違いによってさまざまな種類があります。コンピュータゲームなどにおいて人間にとって予測困難な動作を実現する際には、乱数が非常によく用いられます。例えば「地図上を歩き回っていると予測困難なタイミングで敵と遭遇する」というゲームの場合、「地図上を一歩歩くごとに乱数の先頭から順番に値をひとつ取り出し、その値を 20 で割った余りが 0 であれば遭遇する」といった処理を行うと上手くい

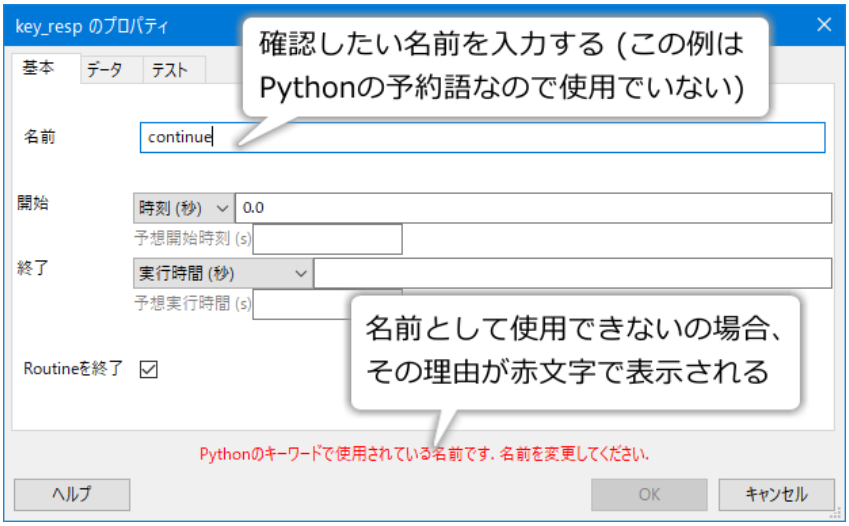


図 3.33　Builder で適当なコンポーネントをルーチンに配置して **[名前]** に文字列を入力することによって、その文字列が予約されていないかを確認することができます。

きます。10 で割ると遭遇しやすくなり、100 で割るとめったに遭遇しないといった具合に遭遇頻度の調節もできます。

このように便利な乱数ですが、実は完全な乱数を発生させるのは難しいことで、代用のために「乱数っぽく見える」数列を作成する方法がいろいろと考案されてきました。「乱数っぽく見えるけれども、実は確定的な計算によって生成されている数列」のことを疑似乱数と呼びます。疑似乱数はシード (seed) と呼ばれるパラメータを持っており、シードの値が一定であれば毎回同じ数列が得られます。あまり良い例ではないかも知れませんが、関数 y=sin kx (k は定数) において、k の値が一定であれば x=1,x=2,x=3,…の時の y の値は何度計算しても同じです。しかし、k の値を変えれば x=1,x=2,x=3,…の時の y の値は変化します。y=sin kx 程度の式であれば生成される数列に規則性があることはすぐにわかりますが、工夫して疑似的に乱数としての性質を持つ数列を生成するように考えられた式が疑似乱数というわけです。

Builder では、**[乱数のシード $]** が空白であれば、実験実行時に Builder が適当なシードを決定して疑似乱数を発生させます。**[乱数のシード $]** に整数が与えられていれば、それをシードとして疑似乱数を発生させます。そして、生成した疑似乱数を用いて条件の実行順を並び替えます。ですから **[乱数のシード $]** を指定すれば必ず同じ順番で繰り返しが行われるのです。

3.12.4 Loop のプロパティ設定ダイアログの [使用する行 $] について

この機能を使いこなすには 第 5 章 以降で学ぶ知識が必要なので、Python の文法をご存じない方はひとまずこの機能は無視して先へ進むことをお勧めします。

[使用する行 $] は、条件ファイルで定義されている条件のうち、一部分だけを利用して実行したい場合に利用します。1 行に 1 条件が定義されているので、使用したい条件が書かれている行番号をカンマ区切りなどで列挙します。注意が必要なのは、Python は順番を表現するときは最初を「0 番目」と書く点です (第 9 章)。

具体的な使用方法ですが、例えば今回の実験で条件ファイルの 1 行目と 3 行目に書かれた条件のみを実施する場合には

```
"0,2"
```

と書きます。" "を忘れないでください。他にもリストやタプル (第 5 章 、 「9.14.3: リストとタプル」) を用いて

```
(0,2)
[0,2]
```

という書き方も受け付けます。この場合は" "が不要です。連続する多数の行を指定する場合はスライス (第 11 章) を使用します。

```
[:20]
```

筆者は普段この機能を利用しないのですが、第 5 章 で学ぶ「実験ダイアログからの情報の取得」を利用して、実行時に利用する行を指定するといった使い方が便利そうです。また、数か月、数年後に実験を再確認する必要が出てきた場合に備えて、条件ファイルにコメントを書いておいて、この機能を用いてコメント行を読み飛ばすといった使い方もできるでしょう。

3.12.5 $を含む文字列を提示する

本文中で、Text コンポーネントの **[文字列]** プロパティに$を入力すると条件ファイル内で定義されたプロパティの値を代入することができると述べました。では、Text コンポーネントで「$を含む文字列を表示させたい」場合はどうすればいいのでしょうか。ただ$を書くと Builder に「名前」として判断されてしまうため、目的を達成できません。一番シンプルな解決策は、「半角ではなく全角の$を使う」というものです。コンピュータにとって半角と全角の$は全く別の文字なので、問題なく表示することができます。

どうしても半角の$を使いたい、という場合はエスケープ文字を利用する方法があります。エスケープ文字とは半角のバックスラッシュ (\ ) のことで、日本語フォントでは円記号 (¥) として表示されます。こう書くと初心者の方は「エスケープ文字って何？」と「日本語フォントでは円記号で表示されるってどういう意味？」という二つの疑問を持たれることと思いますが、まずエスケープ文字の説明から始めましょう。

エスケープ文字の役割は、Builder における$の役割と似ています。例えば Python のスクリプトにおいては、Target という文字列がデータとして文字列を表すのか、データに対してつけられた名前 (変数) を指すのかを区別するために、文字列の前後を半角のシングルクォーテーションまたはダブルクォーテーションで囲みます。

表 3.3　変数と文字列の区別

Target	Target という変数
"Target"	Target という文字列

言語によってダブルクォーテーションのみしか使えないなどの違いがありますが、このような表記は多くのプログラミング言語で用いられています。さて、Python の場合、シングルクォーテーションやダブルクォーテーション自体を含む文字列、例えば

> He said, "Please respond as quickly and precisely as possible."

という文字列はどう表記すればよいのでしょうか。何も考えずに前後にダブルクォーテーションを付けてしまうと

"**He said, "** Please respond as quickly and precisely as possible. **""**

となってしまい、"He said, "までがダブルクォーテーションで囲まれていて、Please から possible までは囲まれてないことになってしまいます。最後に""が残りますが、これは「何も文字がない文字列」と解釈されます。数字の 0 のようなもので、空文字列と呼ばれます。

この問題の回避方法として、「文字列中に含まれるダブルクォーテーションの前には\を付ける」という記法が用意されています。これを使うと、先の文字列は

"He said, **\"** Please respond as quickly and precisely as possible. **\"** "

と表記することができます。多くのプログラミング言語において、\には「その直後に続く文字 (列) を通常とは異なる方法で解釈させる」という役割があります。\などの特別な文字によって、後続の文字を通常とは異なる方法で解釈させることをエスケープと呼び、エスケープによって異なる意味を示す文字列をエスケープシーケンスと呼びます。エスケープシーケンスを開始する記号 (ここでは\) はエスケープシーケンスプレフィックスやエスケープ文字などと呼びます。He said,…の例文では、\"と書くことによって\に続く " が「文字列の終了を表す」という通常の意味を失って、単なる " という文字だと解釈されています。もし皆さんが将来プログラミング言語を本格的に学習するのであれば、他にもさまざまなエスケープの例に出会うことになるでしょう。なお、Python に限って言えば、文字列をシングルクォーテーションで囲んでも良いことと、シングルクォーテーションで囲まれた文字列の中ではダブルクォーテーションを自由に使うことができるということも付け加えておきたいと思います。

以上を踏まえて Builder の Text コンポーネントの話に戻りましょう。Text コンポーネントで$を含む文字列を表示したい場合にも、\によるエスケープが利用できます。つまり、\$と書けば$を表示することができます。ぜひ試してみてください。なお、本文中で述べた「Builder は条件ファイル内に書かれた$記号を単なる文字として解釈する」というルールはここでも有効ですので、Text コンポーネントの **[文字列]** プロパティの値を条件ファイルから読み込んで表示する場合にはエスケープをする必要はありません。$を含む文字列をそのまま表示することができます。

最後に蛇足ですが、「日本語フォントでは円記号で表示されるってどういう意味？」という疑問にもお答えしておきましょう。コンピュータでは、文字を管理するためにすべての文字に数値を割り当てています。例えばアルファベットの大文字の A は 0x41、B は 0x42、小文字の a は 0x61、b は 0x62 といった具合です。この数値を文字コードと呼びます。困ったことに、歴史的な経緯から現在のコンピュータでは複数の文字コードが使用されており、ある文字コードで書かれた文書を別の文字コードで解釈すると、本来意図されていた文字とは全く別の文字として解釈されてしまいます。昔の電子メールで時々生じていた「文字化け」の原因のひとつが送信側と受信側で使用している文字コードの違いでした。今回問題になっているバックスラッシュ (\ ) は、初期のコンピュータからずっと利用され続けている ASCII コードという文字コードで 0x5C に割り当てられています。一方、日本のコンピュータで最初に用いられた JIS 規格の文字コードでは、大半が ASCII と共通のコードが割り当てられていたのですが 0x5C には円記号 (¥) が割り当てられていました。結果として、同じ 0x5C というコードの文字が、解釈に用いるコードによってバックスラッシュになったり円記号になったりするようになってしまったのです。現在のコンピュータでは、欧文フォントを用いて表示するとバックスラッシュで表示され、日本語フォントを用いて表示しなおすと円記号になるといった具合に使用するフォントによって表示が切り替わるという現象が見られます。

3.12.6 PsychoPy の時間計測の精度について (上級)

本文中でも述べた通り、PsychoPy の記録ファイルを確認すると、キーが押された時刻が 0.766847908126 秒のように小数点以下かなりの桁数が出力されています。この数値は一体何桁目まで信頼できるのでしょうか。

残念ながらこの疑問の答えは PsychoPy を実行しているハードウェアや OS、デバイスドライバに依存するので一概には答えられません。筆者が主にプログラムの開発等に使用している PC では、約 2.6MHz のタイマーカウンターが使用されているようです。分解能は約 260 万分の 1 秒ですから 3.8×10^{-7} 、0.38 マイクロ秒です。Web 上でいろいろな資料を確認する限り、この周波数はあまり高くない部類で、10MHz を超えるタイマーカウンターが利用できる場合もあるようです。反応時間を指標とした心理学実験の論文では「ミリ秒」の小数点 1 桁まで報告されることがしばしばありますが、その用途には十分な分解能があると言えます。

むしろ注意しないといけないのは、実験参加者がキー押しの動作をはじめてから実際に「キーが押された」と PsychoPy のプログラムまで到達する時間や、PsychoPy が「刺激を画面に描画せよ」という命令を実行してから、実際にそれがモニターに伝わって画面に表示されるまでの時間です。これらの値もやはりハードウェアや OS、デバイスドライバに依存するので一概に言えないのですが、「ゲーム用」などと謳われている高性能なキーボードやモニターでなければ、数十ミリ秒もの時間を要することも珍しくありません。

- 先行研究の結果と比較して平均反応時間が十数ミリ秒にわたってずれている場合は、機材が原因である可能性も考慮する。可能であれば先行研究と同一条件の追試を行っておき、そこで大きなずれがないか確認しておくことが望ましい。
- 複数の実験参加者を呼んで並行して実験を行う場合、実験に使用するハードウェアやソフトウェアは可能な限り統一する。

といった点を守っておけば、多くの実験において時間計測の精度が大きな問題となることはないと思います。

3.12.7 コンポーネントの開始・終了時刻をデータファイルに出力しないようにする

刺激やキーボードといったコンポーネントの開始・終了時刻をデータファイルに出力する機能は、何か実験中に問題が生じた場合に、どの時点で生じたのかを知る重要な手がかりを与えてくれます。しかし、ルーチン内に配置されているがコンポーネントが多い場合、コンポーネント数に応じてデータファイルの列数が増えてしまうため、非常にデータファイルが見難くなる場合があります。

開始・終了時刻の出力を抑制するには、各コンポーネントのプロパティ設定ダイアログの「データ」タブにある **[開始・終了時刻の保存]** のチェックを外します (図 3.34)。チェックを外したコンポーネントだけが出力されなくなりますので、実験の流れを把握する上で重要なコンポーネントだけチェックを残して他はチェックを外すとよいでしょう。

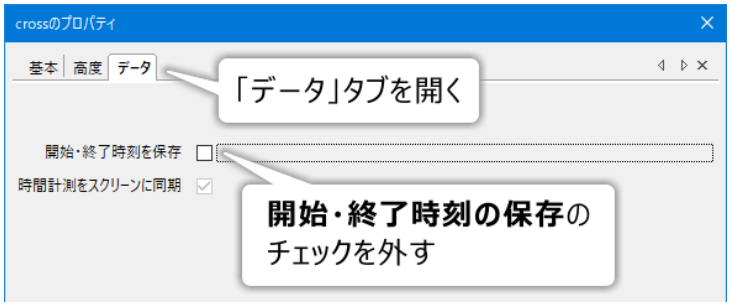


図 3.34 **[開始・終了時刻の保存]** のチェックを外すとそのコンポーネントの開始・終了時刻が出力されなくなります

3.12.8 「CSV 形式のデータを保存 (summaries)」・「xlsx 形式のデータを保存」で作成される記録ファイルの形式

本文で述べた通り、これらの記録ファイルでは条件ファイルで記述された条件ひとつにつき 1 行に結果が要約されています。最初にパラメータが出力され、続いて繰り返し数が出力されていることも本文で述べました。ここではさらにその右側に数百列にわたって出力されている内容について解説します。

これらの情報は、実験内で用いられているコンポーネント毎にまとまっています。本章の実験では (教示などの追加前であれば)Polygon コンポーネントが 2 つ (cross, stimulus) と Keyboard コンポーネント 1 つ (key_resp_2) が使用されているので、これらのコンポーネント毎にまとまっているはずです。出力される情報はコンポーネントにより異なります。表 表 3.4 に Polygon コンポーネントが出力する情報を示します。

表 3.4　Polygon コンポーネントの出力 (**[名前]** は cross とする)

列名	概要
cross.started_mean	刺激の開始時刻の平均値を計算したもの。単位は秒。
cross.started_raw	刺激の開始時刻を試行順に並べたもの。単位は秒。列数は繰り返し数に一致し、1 列目にのみ列名が付く。
cross.started_std	刺激の開始時刻の標準偏差を計算したもの。単位は秒。
cross.stopped_mean	刺激の終了時刻の平均値を計算したもの。単位は秒。ただし、刺激の終了時刻が指定されていない場合は列自体が出力されない。
cross.stopped_raw	刺激の終了時刻を試行順に並べたもの。単位は秒。列数は繰り返し数に一致し、1 列目にのみ列名が付く。刺激の終了時刻が指定されていない場合は空白となる。

Polygon コンポーネントからは、刺激の開始・終了時刻の raw データ、つまり統計処理などをする前の「生」のデータと、平均値と標準偏差が出力されています。raw データならともかく、平均値や標準偏差にはあまり意味がないので、利用することは少ないと思います。Polygon コンポーネント以外の刺激提示用のコンポーネントはほぼ同様の情報を出力します。

続いて Keyboard コンポーネントが出力する情報を表 表 3.5 に示します。Polygon コンポーネントの出力と似たようなレイアウトですが、最初に正答率と各試行の正誤に関する情報 (**[正答を記録]** を選択した場合のみ)、平均反応時間、各試行の反応時間、反応時間の標準偏差が出力されています。これらの情報が出力されていることに魅力を感じるなら、trial-by-trial 記録ファイルではなく xlsx 形式 (または CSV 形式の summaries) の出力ファイルを使うとよいでしょう。

表 3.5　Keyboard コンポーネントの出力 ([名前] は key_resp_2 とする)

列名	概要
key_resp_2.corr_mean	正答率を計算したもの。 **[正答を記録]** を選択した場合のみ出力される。
key_resp_2.corr_raw	試行順に正答なら 1、誤答なら 0 を並べたもの。列数は繰り返し数に一致する。1 列目にのみ列名が付く。 **[正答を記録]** を選択した場合のみ出力される。
key_resp_2.keys_raw	押されたキーを試行順に並べたもの。列数は繰り返し数に一致する。1 列目にのみ列名が付く。
key_resp_2.rt_mean	キー押し時刻の平均値を計算したもの。単位は秒。
key_resp_2.rt_raw	キー押し時刻を試行順に並べたもの。列数は繰り返し数に一致する。1 列目にのみ列名が付く。単位は秒。
key_resp_2.rt_std	キー押し時刻の標準偏差を計算したもの。単位は秒。
key_resp_2.started_mean	キー押し検出開始時刻の平均値を計算したもの。単位は秒。
key_resp_2.started_raw	キー押し検出開始時刻を試行順に並べたもの。単位は秒。列数は繰り返し数に一致し、1 列目にのみ列名が付く。キー押し検出開始時刻が指定されていない場合は空白となる。
key_resp_2.started_std	キー押し検出開始時刻の標準偏差を計算したもの。単位は秒。
key_resp_2.stopped_mean	キー押し検出終了時刻の平均値を計算したもの。単位は秒。ただし、キー押し検出終了時刻が指定されていない場合は列自体が出力されない。
key_resp_2.stopped_raw	キー押し検出終了時刻を試行順に並べたもの。単位は秒。列数は繰り返し数に一致し、1 列目にのみ列名が付く。キー押し検出終了時刻が指定されていない場合は空白となる。
key_resp_2.stopped_std	キー押し検出終了時刻の標準偏差を計算したもの。単位は秒。ただし、キー押し検出終了時刻が指定されていない場合は列自体が出力されない。

すべてのコンポーネントについて情報が出力された後には order という見出しがついた列があり、その列を 1 行目と数えて繰り返し数と同じだけの列数の出力があります。これは各条件の 1 試行め、2 試行め…が実験全体の中で何試行めであったかを示しています。最初の試行が 0 番目と出力される点にご注意ください。以上で出力は終了です。

コンポーネントひとつにつき開始時刻・終了時刻の情報だけで繰り返し数× 2 ＋ α の列数が出力されるので、繰り返し数が 100 回以上、コンポーネント数が十数個もあるような実験を作ると軽く 1000 列を越えるファイルが出力されてしまいます。これだけ列数が多いと必要な情報を探すのが大変なので、あまり実用的ではないというのが筆者の考えです。ただ、「3.12.7: コンポーネントの開始・終了時刻をデータファイルに出力しないようにする」で述べた方法で開始・終了時刻を保存しないようにすると、開始・終了時刻に関する情報がごっそりなくなって Keyboard コンポーネントの反応の正誤、反応時間の出力と order の情報だけになります。これならかなり実用的になるかと思います。

最後に「xlsx 形式のデータを保存」と「CSV 形式のデータを保存 (summaries)」の使い分けですが、これは分析に使用するアプリケーションが対応できる形式のものを選ぶとよいでしょう。一般論として、CSV 形式のデータはただのテキストファイルなので数多くのアプリケーションで開くことができますが、条件ファイルのパラメータに日本語が含まれていると正常に表示されない場合があります。xlsx 記録ファイルは開くために Excel や LibreOffice Calc 等が必要ですが、日本語を含んでいても正常に表示できます。各自の都合に合わせ

て選んでください。

3.12.9 独自のルーチンテンプレートを登録する方法 (上級)

ルーチンのテンプレートの実体は通常の psyexp ファイルで、Builder に標準で登録されているテンプレートはインストールディレクトリの app/Resouces/routine_templates/というフォルダに保存されています。ユーザーが独自のテンプレートを追加する場合は、ユーザー設定フォルダに routine_templates というフォルダを作成して、そこに保存します。ユーザー設定フォルダは Windows なら **%APPDATA%/psychopy3/** 、MacOS や Linux なら **~/.psychopy3/** です。

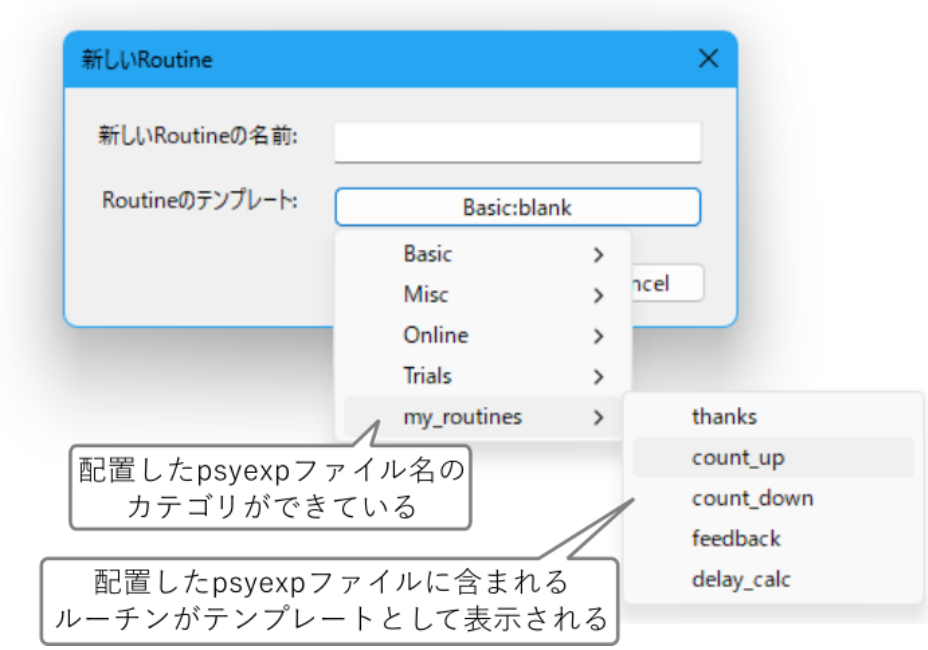


図 3.35　新規ルーチン作成時のテンプレート選択。

図 3.35 に、ユーザー設定フォルダの routine_templates に my_routines.psyexp という名前で psyexp ファイルを配置した例を示します (新たに psyexp ファイルを配置したり編集した場合は PsychoPy を再起動する必要があります)。psyexp ファイルの名前である my_routines というカテゴリが追加されていることがわかります。このカテゴリ内に thanks, count_up, count_down, feedback, delay_calc というテンプレートが表示されていますが、これらはいずれも my_routines.psyexp ファイルに含まれるルーチンです。テンプレートとして使いたいルーチンをひとつの psyexp ファイルにまとめておけば、このようにひとつのカテゴリのテンプレートとして使用できるわけです。以下に注意点を挙げておきます。

- ユーザー設定フォルダの routine_templates フォルダには複数の psyexp ファイルを配置できます。各 psyexp ファイルが追加のカテゴリとして表示されるので、うまく利用するとテンプレートを用途ごとなどに分類して追加できます。
- テンプレートとして作成する psyexp ファイルにおいて、ルーチンはフローに追加しなくても構いません。また、実験として実行できない状態 (設定が必要なパラメータが空白のままになっているなど) でも構いません。
- テンプレート内で画像ファイルなどの外部ファイルを使用している場合、そのファイル名の設定はテンプレートとして使用した際に残っていますが、外部ファイルは自動的にコピーされません。具体例を挙げると、background.png というファイルを背景画像として使用している場合、background.png を

routine_templates フォルダに保存しておいても、作成中の実験に自動的に background.png はコピーされません。

3.12.10 staircase と interleaved staircases による階段法の手続き (上級)

[Loop の種類] の staircase と interleaved staircases はいずれも階段法の手続きを Builder で実現するために用意されたものです。階段法は心理物理学的測定法の一つである極限法の効率を高めた方法です。参加者の反応は「正反応」(刺激が知覚できている) と、「負反応」(刺激が知覚できていない) の二種類に分類できるとします。極限法と同様に、計測したい閾値から離れたところから刺激を提示し始めて、参加者の反応が変化するまで一定方向に刺激を変化させ続けます。極限法ならば最初に参加者の判断が変化したところで系列を終了しますが、staircase 法では今までと反対方向へ刺激を変化させて続行します。その後は正反応、負反応が一定数連続する度に刺激を変化させる方向を反転させます。

図 3.36 は、「正反応が 3 回連続する」または「負反応が 1 回出現する」時に反転させるというルールで階段法を実施した例を示しています。閾値付近で何度も刺激の変化方向が反転して、閾値付近での参加者の判断を効率的に記録できることがわかります。

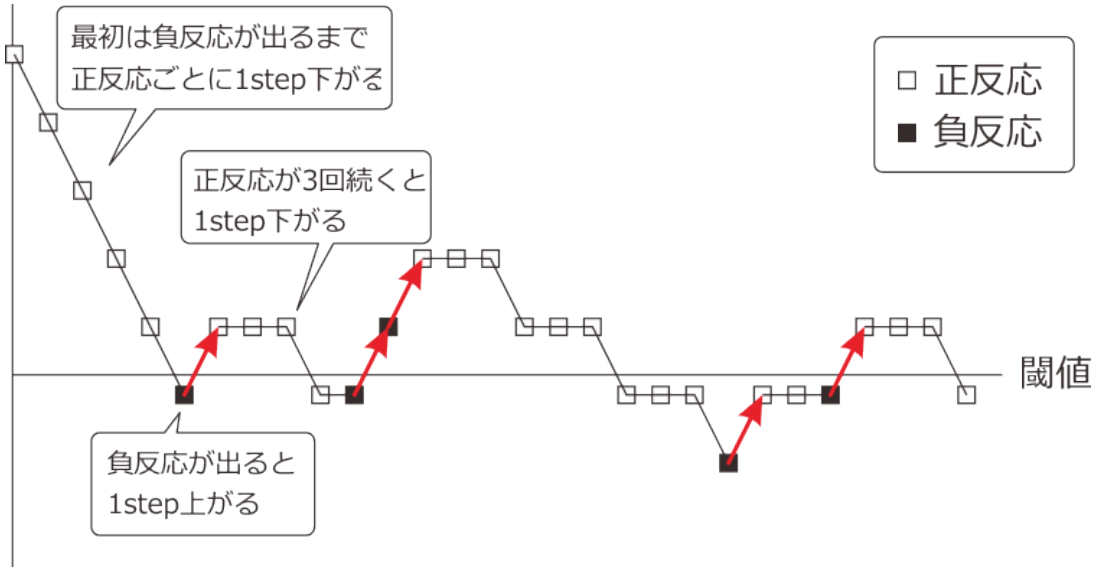


図 3.36　階段法の実行例

以上の例を踏まえて、staircase ループについて説明します。**[Loop の種類]** を staircase に変更すると、**[条件]** などのパラメータが消えて代わりに **[初期値 $]**、**[最大値 $]**、**[最小値 $]**、**[ステップサイズ $]**、**[ステップタイプ]**　、**[誤反応数 $]**、**[正反応数 $]**、**[反転回数 $]** というパラメータが出現します。以上のパラメータのうち、まず試行開始時の初期値と反転のルールを決定するパラメータを 表 3.6 に示します。この表で「刺激レベル」と書かれているのは 図 3.36 の縦軸にあたる値で、Builder 内では level という内部変数で参照することができます。例えば、staircase ループで Polygon コンポーネントの塗りつぶし色 (RGB 値) の明るさを変化させたい場合は、**[塗りつぶしの色]** を level, level, level とします。

表 3.6　初期値と反転のルールを決定するパラメータ

[初期値 $]	開始時の刺激レベルの値を指定します。浮動小数点数でなければいけません。
[誤反応数 $]	正の整数を指定します。負反応がここに定められた回数連続すると刺激レベルが減少します。
[正反応数 $]	正の整数を指定します。正反応がここに定められた回数連続すると刺激レベルが増加します。
[最小値 $]	刺激レベルの最小値を浮動小数点数で指定します。例えば刺激の RGB 値を変化させる場合、値が-1.0 より小さくなると正常に表示されませんが、そのようなときにこのパラメータで下限の値を定めます。
[最大値 $]	刺激レベルの最大値を浮動小数点数で指定します。例えば刺激の RGB 値を変化させる場合、値が 1.0 より小さくなると正常に表示されませんが、そのようなときにこのパラメータで下限の値を定めます。

刺激を変化させる量を決めるには、 表 3.7 のパラメータを使用します。

表 3.7　刺激の変化量を決定するパラメータ

[ステップサイズ $]	刺激レベルの変化量を浮動小数点数で指定します。リストが指定された場合、最初はリストの 0 番目の要素の値が用いられ、変化方向が反転する度にリストの次の要素の値が新たな変化量となります。
[ステップタイプ]	lin、log、db のいずれかを選びます。

[ステップタイプ] は、 **[ステップサイズ $]** と組み合わせて使用します。

lin

[ステップサイズ $] に入力された値がそのまま現在の刺激レベルに加減算されます。

log

現在の刺激レベルが L、ステップサイズが S とします。刺激レベルを増加させるときには次の刺激レベル値を $L \times 10.0^{S}$ 、減少させるときには $L \div 10.0^{S}$ とします。

db

現在の刺激レベルが L、ステップサイズが S とします。刺激レベルを増加させるときには次の刺激レベル値を $L \times 10.0^{S/20.0}$ 、減少させるときには $L \div 10.0^{S/20.0}$ とします。

一連の手続きが終了する条件を決めるには、 表 3.8 のパラメータを使用します。

表 3.8　staircase 終了を決定するパラメータ

[反転回数 $]	終了までに最低限必要な反転回数を自然数で指定します。ただし、**[ステップサイズ $]** にリストが入力されていて、そのリストの長さが **[反転回数 $]** より長い場合は **[反転回数 $]** のリストの長さの回数反転するまで終了しません。
[繰り返し回数 $]	終了までに最低限必要な試行数 (参加者の判断回数) を指定します。この回数を過ぎても、 **[反転回数 $]** または **[ステップサイズ $]** によって定められた回数反転していなければ終了しません。

なお、以下の設定をすると、刺激レベルを 1.0 から 0.1 ずつ減少させていく通常の極限法 (下降系列) のようにも使えます。ただし、最初の刺激レベルがいきなり負反応の場合はうまくいきませんので十分に高い **[初期値 $]** を設定する必要があります。

- **[初期値 $]** = 1
- **[ステップサイズ $]** = 0.1
- **[ステップタイプ]** = lin
- **[誤反応数 $]** = 1
- **[正反応数 $]** = 1
- **[反転回数 $]** = 1
- **[繰り返し回数 $]** = 0

続いて interleaved staircases ですが、これは複数の階段法のパラメータを条件ファイルから読み込んで実行するものです。条件ファイルは **[条件]** に指定します。各 staircase を実行する順番は **[切り替え方法]** から random、sequential、fullRandom の中から選びます。これは **[Loop の種類]** の random、sequential、fullRandom と同じ意味です。 **[繰り返し回数 $]** には各 staircase での最小試行数を指定します。

[ステアの種類] では、各 staircase の刺激レベル変更方式を simple, QUEST, questplus のいずれかから選択することができます。simple は staircase と同じです。QUEST はいくつかの仮定の下で効率的に閾値を探索するアルゴリズムです。詳しくは Watson & Pelli (1983). QUEST: A Bayesian adaptive psychometric method. *Perception & Psychophysics*, 33(2), 113-120 などを参照にしてください。questplus は Watson (2017). QUEST+: A general multidimensional Bayesian adaptive psychometric method. Journal of Vision, 17(3):10. doi: 10.1167/17.3.10. で提案された QUEST+ アルゴリズムに対応するものですが、ここでは解説を省略します。

interleaved staircases の条件ファイルの例を 図 3.37 に示します。条件ファイルは必ず以下の列を含んでいなければいけません。

label

データの出力時に各 staircase を区別するために用いられるラベル。すべての行で異なっている必要があります。

startVal 各

staircase 開始時の刺激レベルの値です。

startValSd (QUEST のみ) 閾

値の推定値の標準偏差の初期値です。

上記以外の列の値は、Builder の変数として読み込まれます。例えば 図 3.37 には sf という列がありますが、この列の値を Grating コンポーネントの **[空間周波数 $]** に指定したい場合は、単に **[空間周波数 $]** に sf と書けばよいということです。 表 3.9 に示した名前を持つ列は、staircase のパラメータとして利用されます。各パラメータの意味は PsychoPy 公式サイトの API をご覧ください (StairHandler および QuestHandler)。

	A	B	C	D	E	F	G
1	label	startVal	sf	stepSizes	maxVal	minVal	
2	low_2	0.001	2	[4,2,2,1]	1	0	
3	high_2	0.1	2	[4,2,2,1]	1	0	
4	low_8	0.001	8	[4,2,2,1]	1	0	
5	high_8	0.1	8	[4,2,2,1]	1	0	
6							
7							
8							

図 3.37　interleaved staircase ループのプロパティ設定ダイアログ

表 3.9　staircase のパラメータとして解釈される変数名

simple の場合	nReversals, stepSizes, nTrials, nUp, nDown, extraInfo, stepType, minVal, maxVal
QUEST の場合	pThreshold, nTrials, stopInterval, method, stepType, beta, delta, gamma, grain, range, extraInfo, minVal, maxVal, staircase

第 4 章

繰り返し方法を工夫しよう―傾きの対比と同化

4.1 この章の実験の概要

この章では、傾きの対比の実験を題材として、実験を実施する度に条件ファイルを変更したり、繰り返しを多重に用いることによってやや複雑な計画の実験を実現したりする方法を学びます。

さっそく実験の内容を確認しましょう。 図 4.1 に縞模様の刺激が描かれていますが、このように正弦波上に明るさが変化する縞模様の刺激をグレーティングと呼びます。今回の実験では、大小 2 個の円形のグレーティング刺激を実験に使用します。スクリーン中央に 図 4.1 のように二つの刺激の中心が一致するように重ねます。グレーティングの模様がスクリーンの垂直方向にぴったり一致しているのを 0 度の方向として、大きいグレーティング刺激の向きを反時計回り、または時計回りに 20 度傾けます。そして小さいグレーティング刺激の向きを 0 度 ±5 度の範囲で変化させて、小さいグレーティング刺激が 0 度より反時計回り、時計回りのどちらに傾いているかを実験参加者に判断させます。「小さい方のグレーティング刺激」などといちいち書くのは面倒ですので、小さい方のグレーティング刺激を「テスト刺激」、大きい方のグレーティング刺激を「コンテクスト刺激」と呼ぶことにしましょう。コンテクスト刺激がテスト刺激の傾きの知覚にどのような影響を与えるかを調べるのが実験の目的です。

実験の作成を始める前に刺激の大きさや反応方法などの詳細を決める必要がありますが、それ以前にこの刺激は前章までに学んだ知識で作成できそうにありません。PsychoPy Builder にはこの刺激を描くためにぴったりな Grating コンポーネントというコンポーネントが用意されているので、まずはその使い方を学びましょう。

4.2 Grating コンポーネント

Grating コンポーネントは、簡単に言うと縞模様の視覚刺激を描くためのコンポーネントです。視知覚の研究においては基本刺激とでも言うべき重要な刺激ですし、視覚認知の研究でもまた頻繁に用いられます。 **[開始]** や **[終了]** で刺激の有効な時間を指定したり、 **[位置 [x, y] $]** で位置を指定したりするなど、大半のプロパティは Polygon コンポーネントや Text コンポーネントと共通です。Grating コンポーネントが Polygon および Text コンポーネントと大きく異なるのは「テクスチャ」タブです。本節ではこのタブの項目について解説します (**[テクスチャ]** は複雑なので「 4.11.1:Grating コンポーネントの **[テクスチャ]** プロパティについて 」で詳しく取り上げます)。

図 4.2 は **[マスク]** 、 **[位相 (周期に対する比) $]** 、 **[空間周波数 $]** を変化させるとどのような刺激が提示される

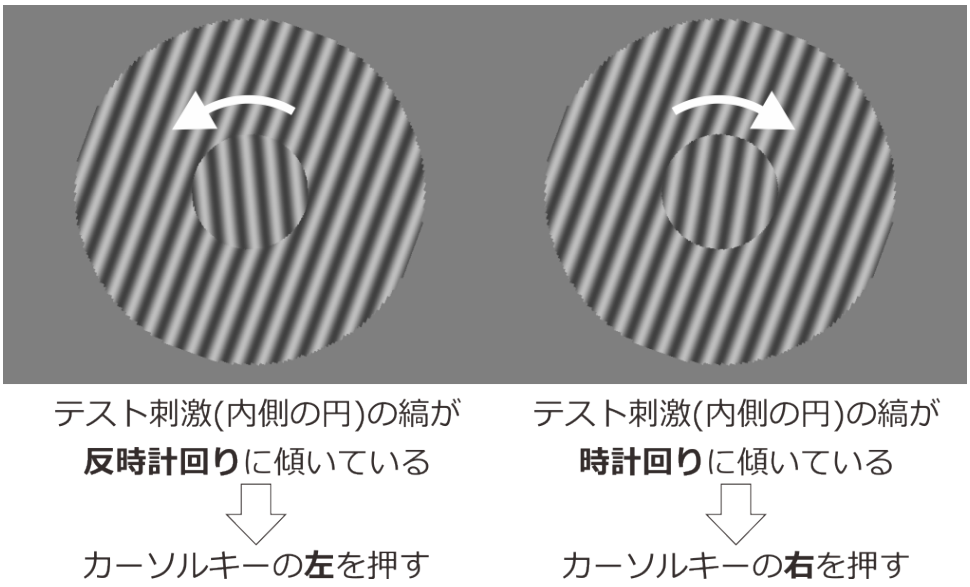


図 4.1　この章の実験。実験参加者は、テスト刺激の縞が垂直方向より反時計回りに傾いているか、時計回りに傾いているかを判断します。テスト刺激の周囲のコンテクスト刺激が判断に与える影響を明らかにするのが目的です。

かを示したものです。**[マスク]** は刺激を切り抜くプロパティで、None、circle、gauss を指定することができます。空白にしておくと長方形の刺激が描画され、circle にすると楕円状に切り抜かれた刺激が描画されます。gauss は 2 次元 Gauss 関数の値に従ってコントラストが変調された縞模様を描きます。Gauss 関数と言われてもピンと来ない方は正規分布の密度関数を思い出してください。正規分布の密度関数は Gauss 関数の一種であり、「Gauss 関数の値に従ってコントラストが変調される」とはあの正規分布の密度関数のように中心から周辺に向かってなだらかに縞模様の薄れていくということです。正弦波を正規分布で変調した刺激は Gabor パッチと呼ばれ、視知覚の実験では非常によく用いられます。

続いて **[空間周波数 $]** ですが、この値を大きくすると縞模様の密度が高くなります。より厳密な表現を用いれば、空間周波数 (spatial frequency) とは一定の空間範囲に描かれる模様の繰り返し回数のことです。**[空間の単位]** が cm と deg の場合には、刺激の幅 1.0 に含まれる縞模様の数に対応します。例えば刺激の幅が 5.0cm で **[空間周波数 $]** が 0.4 ならば 5.0 × 0.4=2.0、つまり刺激には 2 周期分の縞模様が描かれます (図 4.2 中段左)。**[空間の単位]** が norm と height の場合は、刺激に描かれる縞模様の数に直接対応します。つまり、**[空間周波数 $]** が 2.0 なら刺激の幅の値がいくらであろうと常に 2 周期分の縞が描かれます。**[空間の単位]** が pix の場合は複雑で、「**[空間周波数 $]** の値×刺激の幅」の周期分の縞が描かれます。刺激の幅が 200pix であれば、**[空間周波数 $]** に 0.01 を指定すれば 200 × 0.01 で 2 周期分の縞になります。

[位相 (周期に対する比) $] は、縞模様の位相を指定するパラメータです。Grating コンポーネントは初期設定では刺激の中心で明るさが最大になるように縞模様が描かれますが、位相を指定すると明るさが最大になる位置をずらすことができます。単位は 1 周期に対する比であり、0.5 ずらすとちょうど明暗が反転します (図 4.2 下段の左端と右端)。

[テクスチャの解像度 $] と **[補間]** の効果を示したのが 図 4.3 です。グレーティングコンポーネントを大きく拡大した図が描かれていますが、まず左側の二つをご覧ください。上が **[テクスチャの解像度 $]** が初期値の 128、下が 512 の時の結果です。512 の時の結果と比べると、128 の時には刺激の境界も縞模様もぼけています。この「ぼけ」は、PsychoPy がグレーティングを描くときには内部で縞模様の画像データ (テクスチャ) を作成し、それを **[サイズ [w, h] $]** で指定された大きさに拡大するために生じます。**[テクスチャの解像度 $]** は

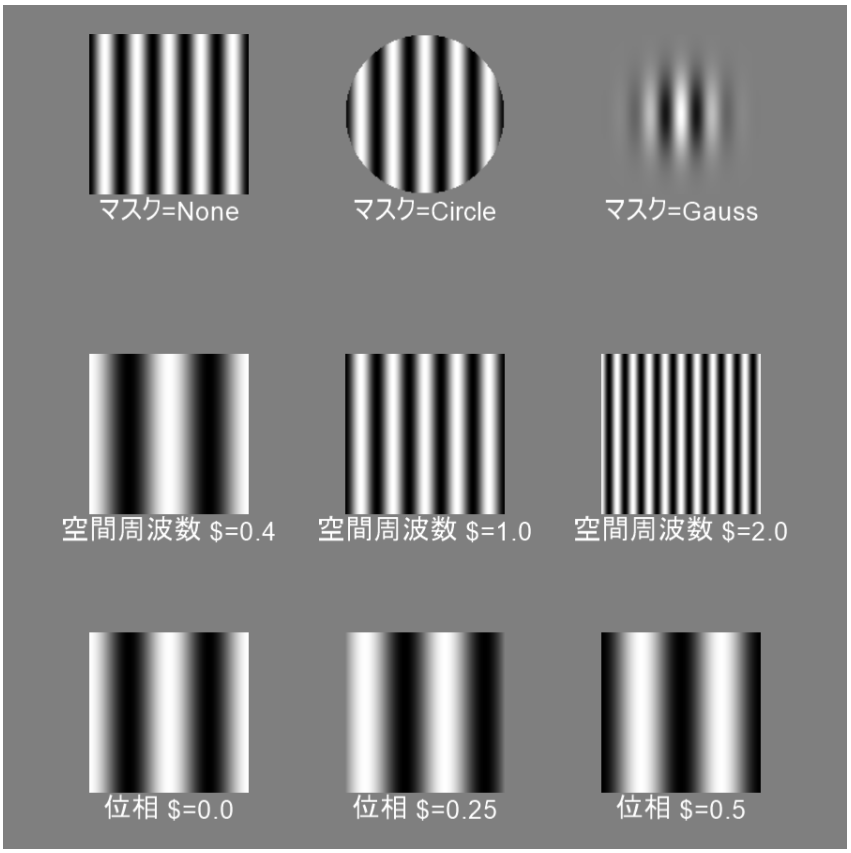


図 4.2　Grating コンポーネントのプロパティのうち、**[マスク]** (上段)、**[空間周波数 $]** (中段)、**[位相 (周期に対する比) $]** を変化させた例。いずれも **[サイズ [w, h] $]** は [5.0,5.0] で、**[単位]** は cm を指定しています。

テクスチャの解像度を指定するプロパティで、128 であれば 128 × 128 ピクセル、512 であれば 512 × 512 ピクセルのテクスチャを生成します。値が高い方が大きく拡大した時のぼけが小さく済みますが、その代わりに PC のグラフィック描画機能に負担をかけます。小さなグレーティング刺激を多数描画する場合は、描画の負担を小さくするために **[テクスチャの解像度 $]** を低く設定するべきです。逆に今回の実験のように大きなグレーティング刺激をせいぜい 2、3 個描画する場合は高い値を設定するとよいでしょう。描画負担は使用している PC のグラフィック性能に大きく依存しますので、グラフィック性能が高い PC であれば 512 に設定して多数のグレーティングを描いても問題は生じません。

[補間] は、グレーティングを拡大した時の補間方法を指定します。補間方法と言われてもピンと来ないかもしれませんが、 図 4.3 右の二つの拡大図を比べてください。上は刺激の境界がぼやけていますが、下は境界がくっきりしていてカクカクしています。上が **[補間]** に「一次」を指定した例で、拡大した時に足りない色情報を周囲のピクセルと滑らかにつながるように補います。結果として、このように大きく拡大した場合はぼけが目立ってしまいます。一方「最近傍」を指定した場合は、元のテクスチャで最も近いピクセルの色を使用しますので、ぼけが生じない代わりにカクカクになってしまいます。滑らかにする方法は使用する PC のグラフィック機能によって異なりますので、自分が使用する PC でどちらの方がよい出力が得られるか各自で確認してください。

なお、「外観」タブの **[前景色]** はすでに Text コンポーネントで解説しましたが、Grating コンポーネントの場合は縞模様を描画しますので少々複雑です。Grating コンポーネントの色は、**[前景色]** の値を **[テクスチャ]** で指定された波形に掛け算することによって決まります。つまり、**[テクスチャ]** が sin で **[前景色]** の値が 1, 1,

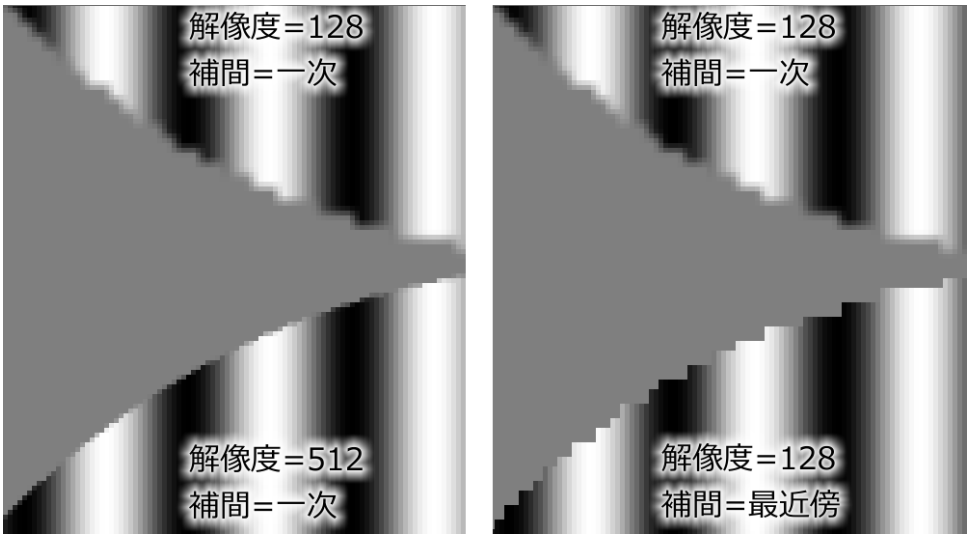


図 4.3　Grating コンポーネントの **[テクスチャの解像度 $]** および **[補間]** オプションの効果。

1 であれば-1 から +1 まで変化します。 **[前景色]** が 0.5, 0.5, 0.5 であれば-0.5 から +0.5 まで変化します。-0.5, -0.5, -0.5 であれば、0.5, 0.5, 0.5 のときと同様に-0.5 から +0.5 まで変化しますが、負の値を掛け算していますので明暗が 0.5, 0.5, 0.5 の時と反転します。 1, 0, 0 のように RGB 各成分の値が異なる場合も、それぞれの成分に波形が掛け算されます。1, 0, 0 の場合は-1, 0, 0 から 1, 0, 0 まで変化するということです。 **[前景色]** に red や green といった色名が指定され場合は、PsychoPy の内部でこれらの色名を RGB に変換したうえで波形との掛け算が行われます。

ずいぶん脱線が長くなってしまいました。以上の解説を踏まえて、実験手続きの詳細を決定して実験を作成しましょう。

チェックリスト

- Grating コンポーネントを用いて長方形、または楕円形に縞模様が描かれた刺激を提示することができる。
- Grating コンポーネントで **[空間の単位]** が cm、deg、pix、norm、height のいずれの場合においても、「幅 x に対して y 周期分の縞模様」を描けるように **[空間周波数 $]** の値を決定できる (x、y は正の数値)。
- Grating コンポーネントで描かれる縞模様を初期設定の状態からずらして描画することができる。
- Grating コンポーネントで描画処理の負担を軽くするためにテクスチャの解像度を下げることができる。
- Grating コンポーネントを大きく表示するときに画質を高めるためにテクスチャの解像度を上げることができる。
- Grating コンポーネントの色を指定したときに、何色の縞模様が描かれるのかをこたえることができる。

4.3 パラメータを決定しよう

それでは実験に用いる刺激のパラメータを決定しましょう。まず、テスト刺激とコンテクスト刺激の大きさと縞模様を 図 4.4 のように設定することにします。 コンテクスト刺激の **[サイズ [w, h] $]** と **[空間周波数 $]** はそれぞれテスト刺激の 3 倍の値が設定されていて、同じ幅の縞が描かれるようにしてあります。いずれの刺激も **[マスク]** に circle を指定して円形にします。 **[前景色]** はいずれも 0.5, 0.5, 0.5 にしておきましょう。

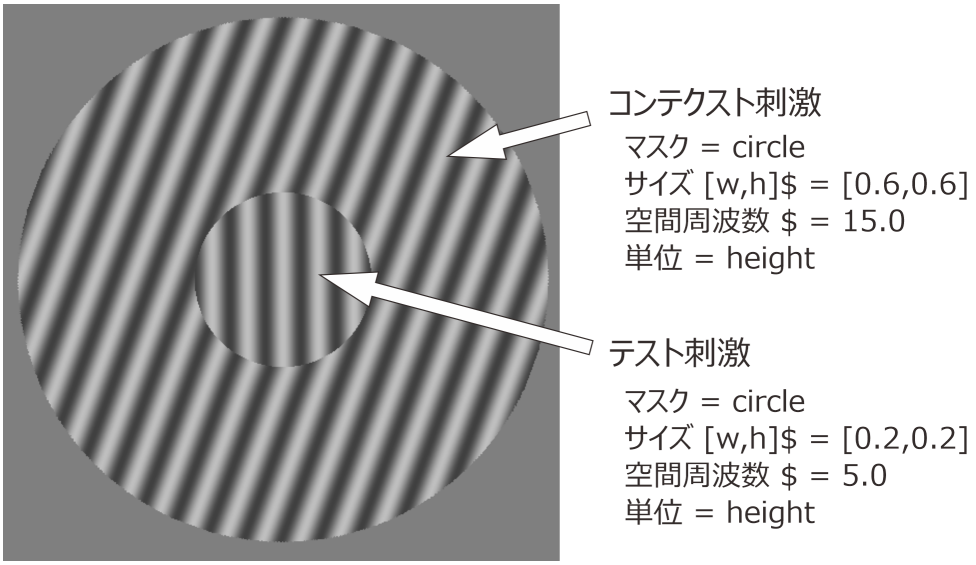


図 4.4　刺激のパラメータ

今回の実験で一番重要なパラメータはテスト刺激とコンテクスト刺激の方向です。すでに概要で述べたとおり、コンテクスト刺激には反時計回りに 20 度 (**[回転角度 $]** =20) 傾いたものと、時計回りに 20 度 **[回転角度 $]** =-20 傾いたものの 2 種類を用います。それぞれのコンテクスト刺激に対して、-5 度から 1 度間隔で 5 度までの計 11 種類のテスト刺激を組み合わせることにしましょう。それぞれの組み合わせに対して 5 回ずつ、無作為な順序に実験参加者に提示して、判断させることにします。試行数はコンテクスト刺激 2 種類×テスト刺激 11 種類×繰り返し 5 回=110 試行です。

続いて 図 4.5 に実験の手続きを示します。まず実験が始まったら、反応方法の教示をスクリーンに提示します。反応方法は、テスト刺激が時計回りに傾いているように見えたらカーソルキーの右、反時計回りに傾いているように見えたらカーソルキーの左を押すものとします。教示画面は各自で自由に作成していただいてよいと思いますが、刺激の例を示しながら反応するキーを示すとよいと思います。

教示画面でカーソルキーの左右いずれかを押すと実験が始まります。各試行はまず 0.5 秒間の空白のスクリーンから始まり、続けて刺激を提示します。実験参加者がカーソルキーの左右いずれかを押して反応するまで刺激を提示し続け、キーが押されたら直ちに次の試行に進みます。全試行終了すれば実験は終了です。 図 4.5 では示していませんが、最後に「実験は終了しました」などのメッセージを表示するのも良いでしょう。

以上が手続きです。いよいよ実験を作成しますが、第 3 章 の実験と比べて使用するコンポーネントが多いので便利なテクニックを紹介しながら作業を進めましょう。

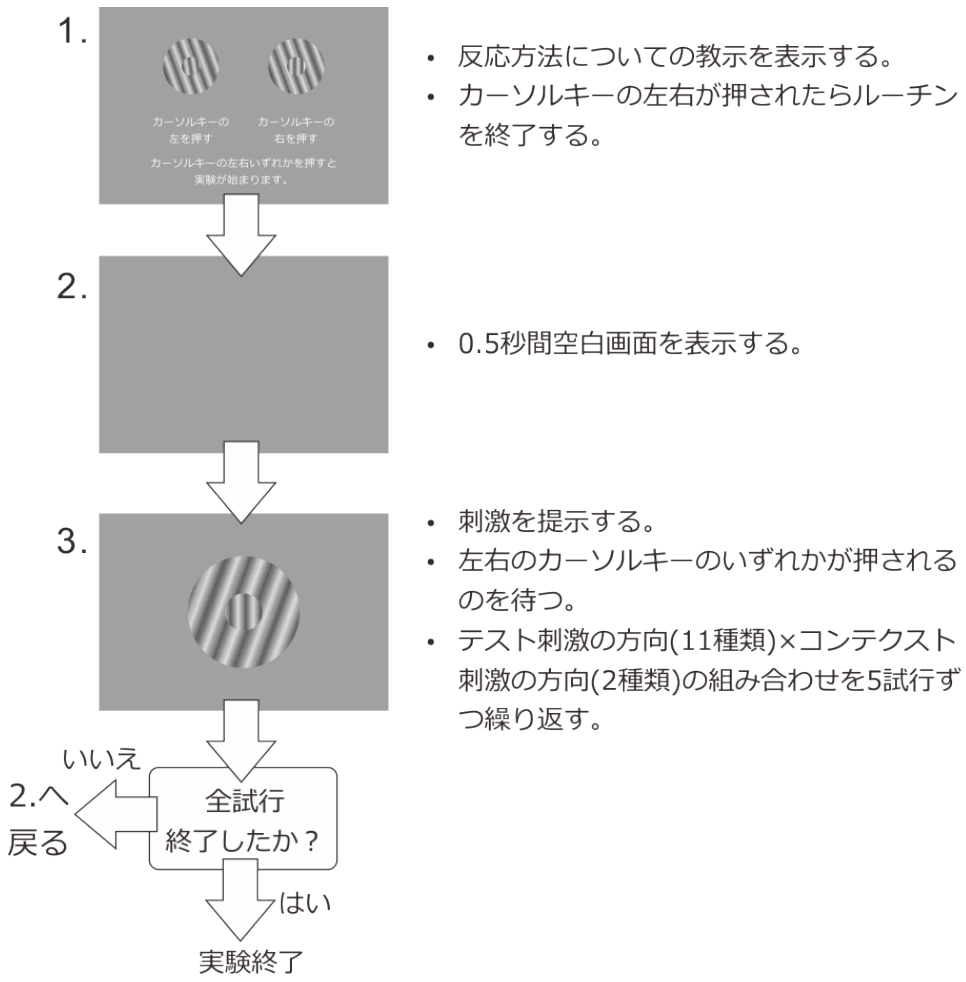


図 4.5　実験の手続き

4.4 コンポーネントのコピーを活用して実験を作成しよう

では、Builder を開いて作業を開始します。実験は exp04.psyexp というファイル名で保存するものとします。

準備作業

- この章の実験のためのフォルダを作成して、その中に exp04.psyexp という名前で新しい実験を保存する。
- 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。

準備ができたら、まず trial ルーチンで以下の作業をしてください。

trial ルーチン

–Grating コンポーネントを 1 つ配置し、以下の設定をする。

* 「基本」タブの **[名前]** を `testStim` とし、 **[開始]** を「時刻 (秒)」で `0.5` に、 **[終了]** を空白にする
* 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.2,0.2)` にする。 **[回転角度 $]** に `testDir` と入力して、「繰り返し毎に更新」にする。

* 「外観」タブの **[前景色]** を `0.5, 0.5, 0.5` にする。
* 「テクスチャ」タブの **[テクスチャ]** が `sin` であることを確認する。 **[マスク]** を `circle` に、 **[空間周波数 $]** を `5.0` にする。 **[テクスチャの解像度 $]** を 512 にする。

これでテスト刺激が完成しました。続いてコンテキスト刺激を作成しなければいけませんが、 **[名前]** と **[サイズ [w, h] $]** 、 **[回転角度 $]** 以外はテスト刺激と設定が同じです。このような場合には、コンポーネントのコピー機能を使用すると作業が楽になります。

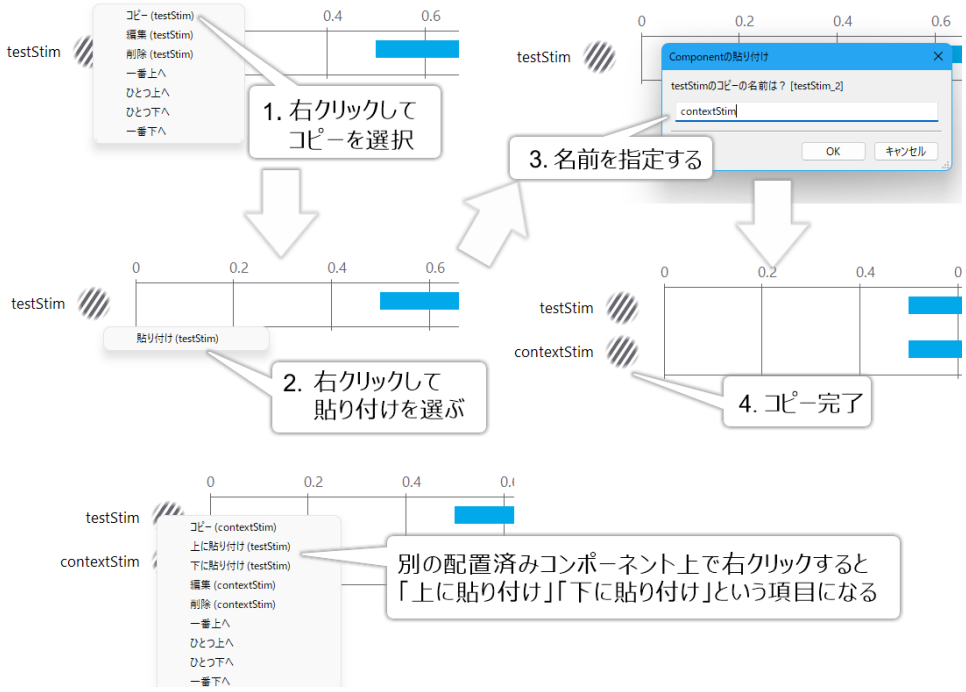


図 4.6　コンポーネントのコピー

図 4.6 にコピーの手順を示します。まず図の左上のように、コピーしたいコンポーネントにマウスカーソルを重ねて右クリックし、メニューから「コピー」を選択します。その後、コピーを作成したいルーチンを表示します。今は trial ルーチンに作成したいので、そのままで結構です。ルーチンの余白部分でマウスを右クリックすると、「貼り付け (testStim)」というメニューが表示されます (図の中段左)。選択すると図の右上のようにコンポーネントの名前をたずねるダイアログが表示されます。Builder では同一の名前を持つコンポーネントを複数作ることができませんので、新しい名前をつけなければいけません。名前を空欄のまま OK をクリックすると testStim_2 のように元のコンポーネントの名前の後ろに数字を添えた名前となりますが、ここは contextStim という名前にしておきましょう。新しい名前を入力して OK をクリックすれば、図の中段右のようにコンポーネントのコピーが完了します。

なお、「貼り付け」のメニューを表示するために右クリックをする際に、配置済みの他のコンポーネント上で右クリックすると、図の下段のように削除や順番変更といった項目の中に「上に貼り付け ()」「下に貼り付け ()」という項目が表示されます (括弧内はコピー中のコンポーネント名)。これを利用すると、たくさんのコンポーネントを配置しているルーチンで、他のコンポーネントの順番を考慮して狙った位置に貼り付けることができます。複雑な実験を作るときにはとても便利なテクニックなので覚えておきましょう。

コンポーネントのコピーが終了したら、contextStim のパラメータを変更します。

trial ルーチン

–contextStim の設定を以下のように変更する。

* 「レイアウト」タブの **[サイズ [w, h] $]** を (`0.6`, `0.6`) に、 **[回転角度 $]** を contextDir にする。
* 「テクスチャ」タブの **[空間周波数 $]** を `15.0` にする。
* contextStim の上に testStim が表示されるようにコンポーネントの順番を並び替える。

これで刺激は完成です。あとは参加者の反応を計測するために Keyboard コンポーネントを置いておきましょう。

trial ルーチン

–Keyboard コンポーネントをひとつ配置し、以下のように設定する。

* 「基本」タブの **[開始]** を「時刻 (秒)」で `0.5` に、 **[終了]** を空白にする。
* 「データ」タブの **[検出するキー $]** を `'left', 'right'` にする。 **[正答を記録]** をチェックして、 **[正答]** に `$correctAns` と入力する。

続いて実験の開始時に表示する教示画面を作成します。 図 4.1 のように時計回り、反時計回りの刺激の実例を表示するとわかりやすいでしょう。 図 4.1 のような画面を作成するとなると、Grating コンポーネントを 4 個配置しなければいけません。たった今覚えたコンポーネントのコピーを使えば楽ができますが、前章で紹介したルーチンのコピーを使うとさらに手順を省略できます。 図 4.7 のように trial ルーチンを instruction という名前でコピーしましょう。

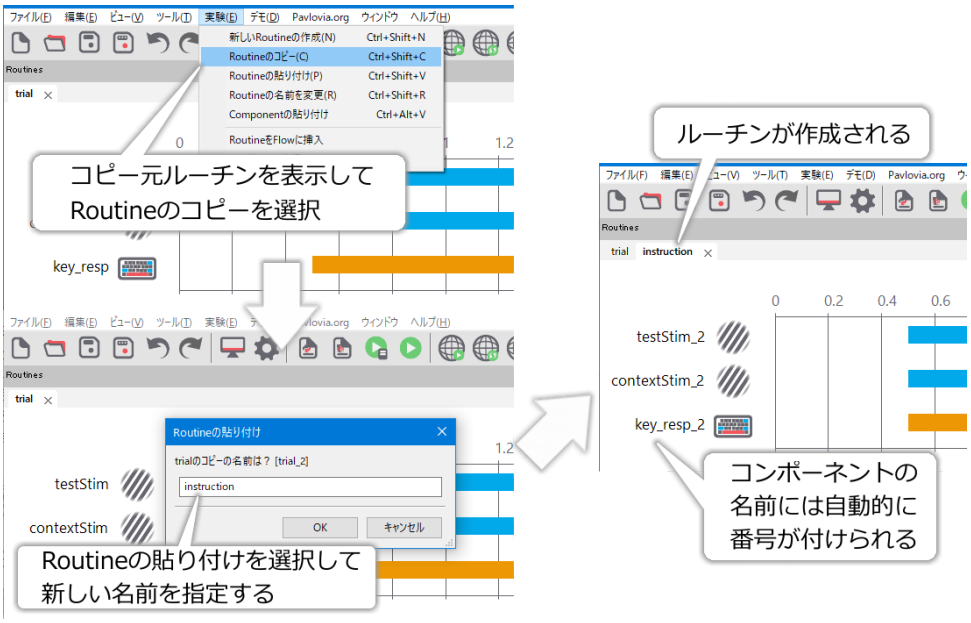


図 4.7　ルーチンのコピー。

コピー元の trials ルーチンと同じコンポーネントが含まれていること、コンポーネントの名前には (名前が重複しないように) 番号が自動的につけられていることを確認してください。これでかなり作業を楽することができました。後は instruction ルーチンで以下のように作業してください。

- instruction ルーチン

- **フローの先頭に挿入する**（忘れがちなので注意！）。
- contextStim_2 の **[名前]** を `left_large` とし、**[開始]** を「時刻 (秒)」で `0.0` にする。**[位置 [x, y] $]** を `(-0.4, 0)` とする。**[回転角度 $]** を `20.0` にし、「更新しない」にする。
- testStim_2 の **[名前]** を `left_small` とし、**[開始]** を「時刻 (秒)」で `0.0` にする。**[位置 [x, y] $]** を `(-0.4, 0)` とする。**[回転角度 $]** を `-10.0` にし、「更新しない」にする。
- left_large をコピーして **[名前]** を `right_large` にする。**[位置 [x, y] $]** を `(0.4, 0)` とする。
- left_small をコピーして **[名前]** を `right_small` にする。**[位置 [x, y] $]** を `(0.4, 0)`、**[回転角度 $]** を `10.0` にする。
- left_large の上に left_small、right_large の上に right_small が描かれるように各コンポーネントが並んでいることを確認する。

これで刺激は完成です。次に進む前に、便利な機能をひとつ覚えておきましょう。Builder のメニューの「実験」から「実験内を検索...」を選択してください。「実験内を検索...」というタイトルのダイアログが開くので、ダイアログの一番上の虫眼鏡のアイコンと「検索」と書かれている入力欄に (0.6, 0.6) と入力して Enter キーを押してみてください。ここまで解説通りに作業してきたなら、図 4.8 右上のように入力欄の下に何か表示されたはずです。この欄をよく見ると、上に Component, パラメータ, 値と見出しがあります。これは (0.6, 0.6) という値が設定されているパラメータを実験内から探し出して、該当するコンポーネントの名前とパラメータ名をリストアップしてくれているのです。(0.6, 0.6) はグレーティング刺激の大きい方の **[サイズ [w, h] $]** に指定した値なので、instruction ルーチンの left_large と right_large、trial ルーチンの contextStim が検出されています。図 4.8 右下は 0.4 を検索した例です。0.4 は instruction ルーチンに配置した刺激の X 座標で、左側に配置したものは-0.4、右側に配置したものは 0.4 なのでした。検索の結果、正しく left_large, left_small, right_large, right_small の 4 つが検出されていて、left とついているものは-0.4、right とついているものは 0.4 になっていることが確認できます。修正する必要がある場合は、検索結果の項目をクリックするとそのパラメータの設定ダイアログを直接開くことができます。

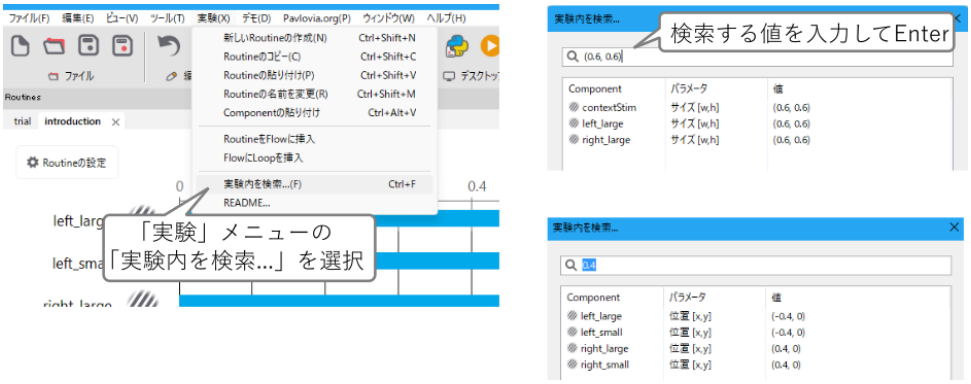


図 4.8　各コンポーネントのパラメータに設定した値を検索し、その値を持つコンポーネントとパラメータの一覧を得ることができます。検出された項目をクリックすると、該当するパラメータの設定ダイアログを直接開くこともできます (検索結果のダイアログは閉じてしまいます)。

複雑な視覚刺激を描画する場合や、多数の刺激を描画する場合は、複数の視覚刺激コンポーネントを適切に組み合わせることが不可欠です。しかし、パラメータの編集はひとつひとつ順番にしかできないので、うっかりひとつ設定し忘れたり間違った値を入力したりといったミスがよく起きます。入力内容に間違いがないかを確認したいとき、もしくは間違えていると思うのだけどもどれを間違えたのかわからないときなどに、この「実

験内を検索...」をうまく使うと作業が楽になります。ひとつ残念なのは、入力内容が正確に一致しないと検出されないことです。例えば 図 4.8 右上の例は [0.6, 0.6] と書いてもサイズの指定として有効ですが、(0.6, 0.6) と検索すると見つかりません。まあこれは仕方がないと思うのですが、(0.6,0.6) でも見つかりません (カンマと次の 0 の間にスペースが無い)。他にも 0 と 0.0 なども値としては同じですが検索では区別されます。普段から「括弧は ( ) を使う」、「カンマの後ろには半角スペースをひとつ入れる」、「0 は 0.0 と書く」など自分なりのルールを決めておくのが理想ですが、0.6 とだけ入力すれば [0.6, 0.6] も (0.6, 0.6) も (0.6,0.6) もすべて検出されますから、工夫次第で対応できるでしょう。

検索機能についての解説はこのくらいにして、実験の作成に戻りましょう。次は instruction にテキストを追加します。

- instruction ルーチン

  Text コンポーネントを 1 個配置して、 [名前] を `TextInst` とする。

  - **[文字列]** に **カーソルキーの左右いずれかを押すと始まります** 等のメッセージを入力する。画面の大きさに応じて文を短くするなり改行するなりするとよい。
  - **[文字の高さ $]** を 0.04 にする (各自の PC 画面の大きさに応じて調節すると良い)。
  - **[位置 [x, y] $]** に `(0, -0.4)` と入力する。

  TextInst をコピーして textCC の名前で貼り付ける。

  - TextCC の **[文字列]** に **反時計回り** と入力し、 **[位置 [x, y] $]** を `(-0.4, -0.35)` にする。

  TextInst をコピーして textC の名前で貼り付ける。

  - TextC の **[文字列]** に **時計回り** と入力し、 **[位置 [x, y] $]** を `(0.4, -0.35)` にする。

最後に Keyboard コンポーネントの調整をします。

- instruction ルーチン
  - 配置済みの Keyboard コンポーネント (標準では key_resp_2 という名前になっているはず) の **[開始]** を「時刻 (秒)」で `0.0` にする。 **[記録]** を「なし」にして、 **[正答を記録]** のチェックを外す。

あとは条件ファイルを作成して、ループを挿入しましょう。

- exp04_20.xlsx (条件ファイル)
  - testDir、contextDir、correctAns の 3 パラメータを設定する。
  - 実験手続きの説明を満たすように testDir に 11 種類の、contextDir に 2 種類の値を入力する。
  - すべての行の correctAns に right を入力する。
  - psyexp ファイルと同じフォルダに保存する。
- trials ループ (作成する)
  - trial ルーチンのみを繰り返すように挿入する。
  - **[繰り返し回数 $]** が 5(初期値) になっていることを確認する。

– **[条件]** の欄の右側のファイル選択ボタンをクリックして exp04_20.xlsx を選択する。

さて、これで各ブロックの手続きは完成です。条件ファイルの 3 つめの項目「すべての行の correctAns に right を入力する」に注目してください。テスト刺激が時計回りに傾いている時にカーソルキーの右を押すのですから、testDir が正の値の時のみ right が正答のはずです。しかし、この実験では敢えてすべての試行の正答を right とした方が楽にデータ処理できるのです。次節でこの点について補足します。

4.5 反応の記録方法を工夫しよう

この章の実験の手続きは、心理物理学的測定法のひとつである恒常法の手続きです。恒常法を用いた実験のデータ分析でよく用いられる方法が心理物理曲線の作成です。 図 4.9 に今回の実験で得られる心理物理曲線の例を示します。横軸はテスト刺激の傾き、縦軸に時計回りに傾いているという反応の頻度です。物理的な刺激と反応が一致していれば、横軸の 0 度を境界として左側では縦軸の値は 0.0、右側では 1.0 となりますが、グラフは 0.0 から 1.0 へなだらかに上昇する曲線を描いています。グラフが 0.0 から 1.0 へ変化する範囲が左右に寄っていれば、実験参加者の判断が全体的に偏っていたことがわかります。また、この範囲が左右に広がっていれば、反時計回りか時計回りかの判断が難しい課題であったことがわかります。

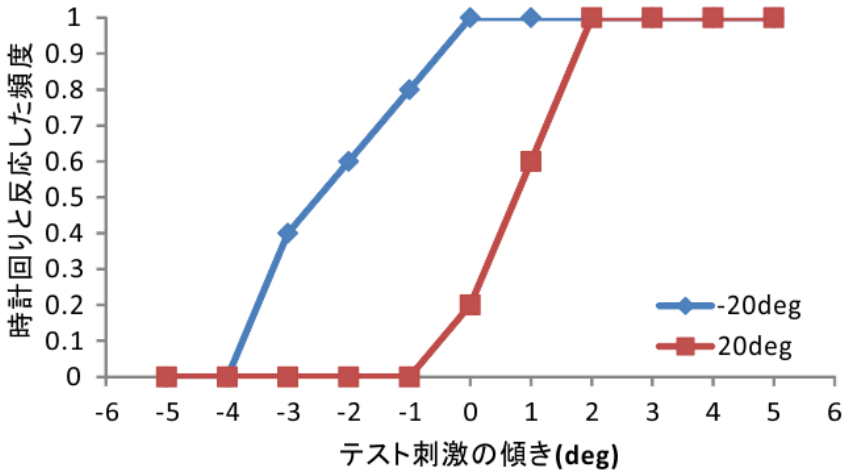


図 4.9　心理物理曲線の例。横軸にテスト刺激の傾き、縦軸に時計回りに傾いているという反応の頻度をプロットしています。折れ線グラフはそれぞれコンテクスト刺激の傾きが-20 度と 20 度の条件に対応しています。

心理物理曲線を描く時には、実験参加者の反応が刺激の物理的特性と一致していたか否かは関係がありません。ですから、PsychoPy で実験参加者の反応を記録する時に、両者が一致していたかを記録しても何の役にも立ちません。前節のように、「right(=時計回りに傾いている)」という反応をしていたかを記録するようにしておけば、正答率の値がそのまま心理物理曲線の縦軸の値として利用できます。分析の手間が大幅に省けます。便利なテクニックですので、ぜひ覚えておいてください。

チェックリスト

- 恒常法の実験において、正答率がそのまま心理物理曲線の縦軸の値として使用できるように正答を定義できる。

4.6 実験情報ダイアログで条件ファイルを指定しよう

この節からがいよいよこの章の本題です。exp04.psyexp ではコンテクスト刺激の傾きとして-20 度と 20 度を用いましたが、これらの条件に加えて-70 度と 70 度傾いた条件のデータも取りたいとします。さらに、-20 度／20 度のコンテクストを使う試行と、-70 度／70 度のコンテクスト刺激を使う試行はそれぞれまとめて実施することにします。つまり、実験を二つのブロックに分割し、一方のブロックではコンテクスト刺激はすべて-20 度／20 度、もう一方のブロックではすべて-70 度／70 度とします。

コンテクスト刺激の種類で実験をブロック化せず、全部無作為な順番で実施するのであれば、条件ファイルに-70 度／70 度の条件に対応する行を追加するだけで対応できます。しかし、ブロック化するのであればこの方法は使えません。一番簡単な方法は、-70 度／70 度条件に対応する新たな条件ファイル (exp04_70.xlsx) を作り、この exp04_70.xlsx を条件ファイルとして使用する実験をもう一個作成するというものでしょう。まあ別にこの方法で乗り切っても構わないのですが、せっかくですのでひとつの実験ファイルで二つの条件ファイルを切り替える方法を習得しましょう。

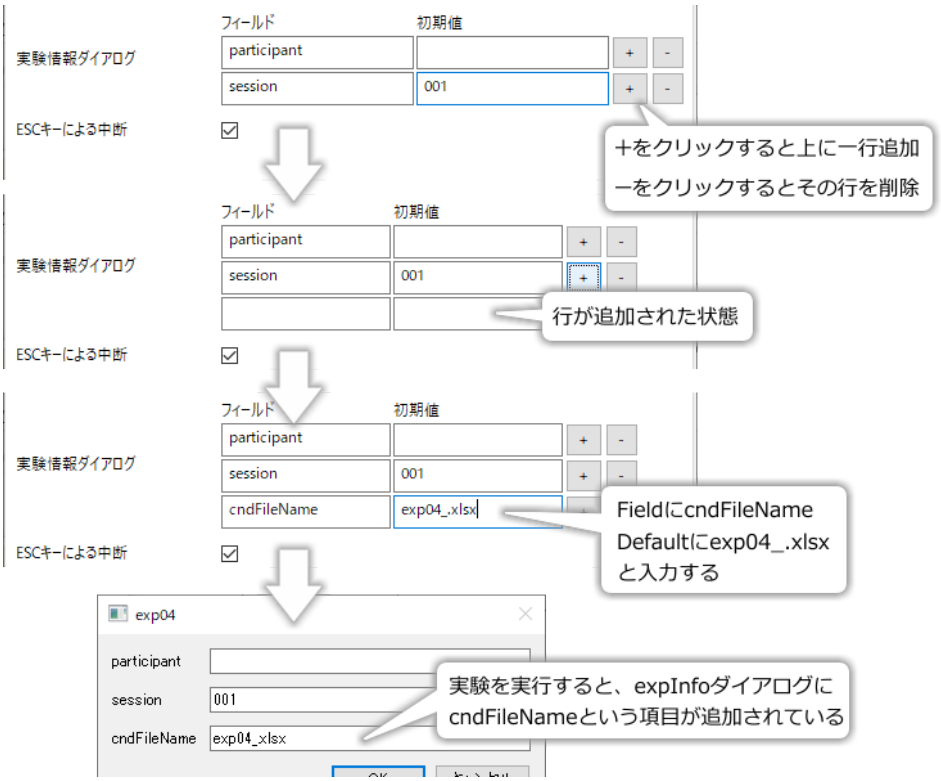


図 4.10　実験情報ダイアログに項目を追加する手順。

使用する条件ファイル名をはじめ、実験のパラメータを実験開始時に指定するには実験情報ダイアログを使用します。初期状態では実験情報ダイアログには session と participant の 2 項目しかありませんが、実験設定ダイアログから項目を追加することができます。 図 4.10 は実験情報ダイアログに項目を追加する手順を示しています。exp04.psyexp を開いて設定ダイアログを開いてみましょう。 **[実験情報ダイアログを表示]** のチェックを外している人はチェックしなおしておいてください。初期状態では実験設定ダイアログの **[実験情報ダイアログ]** に session と participant という項目が表示されていて、その右側に [+]、[-] が書かれたボタンが表示されています。[+] ボタンを押すと、その行の下に新しい行が追加されます。[-] を押すと、その行が削除されます。participant と session のどちらの行の下に追加しても、動作には違いがありません。

では、[+] を押して新しい行を追加して、「フィールド」に cndFileName、「初期値」に exp04_.xlxs と入力して

ください。入力したら実験を実行してみましょう。すると、 図 4.10 の一番下の図のように実験情報ダイアログに cndFileName という項目が追加されていて、exp04_.xlsx という文字列が最初から入力されているはずです。「フィールド」は実験情報ダイアログに表示する項目の名前、「初期値」はその項目に最初から入力されている文字列 (初期値) に対応しています。participant のように「初期値」が空白の場合は、実験情報ダイアログでも空白になります。

実験情報ダイアログの項目名は、条件ファイルのパラメータ名と異なり、空白文字 (スペース) や#、$といった記号を含むことができます。日本語の文字列でも指定できますが、表示が乱れることがありますのであまりお勧めしません。項目が実験情報ダイアログに表示される順番は、従来のバージョンでは PsychoPy が自動的に決定されてしまって自由に指定できなかったのですが、現在のバージョンでは実験設定ダイアログで入力した通りの順番となります (2020.2.6 で確認)。

実験情報ダイアログで入力した値は、条件ファイルに記述されたパラメータのように Builder 内部で参照することができます。ただし、条件ファイルの場合とは書き方が異なっていて、expInfo[' パラメータ名'] という具合に書きます。この書き方の意味を理解するためには Python の文法を理解する必要がありますので、ここではとりあえず「こう書くんだ」と覚えておいてください。

それでは、実験情報ダイアログを利用して「ひとつの実験ファイルで二つの条件ファイルを切り替える」という問題に挑戦してみましょう。先ほどの cndFileName という項目を exp04.psyexp に追加した状態から作業を続けます。trials ループのプロパティを開いて、 **[条件]** を$expInfo['cndFileName'] に変更してください。 **[条件]** には$が付いていないので、条件ファイルからパラメータを読む時と同様に$を付けないといけない点に注意してください (図 4.11)。これで、実験実行時に実験情報ダイアログの cndFileName の項目に入力された名前の条件ファイルを開くようになりました。

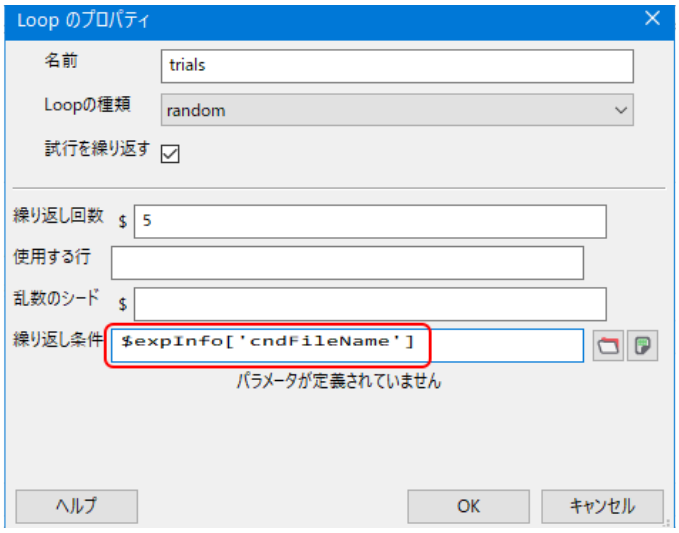


図 4.11　**[条件]** に実験情報ダイアログの項目を指定します。先頭の$と項目名の前後の' を忘れないように注意してください。

続いて、Builder をいったん離れてコンテクスト刺激が-70 度／ 70 度傾いている条件の条件ファイルを作成しましょう。これは単に exp04_20.xlsx を開いて、contextDir の列の-20 を-70 に、20 を 70 に書き換えて別名で保存するだけです。ここでは exp04_70.xlsx という名前で保存しておきましょう。元の exp04_20.xlsx も引き続き使用しますので、上書き保存しないように注意してください。exp04_20.xlsx、exp04_70.xlsx の両方とも exp04.psyexp と同じフォルダに置いてください。

以上で実行時に条件ファイルを切り替えられる実験の作成は終了です。PsychoPy に戻って exp04.psyexp

を実行しましょう。実験情報ダイアログの cndFileName に exp04_20.xlsx と入力すれば、-20 度／ 20 度、exp04_70.xlsx と入力すれば-70 度／ 70 度の条件の試行が始まります。最初から exp04_.xlsx という文字列が入力されているので、_の後に 20 または 70 と入力するだけでよいはずです。ぜひ、-70 度／ 70 度の実験を最後まで行って、-20 度／ 20 度の実験結果と比較してみてください。

チェックリスト

- 実験情報ダイアログの項目を追加、削除することができる。
- 実験情報ダイアログの項目の初期値を設定することができる。
- 実験情報ダイアログの項目名から、その値を利用するための Builder 内における表記に変換することができる。
- 条件ファイル名を実験情報ダイアログの項目から取り出してループのプロパティに設定することができる。

4.7 実験情報ダイアログで項目を選択できるようにしよう

前節で実験情報ダイアログを使って条件ファイルを切り替えできるようになりましたが、実際に使用してみた感想はいかがでしょうか？ 確かに便利なのですが、条件ファイル名を入力するのが面倒ではないでしょうか。今回の例のように入力する内容がいくつかの決まったパターンしかない場合、いちいちキーボードから入力するのではなくドロップダウンメニューから選択できるようにする機能が Builder には用意されています。

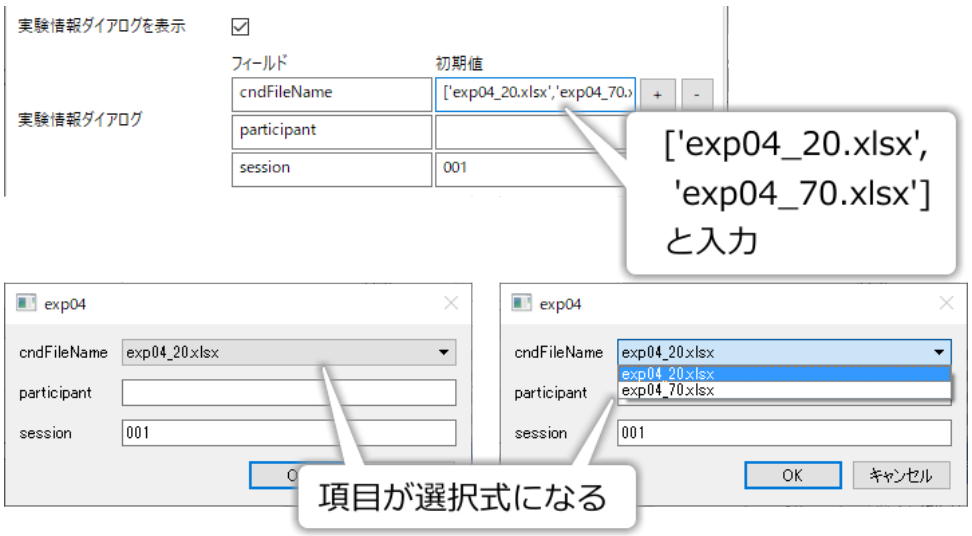


図 4.12　実験情報ダイアログの項目を選択式にする

さっそく使ってみましょう。exp04.psyexp を開いて設定ダイアログを開き、先ほど編集した **[実験情報ダイアログ]** の cndFileName の項目を以下のように変更します。[ ] や' '、, などの記号に注意して入力してください。

```
['exp04_20.xlsx', 'exp04_70.xlsx']
```

入力できたら実験を実行してみましょう。 図 4.12 に示すように、cndFileName の項目が exp04_20.xlsx と exp04_70.xlsx のどちらかを選べるようになりました。この形式をドロップダウンメニューと呼びます。これで条件ファイルの切り替えが少し楽になったのではないでしょうか。

さて、動作を確認したところで、この機能を使うためのポイントを確認しておきましょう。**[実験情報ダイアログ]** の項目の「初期値」が以下の条件を満たすとき、その項目はドロップダウンメニューと解釈されます (上級者向け：厳密には「初期値」に記入されている内容が Python の list オブジェクトとして解釈可能であることが条件)。

1. メニューの各項目が ' ' または " " で囲まれていて、かつ , で区切られている。
2. 1. の内容が [と] で囲まれている。 [の前や] の後ろには文字があってはいけない。

1 番目の条件は、Keyboard コンポーネントの **[検出するキー $]** で'left', 'right' という具合にキー名を ' ' または " " で囲ってカンマ区切りで並べたとの同じように書くということです。いろいろな値を追加して練習してみてください。

チェックリスト

- 実験情報ダイアログの項目をドロップダウンメニューにできる。

4.8 多重繰り返しを活用しよう

今回の実験のようにある要因 (ここではコンテクスト刺激の傾き角度) で試行がブロック化されている場合、一人の実験参加者が両方のブロックへ参加する実験計画を用いることもあれば (参加者内計画)、一人の実験参加者はいずれか一方のブロックしか参加しない計画を用いることもあります (参加者間計画)。前節の実験情報ダイアログで実験条件を指定する方法は、参加者内計画でも参加者間計画でも柔軟に対応できますが、実験者がいちいち条件ファイル名を設定しないといけないので面倒です。面倒なだけならいいのですが、条件ファイル名を間違えてしまうかもしれません。参加者内計画の場合、一方の条件を実行したら次のブロックは必ず残りの一方の条件です。Builder に自動的に二つの条件ファイルを読み込んで実行させるように改造してみましょう。

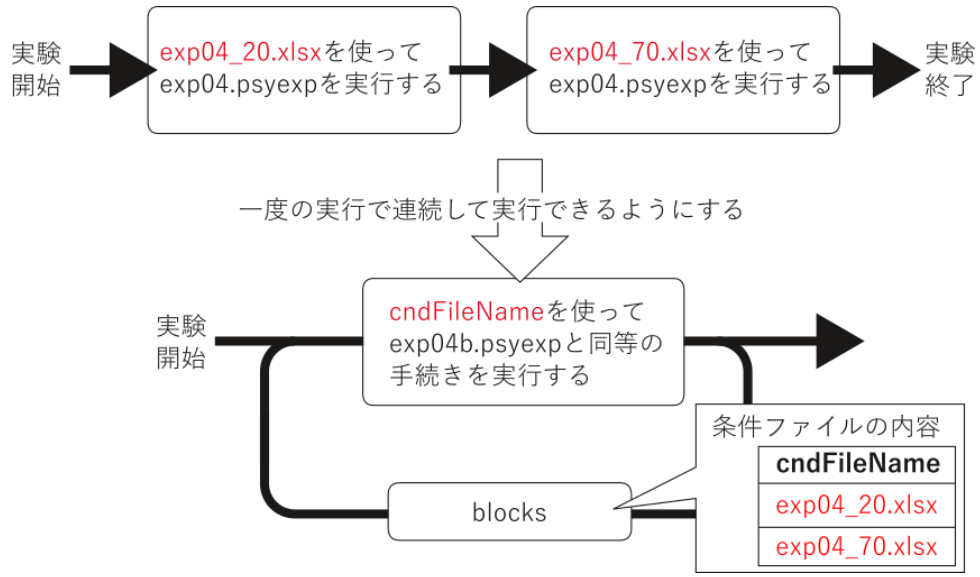


図 4.13　読み込む条件ファイルを変更しながら繰り返すことによって、-20 度／ 20 度と-70 度／ 70 度の条件を連続して実行する実験を作成します。

図 4.13 に改造の方針を示します。前節までの状態では、 図 4.13 の上の図のように、条件ファイル名を変更しながら 2 回 exp04.psyexp を実行しなければいけません。これは、今まで作ってきた実験における「刺激の色

や傾きを変更しながら刺激提示を繰り返す」という作業とよく似ています。ということは、刺激の色や傾きを条件ファイルから読み込んで代入しながら繰り返すことができたように、 図 4.13 下のようにすれば条件ファイルの名前を別の条件ファイルから読み込んで代入しながら繰り返すことができるはずです。

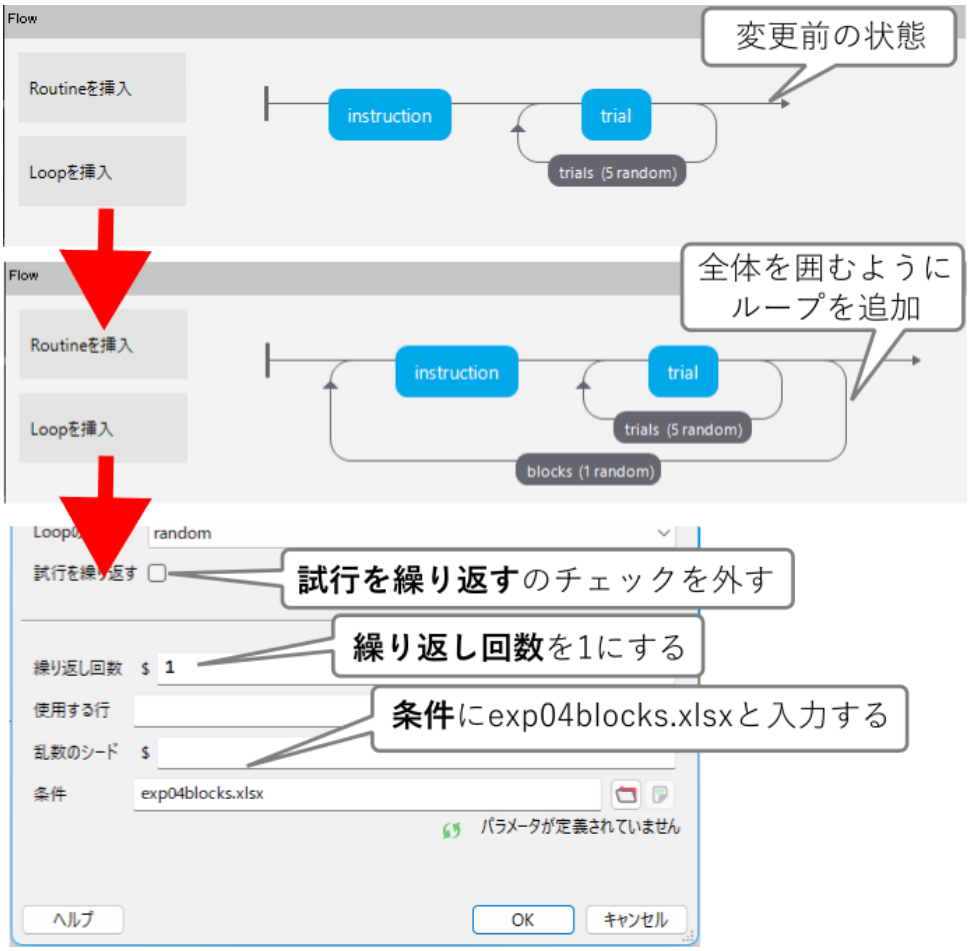


図 4.14　多重に繰り返しを挿入します。

さっそく改造してみましょう。Builder で exp04.psyexp を開いて、 図 4.14 のように blocks というループを挿入してください。Blocks ループの始点は instruction ルーチンの後ろでも構わないのですが、後ろに挿入してしまうと 1 回目の trials ループが終わった直後に 2 回目の trials ループが始まってしまうため、実験参加者に対する予告なしにいきなり-20 度／ 20 度条件と-70 度／ 70 度条件が切り替わってしまいます。instruction ルーチンを bolorks ループに含むようにしておけば、条件が切り替わる前に教示画面が再び表示されますので、切り替わりがわかりやすいでしょう。さらにわかりやすくするためには、trial ルーチンの前にこれから始まる条件を表示するためのルーチンを挿入すると良いでしょうが、これは練習問題とします。bocks ループのプロパティでは、 **[繰り返し回数 $]** を 1 に設定して、 **[条件]** に exp04blocks.xlsx と入力しておきます。 **[Loop の種類]** は random のままでよいでしょう。これで新しいループの追加は完了です。

続いて、blocks ループのための条件ファイル、exp04blocks.xlsx を用意しましょう。この条件ファイルでは 図 4.15 左上のように exp04_20.xlsx と exp04_70.xlsx の二つの値を持つ cndFileName というパラメータを設定します。続いて trials ループで cndFileName に基づいて条件ファイルを読み込むように設定するために、trials ループの **[条件]** を$cndFileName に変更します (図 4.15 右上)。実験情報ダイアログで条件ファイル名を入力する必要はなくなったので、実験設定ダイアログを開いて実験情報ダイアログから cndFileName の行を削除しておきましょう (図 4.15 下)。

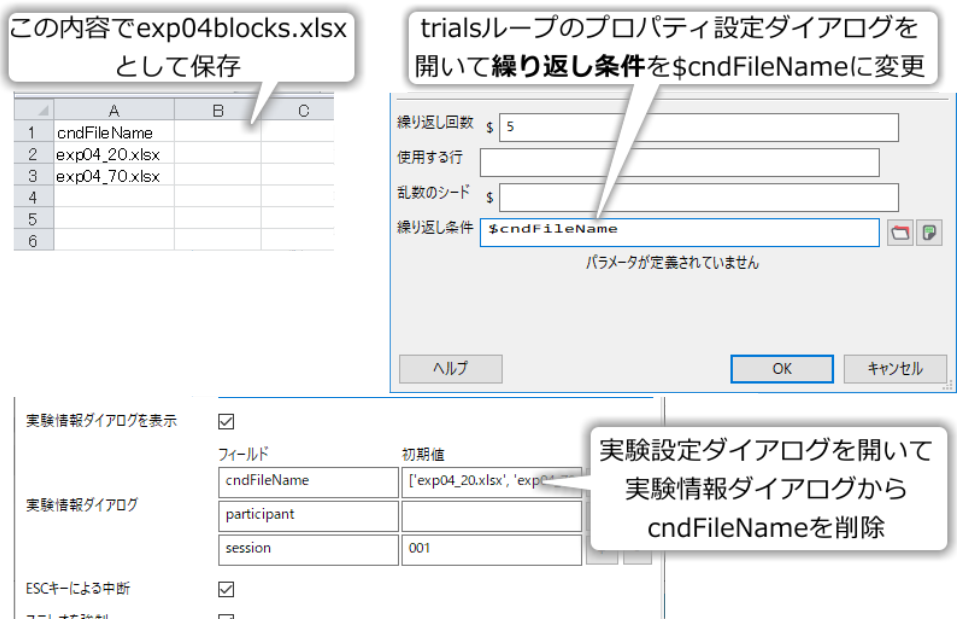


図 4.15　blocks ループを使って条件ファイルを切り替えるように trials ループなどを修正します。

以上で改造は終了です。exp04.psyexp を実行して、自動的に-20 度／ 20 度条件と-70 度／ 70 度条件が続けて実行されることを確認してください。blocks ループの **[Loop の種類]** を random のままにしたので、-20 度／ 20 度条件と-70 度／ 70 度条件のどちらが先に実行されるかは毎回無作為に決定される点に注意してください。

なお、ループのプロパティには **[試行を繰り返す]** という項目がありますが、この項目は多重繰り返しを使ったときのデータ保存形式に関係があります。詳しくは「4.11.2: ループの **[試行を繰り返す]** プロパティについて」を参照してください。

チェックリスト

- 多重繰り返しを挿入できる。
- 多重繰り返しの内側のループで条件ファイル名を外側のループの条件ファイルから読み込んで設定することができる。

4.9 動作確認のために一部の動作をスキップしよう

前節で、-20 度／ 20 度条件と-70 度／ 70 度条件を一回の実験実行でおこないました。「exp04.psyexp を実行して、自動的に-20 度／ 20 度条件と-70 度／ 70 度条件が続けて実行されることを確認してください」と書きましたが、皆さん実際にしていただけましたか？少なくとも一方の条件が終了してもう一方の条件が始まるところまでは実行しないと動作確認になりませんが、そこまでたどり着くには 110 試行を行わないといけません。1 試行ごとに 0.5 秒待って 1 回キーを押さないといけないのでかなり面倒です。

こういう時に便利なのが Routine ペインの左上にある「Routine の設定」と書かれているボタンです。このボタンをクリックすると、 図 4.16 のようなダイアログが開きます。これはルーチンのプロパティ設定ダイアログで、コンポーネントのプロパティを設定するのと同様にルーチンのさまざまな設定を変更できます。「基本」、「Flow」、「ウィンドウ」、「データ」、「テスト」の 5 つのタブがありますが、本節で紹介しておきたいのは 2 つです (「 11.3: 特定の条件を満たさない時に実行されるルーチンを活用して休憩画面を設けよう にも

「Routine の設定」を利用するテクニックが登場します)。

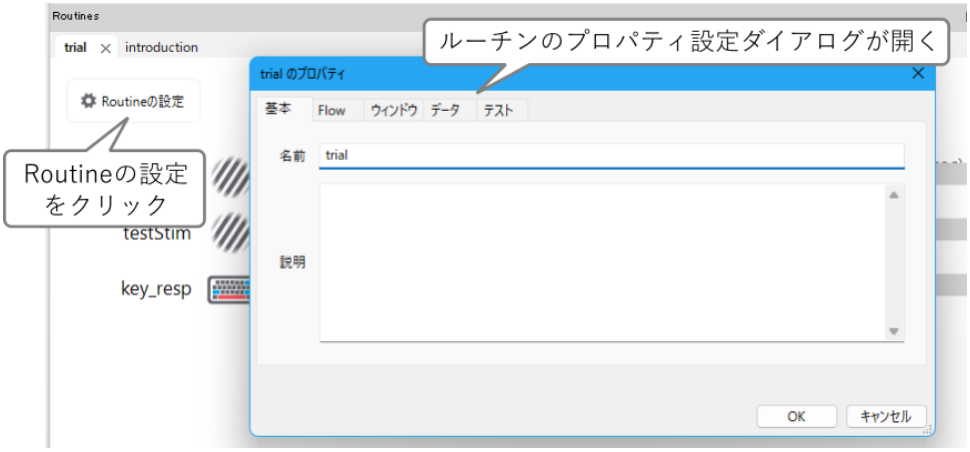


図 4.16　ルーチンのプロパティ設定ダイアログ

まず「テスト」タブをクリックすると、**[ルーチンの無効化]** という項目があります (図 4.17 左上)。この項目にチェックすると、 図 4.17 右上のように、フローに配置されている当該ルーチンがグレーで表示され、実験実行時に実行されなくなります。この例では、instruction ルーチンが終了した後 trials ループに入りますが、trial ルーチンが無効化されていてループの中身が空っぽなので一瞬で終了してしまいます。その結果、trails ループもあっという間に終了して blocks ループの 2 周目に入り、あたかも instrction ルーチンが続けて 2 回実行されたかのような動作になります。

もうひとつ紹介しておきたいのは、「Flow」タブの **タイムアウト** という項目です (図 4.17 左下)。タイムアウトを設定すると、ルーチン内にまだ終了していないコンポーネントが存在していても強制的にルーチンが終了します。例えば trial ルーチンで 図 4.17 左下のように **タイムアウト** を「実行時間 (秒)」で 0.6 に設定すると、図 4.17 右下のようにルーチンのタイムラインの 0.6 秒のところにオレンジ色の縦線が引かれ、コンポーネントの有効時間が 0.6 秒を越えて残っていてもグレーで表示されます。この状態で実験を実行すると trial ルーチンが 0.6 秒で終了してしまうので、trials ループに入った後 0.5 秒間の空白画面と 0.1 秒間の刺激描画が何も操作しなくても繰り返されていきます。**[ルーチンの無効化]** のように完全にルーチンが実行されなくなってしまうと、動作確認として不十分と言うか、本当にきちんと動作しているのか不安になりますが、一瞬だけ刺激を表示しておくと「ああ、確かにここで刺激が表示されるんだ」と確認しつつ、自分自身でキー押し操作をすることなく blocks ループの 2 回目の実行まで進めることができます。**[ルーチンの無効化]** と場面に応じて使い分けるとよいでしょう。

なお、今回の実験では trails ループのなかにルーチンは 1 つしかありませんが、多数のルーチンがある場合にひとつひとつ　**[ルーチンの無効化]** をチェックしていくのは面倒です。このような場合はループの **[繰り返し回数]** を 0 にすると、「ループが 0 回実行される = 1 度も実行されない」ということで、ループ内のルーチンをまとめてスキップすることができます。このテクニックも覚えておくとよいでしょう。

関連テクニックとして、コンポーネントのプロパティ設定ダイアログにも「テスト」タブがあり、**[コンポーネントの無効化]** という項目があることも記しておきます (図 4.18)。これは「あるルーチンに含まれる一部のコンポーネントだけ実行したくない」場合に利用できます。例えば「実験を実行してみるとエラーで停止してしまうのだけれど、どのコンポーネントが原因なのかわからない」場合などに便利です。原因不明のエラーで困っている時に、あるコンポーネントを無効化して実行するとエラーが発生しないのなら、きっとそのコンポーネントがエラーに関与しているはずです。中級者から上級者向けの機能かも知れませんが、こんな機能があるということを頭の片隅に置いておくと役に立つかもしれません。

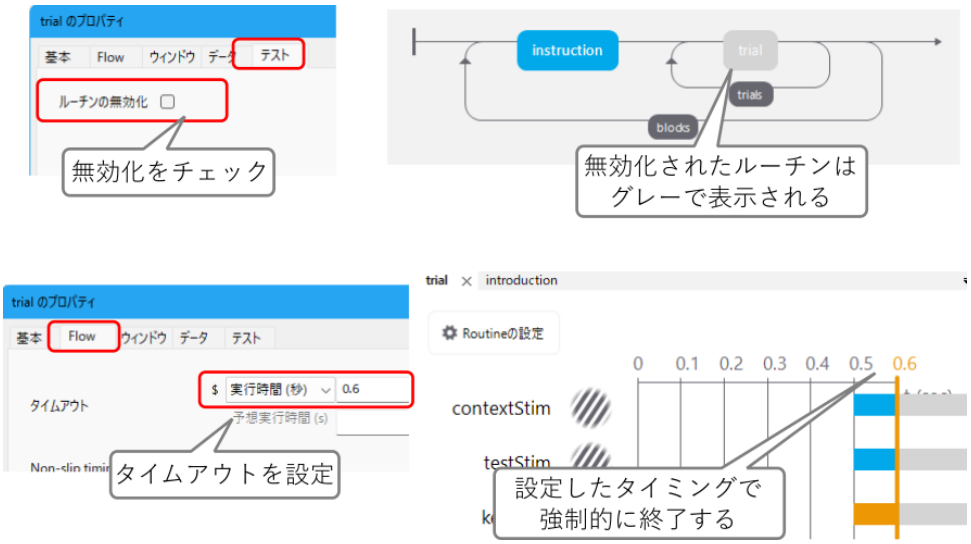


図 4.17　ルーチンのプロパティ設定によるルーチン実行スキップ。 **[ルーチンの無効化]** をチェックすると、実行時にそのルーチンはまったく実行されなくなります。

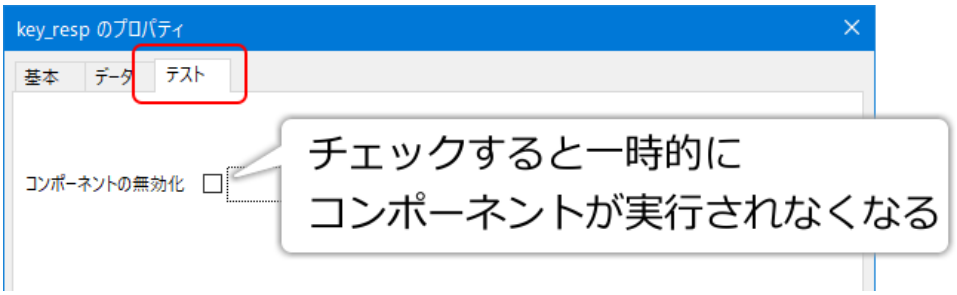


図 4.18　「テスト」タブの **[コンポーネントの無効化]** をチェックするとコンポーネントを一時的に無効化できます。

チェックリスト

- 一時的に、特定のルーチンを実行しないようにすることができる。
- ルーチンを、まだ実行中のコンポーネントが残っていても指定した実行時間で強制的に中断させることができる。
- 一時的に、特定のループを実行しないようにすることができる。

4.10 参加者間でカウンターバランスをとろう (中級)

以上で多重ループを用いて複数のブロックを一回の実験でまとめて実行するという目標が達成できましたが、現状では-20 度／ 20 度条件と-70 度／ 70 度条件のどちらが先に実行されるかはランダムに決定されます。「本当にランダムならすべての選択肢がほぼ均等な頻度で生じるはずだ」と考えている人は結構多いのですが、実際に試してみると意外と偏りがあるものです。本章の実験を 20 名の参加者に実行してもらったとして、「-20 度／ 20 度条件が先に実行された人は 20 名中 3 名しかいなかった」ということが十分に起こり得ます。しかし、心理学実験では「各参加者にランダムに実行順序を割り当てつつ、各順序が割り当てられた人数を等しくしたい」といったことがよくあります。

解決策のひとつは、「4.6: 実験情報ダイアログで条件ファイルを指定しよう 」および「4.7: 実験情報ダイアログで項目を選択できるようにしよう 」で覚えたテクニックを使って、実験開始時に-20 度／ 20 度条件を

先にするか-70 度／ 70 度条件を先にするか選べるようにすることです。具体的には、blocks ループの条件ファイル名を実験情報ダイアログから取得することにして、blocks ループ用の条件ファイルを exp04_20.xlsx、exp04_70.xlsx の順に並べたものと、exp04_70.xlsx、exp04_20.xlsx の順に並べたものの 2 種類を用意します。そして、条件ファイルに並べた順番通りにブロックを繰り返すように blocks ループの **Loop の種類** を sequential にするのがポイントです (図 4.19)。これらの条件ファイルを「4.7: 実験情報ダイアログで項目を選択できるようにしよう」のように実験情報ダイアログで選択できるようにすればよいでしょう。これらは練習問題としておきます。

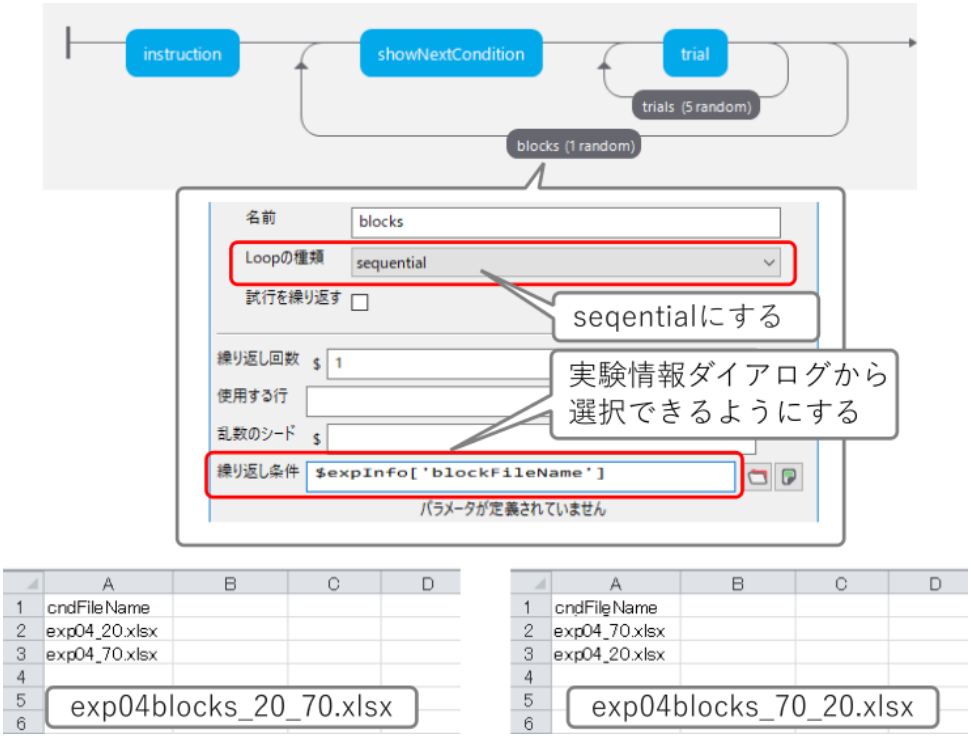


図 4.19　ブロックの順序を実験実行時に選択できるようにします。

このように人が手作業で管理する方法は、さまざまな事態に臨機応変に対応できる利点もあるのですが、心理学実験の面倒な手続きを PC に任せよう！というのが本書のテーマなので、なんとか PC 任せにしてしまいたいところです。そこで登場するのが PsychoPy 2024 から導入された Counterbalance ルーチンです。ルーチンという名前ですが、他のコンポーネントと同様コンポーネントペインから選択します (「カスタム」カテゴリの中にあります)。すると 図 4.20 のように、ルーチンペインに counterbalance という新しいルーチンが追加されて、ルーチンペインいっぱいに counterbalance ルーチンのプロパティ設定ダイアログが表示されます。このルーチンは通常のルーチンとは異なり、コンポーネントを配置することができませんので、 図 4.20 の右側のコンポーネントペインに並んだコンポーネントが薄い色で表示されてクリックできないようになっています。ルーチンペインの表示を trial ルーチンなどの通常ルーチンに切り替えるとコンポーネントを置けるようになります。

Counterbalance ルーチンの設定をおこなうには、 **スロット** と **グループ** という用語を理解しておく必要があります (図 4.21)。スロットは区画のことで、ここでは「実験 1 回」を指しています。グループはスロットをまとめて入れておく器のようなものと考えてください。図 4.21 左上は、「0」、「1」とラベルをつけられた 2 つのグループに 5 個ずつスロットが入っています。実験を 1 回実行するごとに (正確には Counterbalance ルーチンが実行されるごとに)、ランダムに選ばれたいずれかのグループからスロットがひとつ消費されます。10 回実験を実行するとスロットの残りは 0 個になります。図 4.21 右上に「全てのグループのスロットが空になると

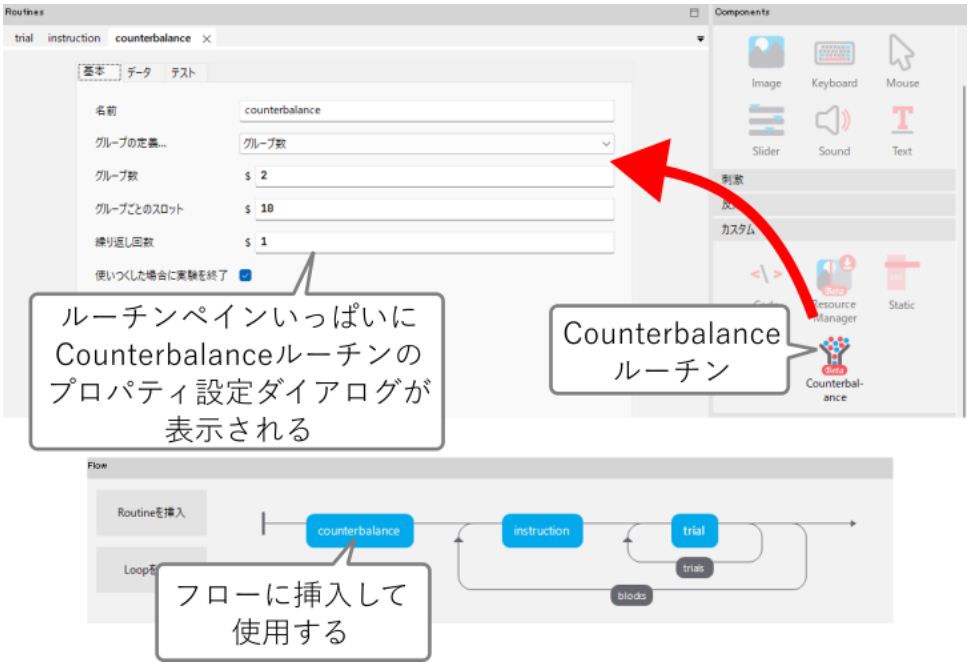


図 4.20　Counterbalance ルーチン。通常のコンポーネントと異なり、コンポーネントペインの Counterbalance ルーチンのアイコンをクリックするとルーチンが新たに追加され、ルーチンペインにプロパティ設定ダイアログが表示されます。

グループは None になる」とありますが、None は Python で「ない」ということを表す値です (これまでにも「2.5: 色の指定方法を理解しよう」で塗りつぶさないことを指定する時などに出てきました)。ここまで説明したらなんとなく想像できた方もおられると思いますが、Counterbalance ルーチンを実行した後に「どのグループからスロットが取り出されたか」に応じて条件ファイルを切り替えらえたら、参加者間のカウンターバランスが実現できます。

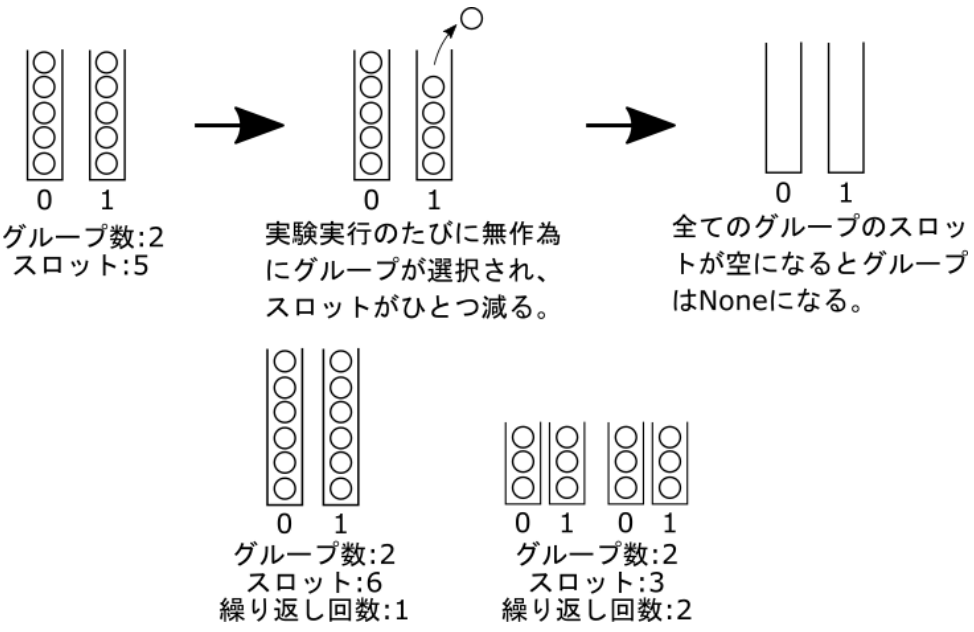


図 4.21　Counterbalance コンポーネントの働き

スロットとグループの説明を終えたところで、Countebalance ルーチンのプロパティを説明していきましょう。

[グループの定義...] はグループの名前と個数、スロットの個数をどのように定義するかを選択します。「グループ数」を選ぶと下に **[グループ数]** と **[グループごとのスロット]** というプロパティが表示され、ここに数値を入力して定義することができます。「条件ファイル (ローカルのみ有効)」を選ぶと **[条件]** というプロパティが表示され、xlsx ファイルや CSV ファイルを使用してグループを定義できます。こちらの方が柔軟な定義ができるので、本章ではこちらの方法を解説します。**[繰り返し回数]** はちょっと後回しにして先に **[使いつくした場合に実験を終了]** を解説しましょう。これは文字通り、スロットが使いつくされて残っていないのに実験を実行すると、このルーチンのところで強制的に実験が終了してしまうということです。通常はチェックしておくと良いでしょう。チェックしなかった場合は「どのグループも選択されていない」という状態でフローの次のルーチンへ進んでしまいます。スロットの残りがない時にいきなり実験を終了するのではなく、なにか処理をおこないたい場合に便利ですが、Builder の実験がどのように Python のコードに変換されるかを理解したうえでコードを書かないといけないので上級者向けです。

[使いつくした場合に実験を終了] をチェックしているという前提で **[繰り返し回数]** について説明します。図 4.21 左下は「グループ数 2、スロット数 6 で繰り返し回数 1」と設定した場合で、2 × 6=12 回実験を実行すると残りスロットが 0 になります。繰り返し回数が 1 なので、13 回目の実験を実行するとスロットの残りがないので **[使いつくした場合に実験を終了]** の設定に従って実験が強制的に終了します。図 4.21 右下は「グループ数 2、スロット数 3 で繰り返し回数 2」と設定した場合で、2 × 3=6 回実験を実行すると残りスロットが 0 になりますが、繰り返し回数が 2 なのでスロットがすべて満たされた状態に戻ります。さらに 2 × 3=6 回、合計 12 回実験を実行すると再びスロットが 0 になり、次 (13 回目) の実行では強制的に終了します。結果的にどちらの設定でもグループ 1 を 6 回、グループ 2 を 6 回の 12 回実行されますが、 図 4.21 左下の例では「グループ 1 が 6 回連続した後、グループ 2 が 6 回連続する」といったことが起こり得ます。 図 4.21 右下の例ではそういったことは起こりません。 **[Loop の種類]** の fullRandom と random の違いに似ています (「 3.6: 繰り返しを設定しよう 」)。

	A	B	C	D
1	group	cap	blockFile	
2	70度条件から	4	exp04_70_20.xlsx	
3	20度条件から	6	exp04_20_70.xlsx	
4				

図 4.22　xlsx ファイルで Counterbalance コンポーネントのグループとスロットを定義できます。

図 4.22 は「条件ファイル (ローカルのみ有効)」でグループとスロットを定義する時の xlsx ファイルの例です。**group と cap というパラメータが必須** で、 group は各グループにつけるラベル、cap は各グループに割り当てるスロット数です。この例では「70 度条件から」と「20 度条件から」というグループが定義され、「70 度条件から」には 4、「20 度条件から」には 6 のスロットが割り当てられています。条件ファイルを用いると、このようにグループによって異なるスロット数を割り当てることが可能です。図 4.22 には blockFile というパラメータも定義されていますが、これはグループに応じて変化させたいパラメータとして使用できます。この例では「70 度条件から」に exp04_70.xlsx、「20 度条件から」に exp04_20.xlsx というファイル名を割り当てることによって、blocks ループで使用する条件ファイルを切り替えることを意図しています。Excel はひとつのセルにかなり長い文字列 (例えば複数の改行を含む 100 文字以上の文字列) を入力できるので、 **グループごとに教示を切り替えるといったことも可能** でしょう。せっかくですので、この条件ファイルを使って exp04.psyexp を改造してみましょう。

条件ファイルの準備

- 図 4.22 の内容の xlsx ファイルを作成し、exp04groups.xlsx という名前で exp04.psyexp と同じフォルダに保存する。
- 図 4.19 に示されている exp04_70_20.xlsx と exp04_20_70.xlsx を作成し、exp04.psyexp と同じフォルダに保存する (先ほどの練習問題を済ませていたらそのまま流用できます)。

— Counterbalance ルーチンの配置

•Builder で exp04.psyexp を開き、Counterbalance ルーチンを追加 (コンポーネントペインの Counterbalance ルーチンのアイコンをクリック) して以下の通り設定する。

- **[グループの定義...]** を「条件ファイル (ローカルのみ有効)」に変更する。
- **[条件]** に exp04groups.xlsx を指定する (このプロパティは **[グループの定義...]** を「条件ファイル (ローカルのみ有効)」にしないと表示されません)。
- **[繰り返し回数]** が 1 であり、 **[使いつくした場合に実験を終了]** がチェックされていることを確認する。

- フローペインの「Routine を挿入」をクリックすると、挿入できるルーチンの一覧に Counterbalance という名前のルーチンがあるので、そのルーチンを選択してフローの先頭 (instruction ルーチンの前) に挿入する。

blocks ループの変更

- **[Loop の種類]** を sequential にする (図 4.19 の作業をしていたら既に sequential になっている)。
- blocks ループのプロパティを開き、 **[条件]** に `$counterbalance.params['blockFile']` と入力する。

blocks ループの **[条件]** に入力している内容を説明するとかなり話が長くなるので、ここはとりあえず Counterbalance コンポーネントの **[名前]** に.params[] と付けると、exp04groups.xlsx で定義したカウンターバランスのパラメータを参照できると覚えておいてください。[] の中に' ' で囲んでパラメータ名を書くのは expInfo と同じですね。

その他

- 動作確認を短時間で済ませるため、trials ループの **[繰り返し回数]** を 1 にしておく。
- 先ほどの練習問題を済ませているなら、実験情報ダイアログに xp04_70_20.xlsx と exp04_20_70.xlsx を選択するための項目が追加されているはずなので、実験設定ダイアログからその項目を削除しておく。

以上で作業は終わりです。ちょっと大変ですが、実験を 10 回実行して、exp04groups.xlsx の cap に設定した通りに「70 度条件からが始まる実験が 4 回、20 度条件から始まる実験が 6 回」になることを確認しましょう。trials ルーチンのプロパティ設定ダイアログから **[タイムアウト]** を設定すると、キーを押さなくても次々と試行が進むので楽です。**実験にエラーがあって途中で止まってしまった場合でも、Counterbalance ルーチンが実行されていれば 1 回とカウントされる** ので注意してください。さらに、11 回目を実行しようとすると Attempting to measure frame rate of screen, please wait... のメッセージの後にすぐ実験が終了してしまうことも確認してください。終了してしまった後、Runner の「標準出力」のタブを見ると、 図 4.23 のように「カ

ウンターバランス counterbalance のスロットが使いつくされたため, 実験を終了しました」と出力されているのがわかります (counterbalance のところには Counterbalance コンポーネントの **[名前]** が入ります)。

エラーで止まってしまった回があって「70 度条件からが始まる実験が 4 回、20 度条件から始まる実験が 6 回」だったのかきちんと確認できなかった場合は、exp04.psyexp ファイルがあるフォルダを開いてみてください。shelf.json という名前のファイルができているはずです。ここに残りの繰り返し回数と各グループの残りスロットの情報が保存されているので、 **shelf.json を削除すると最初からカウントし直しになります。** 実験を作成している途中でグループの定義を変更した時や、実際の研究場面で「何回か動作確認をして問題なさそうだから、いよいよ次から本番」という時にも、shelf.json を削除するとよいでしょう。Counterbalance コンポーネントを使うときにはよく覚えておいてください。shelf.json の内容や、xlsx ファイルを使わないグループの定義方法などは「4.11.3:shelf.json および Counterbalance コンポーネントに関する補足 (上級)」を参照してください。

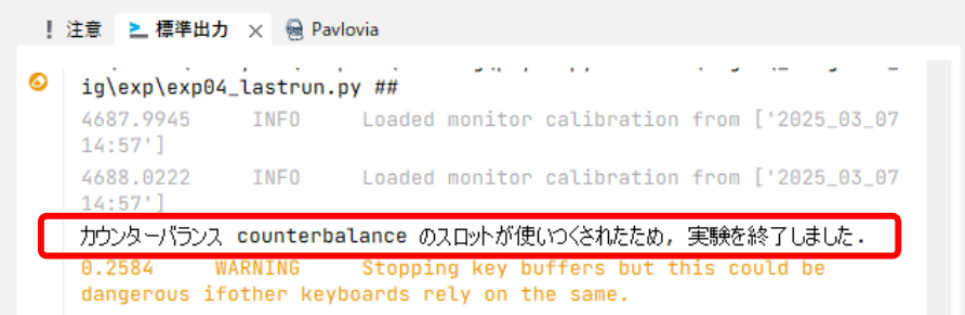


図 4.23　すべてのスロットを使いつくして実験が終了した場合、Runner の標準出力にメッセージが表示されます。

これで参加者間のカウンターバランスを PC に管理してもらうことができるようになりました。話の区切りもちょうどいいので、ここで本章を終わりにしたいと思います。最後の Counterbalance コンポーネントの解説で、counterbalance.params['blockFile'] という入力内容の解説を十分にできませんでした。これがどういう意味を持つのかをきちんと説明しようとすると、どうしても Python の文法に触れる必要があります。次章から、Builder の実験の中で Python のコードを利用する方法を学んでいきましょう。

チェックリスト

- Counterbalance ルーチンにおけるグループとスロットの意味を説明できる。
- Counterbalance ルーチンで各グループに異なるスロット数を設定できる。
- xlsx ファイルでグループとスロットを定義する際に、追加で定義したパラメータの値を利用することができる。
- Counterbalance ルーチンで管理している各グループのスロットの残りをリセットすることができる。

4.11 この章のトピックス

4.11.1 Grating コンポーネントの [テクスチャ] プロパティについて

視知覚の研究でグレーティングと言うと一般的に正弦波状に明るさが変調された刺激を指すのですが、PsychoPy の Grating コンポーネントはより一般的な「同一のパターンが 2 次元に繰り返される刺激」を描く機能を持っています。 図 4.24 は **[テクスチャ]** に指定できる値を示しています。sin は正弦波、saw は鋸波、

sqr は矩形波、tri は三角波を描きます。None を指定すると、 **[前景色]** で指定された色で塗りつぶします。これらの波は 1 次元、すなわち一方向にのみ明るさが変化しますが、sinXsin では垂直方向と水平方向に変化する 2 次元の波を描きます。同様に sqrXsqr、gauss、radRamp、raisedCos といった波形を指定できます。垂直方向と水平方向の空間周波数を別々に指定するためには、 **[空間周波数 $]** に刺激の大きさを指定する時と同様の記法を用います。 図 4.25 左上は sqrXsqr の波形に **[空間周波数 $]** として [2,3]、[3,2] を指定した時の出力を示しています (単位は height)。図からわかるように、第 1 の値が水平方向、第 2 の値が垂直方向に対応しています。

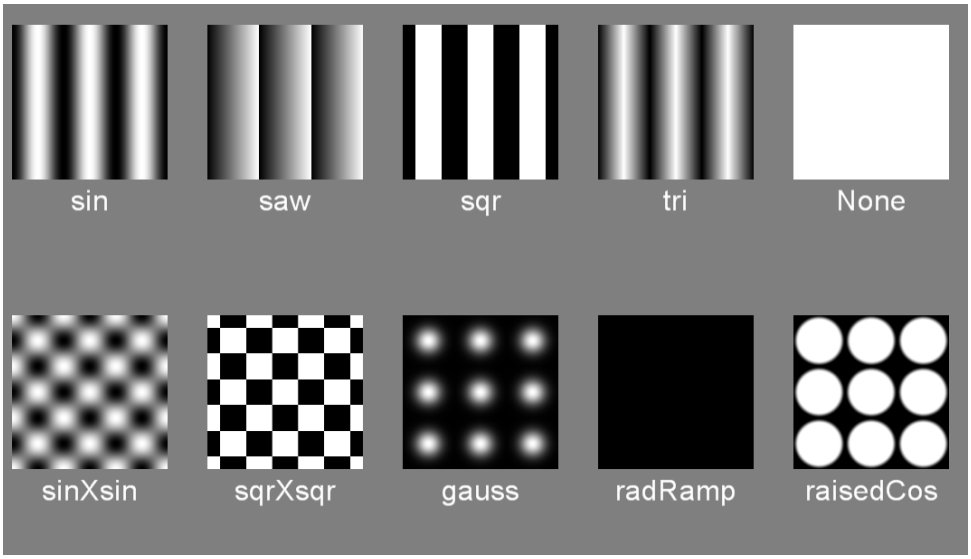


図 4.24　Grating コンポーネントの **[テクスチャ]** として指定できる値。これらの他に画像ファイル名を指定することができます。いずれも **[空間の単位]** は height で **[空間周波数 $]** は 3.0 を指定しています。

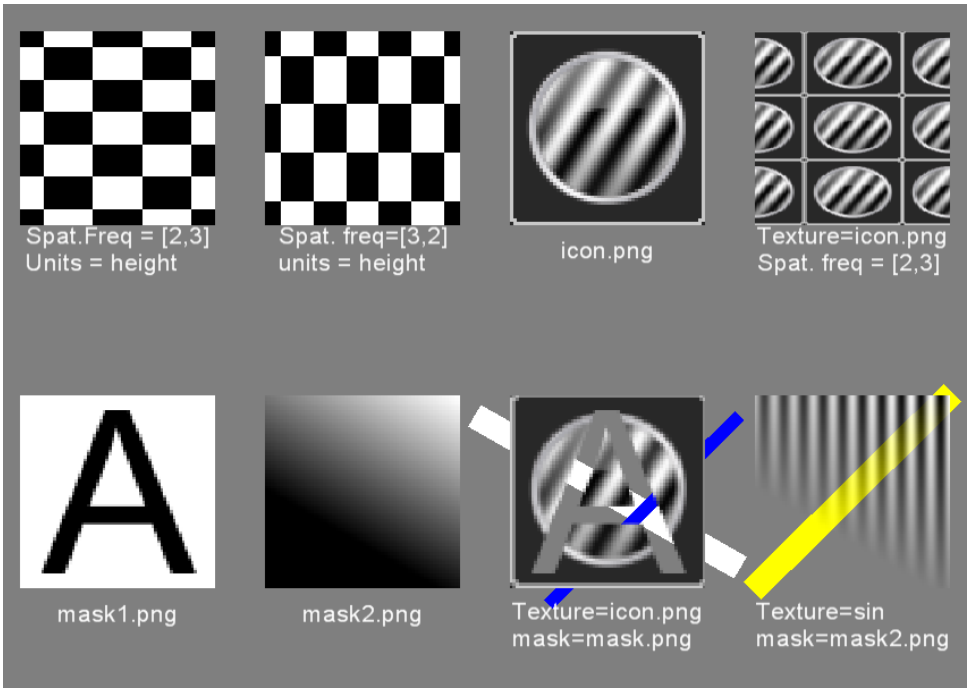


図 4.25　**[空間周波数 $]** に 2 つの値を指定する例と (上段の左側 2 つ)、 **[テクスチャ]** に画像ファイルを指定する例 (上段の右側 2 つ)、 **[マスク]** に画像ファイルを指定する例 (下段)。

[テクスチャ] および **[マスク]** には、画像ファイルを指定することもできます。 図 4.25 右上は、PsychoPy の

アイコン画像を保存した icon.png というファイルを **[テクスチャ]** に指定した例を示しています。 **[テクスチャの解像度 $]** の値が 2 の冪乗 (64 や 128 など) であったように、PsychoPy 内部ではテクスチャは縦横の解像度が 2 の冪乗の正方形でなければいけません。それ以外の解像度の画像を指定すると PsychoPy が内部的に拡大縮小を行いますので、正確さを期する場合は最初から 2 の冪乗の正方形の画像ファイルを作成することが大切です。 **[マスク]** にモノクロの画像ファイルを指定すると、 **[マスク]** 画像のピクセルの明るさがテクスチャの透明度となります。文章では説明しにくいので 図 4.25 下段をご覧ください。mask1.png、mask2.png という 2 種類のモノクロ画像をマスクとして用意しました。 **[マスク]** に mask1.png を、 **[テクスチャ]** に先ほどの icon.png を指定した例が 図 4.25 下段左から 3 番目です。mask1.png の黒色の部分が透明になって、背景の灰色が見えています。透明になっていることがわかりやすいように、白色と青色の Polygon コンポーネントを背後に配置しています。 図 4.25 下段の右端は、 **[テクスチャ]** を sin にして、 **[マスク]** に mask2.png を指定した例を示しています。mask2.png が黒色から白色に向かって滑らかに変化しているので、縞模様が滑らかに透明から不透明へと変化しています。

4.11.2 ループの [試行を繰り返す] プロパティについて

ループの **[試行を繰り返す]** プロパティは、実験設定ダイアログで「xlsx 形式のデータを保存」または「CSV 形式のデータを保存 (summaries)」をチェックして出力されるデータファイルの形式を設定するものです。

「xlsx 形式のデータを保存」をチェックして exp04c.psyexp を実行することによって作成される xlsx ファイルの例を 図 4.26 に示します。ただし、 **[試行を繰り返す]** はチェックしていないものとします。

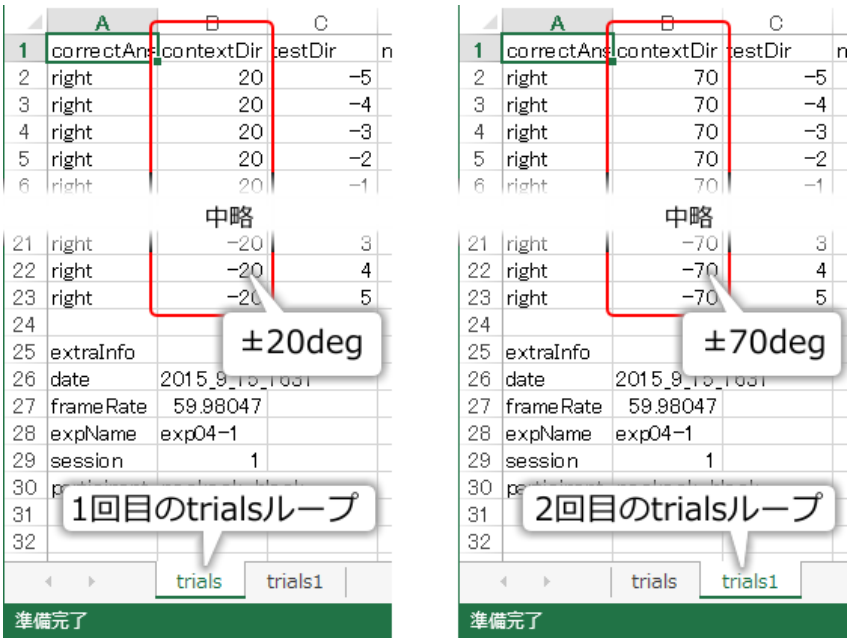


図 4.26　多重ループを用いた実験によって作成される xlsx 記録ファイルの例 (**[試行を繰り返す]** をチェックしない場合)。

表 4.1　多重ループによって作成される xlsx 記録ファイルのシート名 (内側のループ名が trials の場合)

trials	最初に実行された trials ループ
trials1	2 回目に実行された trials ループ
…	…
trialsN (N は自然数)	N+1 回目に実行された trials ループ

trials と trials1 というシートが作成されていますが、これは 表 4.1 に示すようにそれぞれ trials ループの 1 回目、2 回目に対応しています。シートの内容を確認すると、 図 4.26 の例の場合は trial シートの contextDir の列が 20 または-20 なので、第 1 ブロックは-20 度／ 20 度条件であったことがわかります。同様に、trial1 シートの contextDir の列を確認すると大 2 ブロックは-70 度／ 70 度条件であったことがわかります。

今回の実験では blocks ループによる trials ループの繰り返しは 2 回のみでしたが、仮に blocks ループによる繰り返しが 20 回に及ぶ場合は trials、trials1、…、trials19 と 20 枚もシートが作成されます。あまりシート数が多くなると Excel 上での作業が大変ですので、繰り返し回数が多くなる場合は実験を分割することも検討すべきです。

続いて blocks ループの **[試行を繰り返す]** をチェックした場合に出力される xlsx 記録ファイルの例を 図 4.27 に示します。trials、trials1 というシートができるのはチェックを外した場合と同じですが、それに加えて blocks というシートが含まれています。このシートには、blocks ループの繰り返しにおいて、exp04blocks.xlsx から読み込んだ条件のどの行が使われたかが記録されています。別にこのシートが存在していてはいけないわけではないのですが、trials、trials1 のシートから読み取ることができる情報しか含まれていないので、無駄なシートといえます。今回のように blocks ループが一番外側のループだったら 1 枚余分なシートが増えるだけですが、さらに外側にループが組まれている実験では何枚も無駄なシートが作成されてしまいます。このような場合は **[試行を繰り返す]** のチェックを外してシート数を抑えることが有効です。

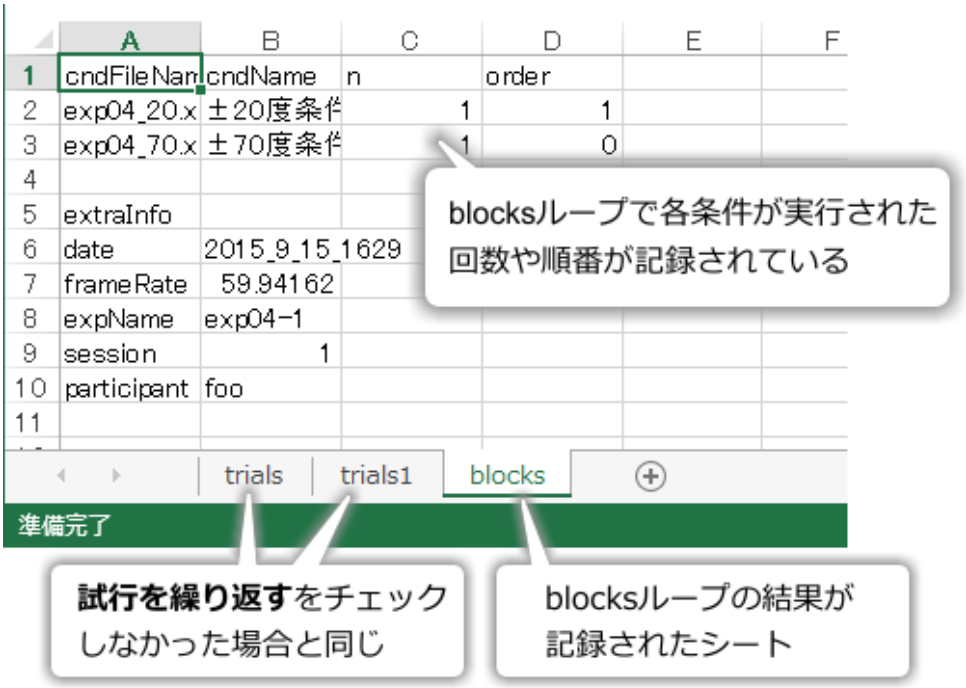


図 4.27　多重ループを用いた実験によって作成される xlsx 記録ファイルの例 (**[試行を繰り返す]** をチェックした場合)。

なお、外側のループ内にループに含まれないルーチンが存在していて、そのルーチンでの刺激や反応を記録する必要がある場合は、**[試行を繰り返す]** のチェックを外してはいけません。具体的には、 図 4.28 の test ルーチンのようなものが blocks ループ内にある場合です。このような場合に blocks ループの **[試行を繰り返す]** のチェックを外してしまうと、test ルーチンで用いられた条件や、test ルーチンにおける参加者の反応などが記録されません。チェックを外していいか自信がない場合、xlsx 記録ファイルのシート数を気にしないのであれば **[試行を繰り返す]** のチェックはつけたままにしておくというのも一つの手だと思います。

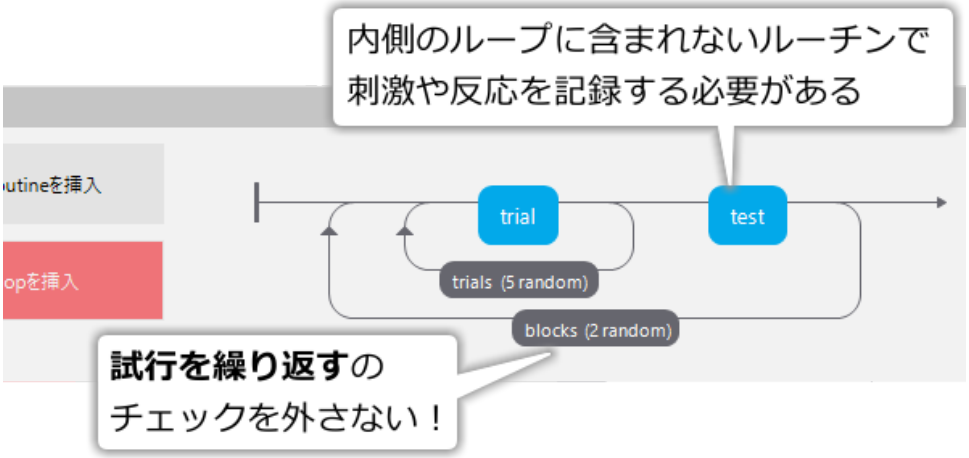


図 4.28　**[試行を繰り返す]** のチェックを外してはいけない例

4.11.3 shelf.json および Counterbalance コンポーネントに関する補足 (上級)

shelf.json は拡張子が示す通り JSON 形式のテキストファイルです。テキストエディタで簡単に内容を確認、編集することができます。今回の実験で作成される shelf.json の内容は以下のようになります。

```
{
 "counterbalance": {
 "_reps": 1,
 "70\u5ea6\u6761\u4ef6\u304b\u3089": 3,
 "20\u5ea6\u6761\u4ef6\u304b\u3089": 1
 }
}
```

2 行目の"counterbalance"は Counterbalance ルーチンの **[名前]** に対応しています。3 行目の_reps は残りの繰り返し回数です。4 行目と 5 行目は日本語の文字が Unicode エスケープシーケンスで表記されていますが、条件ファイルで指定されているグループ名 (「70 度条件から」と「20 度条件から」) と残りスロット数を表しています。通常は shelf.json の内容をエディタで開いて確認する必要はありませんが、いざというときにエディタで開く可能性も考慮するならグループ名は半角英数文字でつけた方が読みやすいでしょう。

少しわかりにくいのですが、「残り繰り返し回数」はすべてのスロットが使いつくされて次の繰り返しに入る時に 1 減少します。つまり、Code コンポーネントの **[繰り返し回数]** が 3 に設定されているなら、1 回目の繰り返しでのすべてのスロットが使いつくされた直後に shelf.json を確認すると、_reps はまだ 3 のままです (全グループのスロットは 0 になっている)。次に 1 回実験を実行した時に初めて_reps が 2 になります。3 回目の繰り返しのすべてのスロットを使いつくした後には、_reps が 1(0 ではなく) で全グループのスロットは 0 に

なっています。通常、shelf.json を直接編集する必要はないはずですが、何らかの理由で編集しなければならなくなった時は注意してください。

[グループの定義...] で「グループ数」にすると、**[グループ数]** と **[グループごとのスロット]** でグループを定義します。グループ名は自動的に 0 からグループ数-1 までの数字が割り当てられます (データ型は int ではなく str なのでご注意ください)。各グループのスロット数は全て **[グループごとのスロット]** の値となります。このように、「グループ数」による定義ではグループ名を自由につけたりグループごとに異なるスロット数を割り当てることができません。また、本文の blockFile のように付随するパラメータを定義することもできません。その代わりに、(1)xlsx ファイルを必要としない、(2) オンライン実験でも使用できるという利点があります。xlsx ファイルを使う方法はオンライン実験に対応していませんので、オンライン実験で Counterbalance コンポーネントを使用するには「グループ数」による定義しか選択肢がありません。

「グループ数」による定義では付随するパラメータを定義できないので、本章のように読み込ませたい条件ファイル名を変化させたい場合は、グループ名から条件ファイル名を得るコードを書く必要があります。グループ名は counterbalance オブジェクトの group というデータ属性で直接参照することができます。Counterbalance ルーチンの **[名前]** が counterbalance なら counterbalance.group です。これを利用して以下のようなコードを Code コンポーネントを使って挿入すればよいでしょう。

```
if ((counterbalance.group === "0")) {
    blockFile = "exp04_20_70.xlsx";
} else {
    blockFile = "exp04_70_20.xlsx";
}
```

第 5 章

Python コードを書いてみよう―視覚の空間周波数特性

5.1 この章の実験の概要

この章では、視覚の時空間特性の計測実験を題材として、Builder による実験に Python コードを組み込む方法を解説します。グラフィカルユーザーインターフェース (GUI) で実験を作成できる事が Builder の長所ですが、正直なところ本格的な実験をしようとすると「あれができない、これができない」という事だらけで行き詰ってしまいます。しかし、Builder に Python のコードを組み合わせることによって、飛躍的に「できる事」の幅が広がります。プログラミングの経験がない方には最初かなり難しく感じられるかもしれませんが、実際に実験を動かしながら少しずつ理解を深めてください。

この章の実験では、視覚の空間周波数特性を取り上げます。「空間周波数」という用語は 第 4 章 で紹介しましたね。皆さんは、視力検査であまりにも小さな視標は知覚できない事を経験していると思います。視標が小さいということは狭い範囲内で明暗が大きく変化するということですから、「空間周波数」という用語を使えば「空間周波数が高すぎる視覚刺激は私たちには知覚できない」と表現できます。視力検査では高空間周波数の刺激がどこまで知覚できるかしか調べませんが、実は空間周波数が低すぎる刺激に対しても私たちの視覚の感度が低下することがわかっています。この章では、空間周波数の変化に応じて私たちの視覚の感度が変化することを確認する実験を作成します。便宜上「実験」と呼んでいますが、ベースになっているのは筆者が以前に知人から紹介してもらったゲーム風のデモです。正確な感度の計測に適した手続きではありませんが、Python のコードを Builder に組み込む最初の一歩としておもしろい手続きなので取り上げることにしました。

なお、この実験では刺激の視角が非常に重要な意味を持ちますので、**[空間の単位]** に deg を使用します。モニターの観察距離の設定などをしっかりおこなっておいてください (第 2 章 参照)。

図 5.1 に使用する刺激を示します。スクリーン中央に直径 0.1deg の白色の小さな円を提示します。実験中、実験参加者はこの刺激を固視し続けないといけません。この刺激を固視点と呼ぶことにします。固視点から視角で 5.0deg 離れた位置に、直径 4.0deg の時計回りまたは反時計回りに 15 度傾いたグレーティング刺激を 1.0 秒提示します。この刺激をターゲットと呼ぶことにします。ターゲットの出現方向は固視点の右を 0 度として反時計回りに 0 度、45 度、90 度、135 度、180 度、225 度、270 度、315 度の 8 方向の中から無作為に選びます。グレーティングの空間周波数は 0.2、0.4、0.8、1.6、3.2、6.4、12.8 cpd (cyecle per degree: 視角 1 度あたりの繰り返し回数) の 7 種類を用います。

図 5.2 に実験の手続きを示します。試行開始時には、スクリーン中央に固視点のみが提示されています。実験

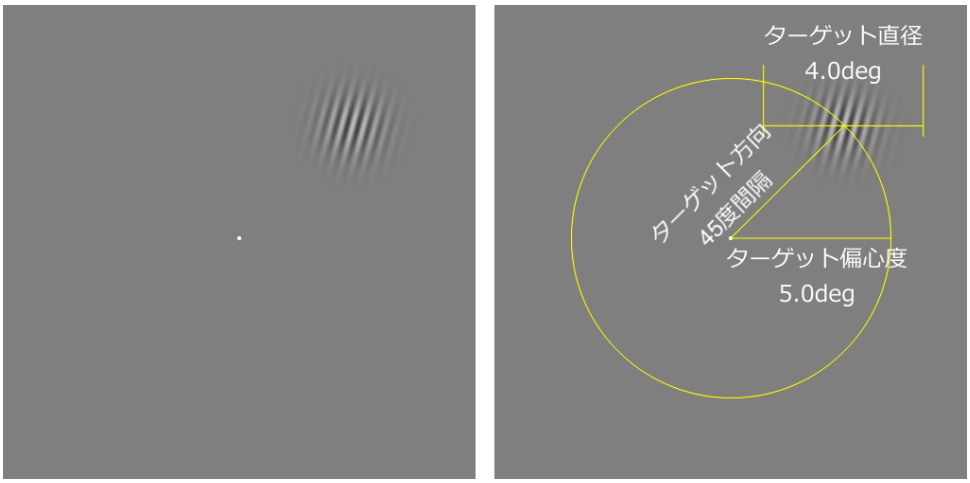


図 5.1　実験に使用する刺激。

参加者がカーソルキーの左右どちらかを押すと、1 秒後にターゲットがいずれかの位置に出現します。ターゲットの色はキーが押された直後には 0, 0, 0、すなわち背景と全く同一で知覚することができませんが、6 秒間かけて一定の速度で 1, 1, 1 まで変化します。視覚刺激の明暗差のことをコントラストと言いますが、この用語を使えばターゲットのコントラストが上昇していくと表現できます。実験参加者は、ターゲットがどちらに傾いているかを知覚できたらできるだけ速く、反時計回りであればカーソルキーの左、時計回りであれば右のキーを押します。キーが押されるとターゲットは消えて、直ちに次の試行に進みます。ターゲット出現後 6 秒経過しても反応がなかった試行はエラーとして記録して、やはり直ちに次の試行に進みます。時間と共にターゲットのコントラストが上昇するので、実験参加者の反応時間が早いほど低いコントラストでターゲットを知覚できたと考えられます。以上の手続きを、すべてのターゲット方向 (8 種類) とターゲット空間周波数 (7 種類) の組み合わせに対して反時計回りを 1 回、時計回りを 1 回、合計 $8 \times 7 \times 2 \times 2 = 112$ 試行行います。途中に休憩を挟まずに一気に実行し、すべての試行が終了すれば実験は終わりです。

最初に述べたとおり、この手続きは正確な感度を計測することよりも、簡単な手続きで空間周波数による感度の違いを感じるためのデモと言った方が適切です。この手続きでは試行数が少なすぎますし、なにより参加者の反応時間からターゲットを検出できる閾値のコントラストを測定しようという発想自体が正確な測定に適していません。というのも、反応時間には実際にターゲットが知覚できてからキーを押すまでの時間などが含まれて、その間にもターゲットのコントラストは上昇し続けているからです。正確さを期するのであれば 第 4 章 のような恒常法の手続きを用いるべきです。この章の実験は、あくまで Builder の実験に Python のコードを組み込む方法を学習するための例題だと考えてください。

さて、まずは 第 4 章 までに解説済みの作業でできるところまで実験を作成しましょう。

準備作業

- この章の実験のためのフォルダを作成して、その中に exp05.psyexp という名前で新しい実験を保存する。
- 実験設定ダイアログの「スクリーン」タブの **[単位]** を deg にする。モニターの設定 (観察距離、モニターの幅長および解像度) をまだ設定していない場合は 第 2 章 を参考にモニターセンターを開いて設定する。
- 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。

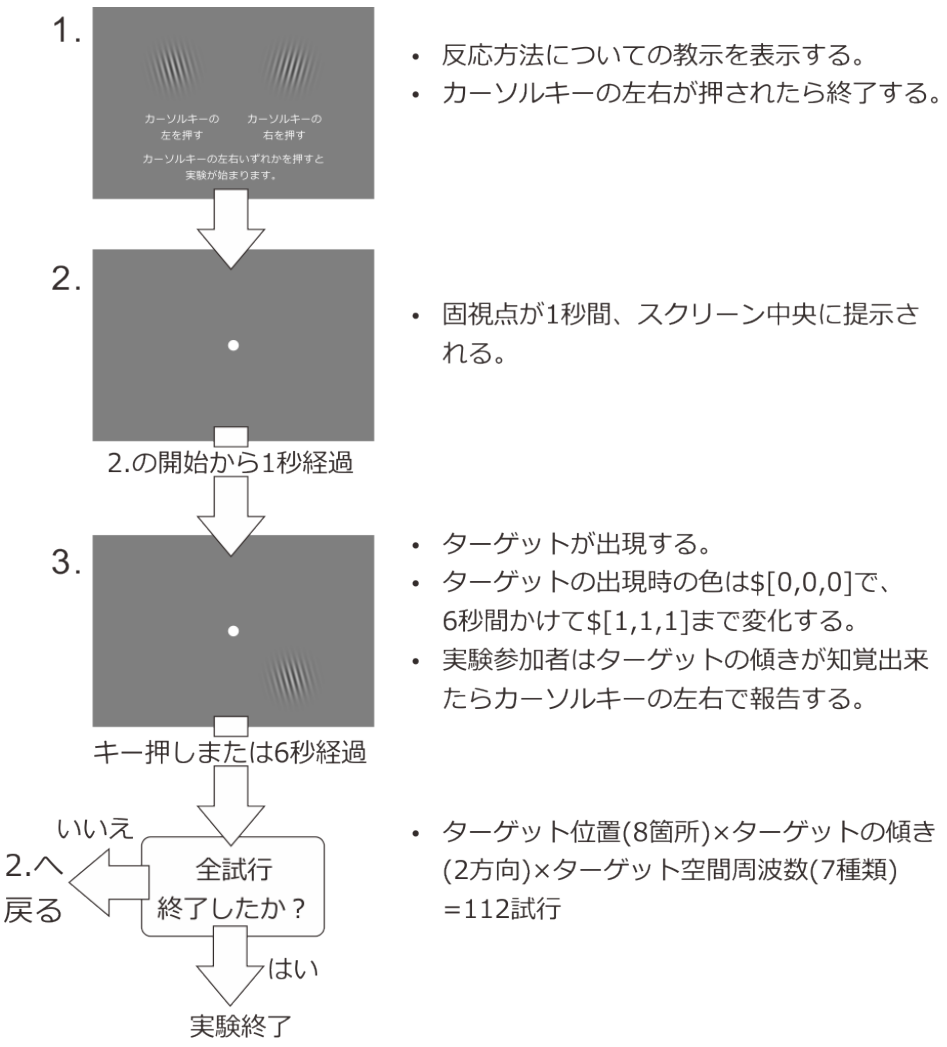


図 5.2　実験の手続き。

- trial ルーチン
  - Grating コンポーネントをひとつ配置し、以下のように設定する。
    - 「基本」タブの **[名前]** を `targetStim` に、 **[終了]** を実行時間 (秒) で `6.0` にする。
    - 「レイアウト」タブの **[サイズ [w, h] $]** を `(4.0, 4.0)` にする。 **[回転角度 $]** に `targetDir` と入力して、「繰り返し毎に更新」にする。
    - 「テクスチャ」タブの **[マスク]** を `gauss` に、 **[テクスチャの解像度 $]** を 512 にする。 **[空間周波数 $]** を `targetSF` にして、「繰り返し毎に更新」にする。
  - Keyboard コンポーネントをひとつ配置し、以下のように設定する。
    - 「基本」タブの **[終了]** を実行時間 (秒) で `6.0` にする。 **[検出するキー $]** を `'left', 'right'` にする。
    - 「データ」タブの **[正答を記録]** をチェックして、 **[正答]** に `$correctAns` と入力する。
  - Polygon コンポーネントをひとつ配置し、以下のように設定する。

    * 「基本」タブの **[名前]** を `fixation` 、 **[終了]** を実行時間 (秒) で `6.0` にする。 **[形状]** を円にする。

    * 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.1, 0.1)` にする。

- instruction ルーチン (作成する)

  – フローの先頭に挿入する。

  – Grating コンポーネントを 1 個配置して以下のように設定する。

    * 「基本」タブの **[名前]** を `left_sample` とする。 **[終了]** を空白する。

    * 「テクスチャ」タブの **[マスク]** を `gauss` 、 **[空間周波数 $]** を `1.0` 、 **[テクスチャの解像度 $]** を 512 にする。

    * 「レイアウト」タブの **[サイズ [w, h] $]** に `(0.4, 0.4)` と入力し、**[位置 [x, y] $]** に `(-5.0, 0.0)` と入力する。 **[回転角度 $]** を `-10.0` にする。

  ~~left_sample~~ **をコピーして** `right_sample` **という名前で貼り付け、以下のように設定する。**

    * 「レイアウト」タブの **[位置 [x, y] $]** を `(5.0, 0.0)` とする。 **[回転角度 $]** を `10.0` にする。

  – Text コンポーネントを 1 個配置して、以下のように設定する。

    * 「基本」タブの **[名前]** を `left_label` にする。 **[終了]** を空白する。 **[文字列]** に **反時計回り** と入力する。

    * 「書式」タブの **[文字の高さ $]** を適当な値 (`0.5` など) にする。

    * 「レイアウト」タブの **[位置 [x, y] $]** を `(-5.0, -5.0)` など、left_sample の少し下に配置する。各自のスクリーンサイズに応じて調節すること。

  — left_label をコピーして `right_label` という名前で貼り付け、以下のように設定する。

  – 「基本」タブの **[文字列]** に **時計回り** と入力する。

  – 「レイアウト」タブの **[位置 [x, y] $]** の X 座標を `5.0` にする (つまり right_sample の少し下になるようにする)。

  – もう一度コンポーネントの貼り付け操作をする。すでにコピー済みの left_label が貼り付けられるはずである。名前は `text_inst` とする。

    * 「基本」タブの **[文字列]** に **反時計回りならばカーソルキーの左、時計回りならば右をできるだけ速く押して反応してください** といった内容の教示文を入力する。各自のスクリーンサイズや文字サイズに応じて改行するなどして画面に入るようにすること。

    * 「レイアウト」タブの **[位置 [x, y] $]** を `(0.0, -6.0)` など、スクリーン中央で left_label, right_label よりやや下に、文字列が重ならないように配置する。 **[文字列]** の教示文に改行が含まれていると少し下に配置しないと重なるので注意すること。

  – Keyboard コンポーネントをひとつ配置し、「基本」タブの **[終了]** を空白に、 **[検出するキー $]** を `'left', 'right'` にする。「データ」タブの **[記録]** を「なし」にする。

- blank ルーチン (作成する)
  - フローの instruction ルーチンと trial ルーチンの間に挿入する。
  - trial ルーチンに配置してある fixation をコピーして貼り付ける。名前はそのまま (`fixation_2`) でよい。「基本」タブの **[終了]** を「実行時間 (秒)」で `1.0` にする。
- trials ループ (作成する)
  - blank ルーチンと trial ルーチンをまとめて繰り返すように挿入する。
  - **[繰り返し回数 $]** の値を 1 にする。
  - **[条件]** に `exp05cnd.xlsx` と入力する。
- exp05cnd.xlsx (条件ファイル)
  - targetSF、targetDir、correctAns、targetPos の 4 パラメータを設定する。
  - 実験手続きの説明を満たすように targetSF に 7 種類の空間周波数、targetDir に 2 種類の傾きの値を入力する。targetDir と対応するように correctAns に値を入力する (targetDir が-15 の行は'left'、15 の列は'right')。targetPos はとりあえず空白にしておく。この時点でパラメータ名の行を除いて 14 行となる。

blank というルーチンではただ 1 秒間固視点を提示しているだけです。これは実験手続きの「1 秒間固視点が提示される」という点を実現するために挿入されているのですが、blank ルーチンが無くても trial ルーチンで targetStim の **[開始]** を fixation の **[開始]** から 1 秒遅らせれば実現できます (第 3 章 参照)。しかし、このテクニックを使うと後でコントラストを時間の経過とともに上昇させる処理が少し複雑になってしまいますので、今回は固視点の提示のために独立したルーチンを使用しています。

これで準備が整いました。いよいよ Python のコードを Builder に追加しますが、その前に用語の説明をしておきましょう。退屈かもしれませんが、用語を覚えておかないと後の解説がわからないのでしっかり覚えてください。

5.2 変数、データ型、関数といった用語を覚えよう

第 3 章 で Python における「名前」の話が出てきたことを思い出してください。刺激に stimulus、刺激色を表すパラメータに stim_color という「名前」を付けて、「stimulus の **[前景色]** プロパティに stim_color の値を代入する」とかいったことをしたのでした。この時に敢えて触れなかったのですが、この「名前」は一体何の名前なのでしょうか。「えっ、今『刺激』や『パラメータ』に名前を付けたって言ったじゃないか」と言われるかも知れません。確かにその通りなのですが、より正確に言うと、これらのものを収納しておく「箱」に名前をつけたのです。stimulus という名前の「箱」に Polygon コンポーネントを収納したり、stim_color という名前の「箱」に色の値を収納したりしていたのです。この名前は「箱」の名前なのですから、当然中身を入れ替えることもできます。第 3 章 の実験でループの繰り返しの度に刺激色が変化していたのは、Builder が毎回 stim_color という箱の中におさめられた値を変更してくれていたからです (図 5.3)。この箱のことを変数と呼びます。今まで「名前」と呼んでいたものは変数の名前、すなわち変数名です。

Python では、変数の中にさまざまなものを収納することができます。C 言語のように変数に「型」がある言語では、整数値を入れる変数には整数値のみ収納できるといった制限がありますが、Python にはそのような制限がありません。Python で扱える型のデータであればすべて収納できます。プログラミングを学んだ経験が

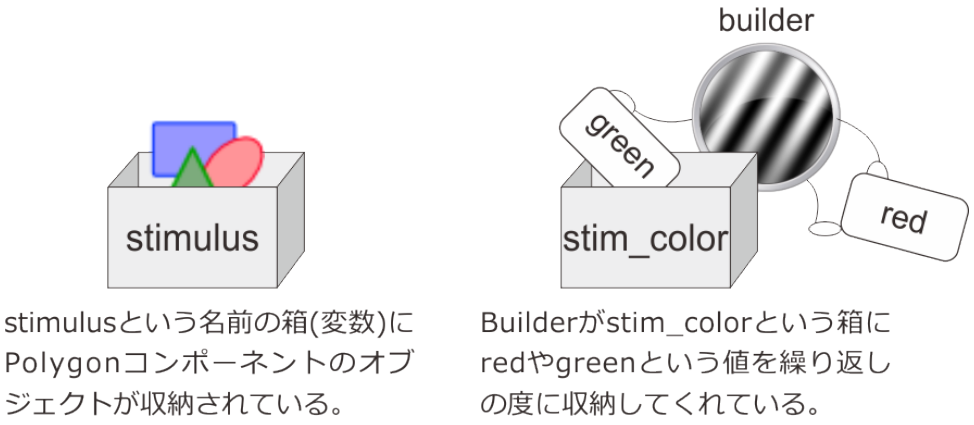


図 5.3　変数とはいろいろなものを収納しておける箱のようなもの。

ない方は「データの型ってなんだ？」と思われるかもしれませんね。ほとんどのプログラミング言語では、扱うことができるデータにいくつかの「型」があって、型によって適用できる処理が異なります。表 5.1 に基本的な Python の型を示します。ここでそれぞれの型を詳しく説明し出すとなかなか実験の作成にたどり着きませんので、ごく簡単な説明に止めています。ここで注目していただきたいのは「シーケンス」という型です。他のプログラミング言語を学んだことがある方は「配列」という名前の方がピンと来るかもしれません。第 2 章 で刺激の位置や大きさを指定したりする時に「両端の括弧は意味があるので忘れずに入力してください」と書きましたが、実はこれはシーケンスの一種である「リスト」を書くときの Python の文法なのです。[1.0, 0.0, 0.5] と書かれていたら、Python はこれをリストとして解釈して「最初の要素は 1.0、その次は 0.0、最後は 0.5」という具合にそれぞれの値を取り出すことができます。[ ] が欠けているとリストとして解釈することができないのでエラーとなるわけです。

表 5.1　Python の基本的なデータ型。ここでは最小限の説明に止めます。

型	概要
数値	単一の数値。Python 内部では整数のみを扱う型 (整数型) と小数点つきの数値を扱う型 (浮動小数点型) がある。
文字列	アルファベットやかな文字、漢字、各種記号等の文字が連なったもの。
シーケンス (リスト、タプル)	複数のデータを順番に並べてひとまとめにしたもの。「2 番目の要素を取り出す」といった具合に位置を指定して内容を取り出すことができる。カンマで区切られた要素を [ ] で囲んだものをリスト、( ) で囲んだものをタプルと呼ぶ (両者の違いについては「 9.14.3: リストとタプル 」 参照)。
辞書	複数のデータをひとまとめにして、キーと呼ばれる値を用いて内容を取り出せるようにしたもの。第 4 章 で実験情報ダイアログの入力値を保持していた expInfo という変数に格納されているデータがこれに該当する。

データ型の詳細については必要に応じて解説することにして、最後に関数という用語に触れておきましょう。関数と聞くと、数学の授業で習った「y は x の関数」とか「2 次関数のグラフ」といった内容を思い出される方も多いと思います。例えば「z は x と y の関数」と言った場合、x と y の値が与えられたら、その関数で定められた計算を行えば対応する z の値を得ることができます (図 5.4 左)。Python の関数とはこの関係を一般

化したようなものです。例えば、「ファイルを保存する」という関数があるとします。この関数にファイル名と保存したいデータを与えると、ファイルを作成したりデータを書きこんだりといった処理が行われて、保存が成功したか否かを示す値が得られるとします (図 5.4 右)。ずいぶん数学で習った関数と違うような感じがするかもしれませんが、「値を入力すると定められた処理が行われて結果が出力される」という構図はよく似ています。このような働きを持つものを Python では関数と呼びます。関数に入力する値のことを引数、出力のことを戻り値と呼びます。実は関数にもいろいろな種類があるのですが、それについても今後必要に応じて説明することにして、関数、引数、戻り値という用語を覚えておいてください。

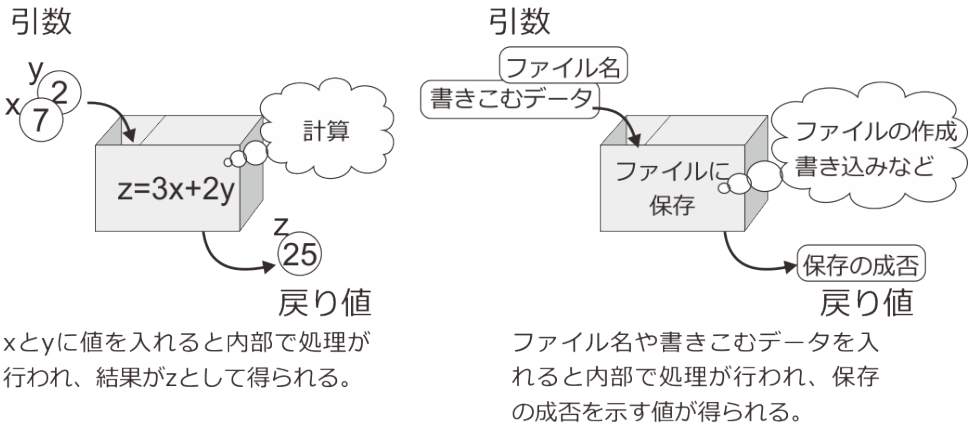


図 5.4　数学で習った関数と Python における関数。

チェックリスト

- 変数の役割を説明できる。
- 関数の役割を、引数、戻り値という用語を用いながら説明できる。

5.3 数学関数を利用して刺激の位置を指定しよう

退屈な用語の説明はいったん切り上げて実験の作成に戻りましょう。まず、Builder で exp05.psyexp を開いて、trial ルーチンの targetStim のプロパティ設定ウィンドウを開いてください。先ほどの作業では、まだ targetStim の **[位置 [x, y] $]** が初期値のままに残されていました。刺激の位置は固視点から半径 5.0deg の円周上で、右方向を 0 度として 0 度から 45 度間隔で 8 方向の中から無作為に選ぶのでした。試行毎に変化するのですから条件ファイルにこれら 8 種類の位置を書きこまなければいけないのですが、45 度や 135 度の方向の x 座標、y 座標の値を書きこむのが少々面倒です。半径が 5.0 ですから、45 度の方向は x 座標、y 座標とも 5.0 × 1/ $\sqrt{2}$ =3.53553…です。幸い 135 度や 225 度の方向は符号を変えたらいいだけですのでこの程度なら手入力してもいいのですが、後々半径を変更したくなったり 45 度間隔の代わりに 60 度間隔にしたくなったりした時に面倒です。なにより実験記録ファイルを見たときに刺激位置のパラメータが 3.53553, 3.53553 と書かれているよりも 45 と書かれていた方が圧倒的にわかりやすいでしょう。そこで、さっそく関数を用いて条件ファイルに 45 とか 135 とか書けば Builder が座標値を計算してくれるように改造してみましょう。

表 5.2　Builder で使用できる数学関数。min と max は任意の個数の引数を受け取ることができます。

sin(x)	正弦関数	min(x,y,z…)	x, y, z,…の最小値
cos(x)	余弦関数	max(x,y,z…)	x, y, z,…の最大値
tan(x)	正接関数	average(x)	x の平均値
log(x)	自然対数関数	std(x)	x の標準偏差
log10(x)	常用対数関数	deg2rad(x)	x を度からラジアンに変換
pi	円周率	rad2deg(x)	x をラジアンから度に変換
sqrt(x)	平方根		

視覚刺激を用いた知覚や認知の実験では、三角関数や指数関数の数学関数や円周率などの定数はしばしば用いられるので、Builder では 表 5.2 に示す関数が用意されています。今「三角関数や指数関数」と書いたばかりなのに指数関数がないじゃないか、と思われるかも知れませんが、これ以外の数学関数を使用する方法については「 5.8.1: 他の数学関数を使用する方法 」を参照してください。さて今回の目的の場合、条件ファイルに書かれた角度から座標値を計算するのですから、deg2rad( ) と sin( )、cos( ) が役に立ちそうです。まずは 45 度の sin を計算する式を作るところから始めてみましょう。数学で x の関数 f(x) に対して x に 3 を代入する時に f(3) と書いたように、deg2rad( ) を用いて 45 度をラジアンに変換するには deg2rad(45) と書きます。座標値を計算するにはラジアンに変換した値を sin( ) および cos( ) に代入する必要がありますが、Python を含む多くのプログラミング言語では、関数の引数に他の関数 (の戻り値) を用いることが許されています。このテクニックを使うと、sin(deg2rad(45)) と書けば「まず deg2rad(45) を計算して、その結果を sin( ) に代入する」という計算ができます (図 5.5)。同様に、45 度の cos は cos(deg2rad(45)) という式で計算ができます。

図 5.5　関数の引数に関数を書くと Python が順番に計算をします。

これで 45 度の sin と cos を計算する式ができましたが、目的の座標値を得るためにはこれらの式に半径の値、5.0 を掛けなければいけません。数値同士の足し算や掛け算を行うには、算術演算子と呼ばれる記号を用います (表 5.3)。また難しそうな用語が出てきたと思われるかもしれませんが、要するに + とか-とかいった記号のことです。昔のタイプライターで×や÷といった記号を使えなかった名残で、掛け算は * 、割り算は / と書くことに決まっています。この算術演算子を利用すれば、ようやく 45 度の時の座標値を計算する式を完成させることができます。 **[位置 [x, y] $]** に書くときには x 座標、y 座標の値をこの順番に並べたリストにしなければいけないことに注意すると、以下の式が得られます。

```
[5*cos(deg2rad(45)), 5*sin(deg2rad(45))]
```

表 5.3　Python の算術演算子

x + y	x 足す y	-x	x の符号を反転
x - y	x 引く y	x % y	X を y で割った余り
x * y	x 掛ける y	x ** y	X の y 乗
x / y	x 割る y	x // y	x 割る y(割り切れない場合は切り上げ)

これでターゲットの方向が 45 度の時の式が完成しましたが、今回の実験では条件ファイルから角度を読み込んで座標値を計算しないといけません。幸い、関数の引数には変数を書くことができるので、単に先ほどの式の 45 を変数名に書き換えるだけでこの目標は達成できます。先ほど条件ファイルを作成した時に空白のまま残しておいた targetPos を利用することにしましょう。Builder を開いて trial ルーチンの targetStim の **[位置 [x, y] $]** に以下のように入力し、「繰り返し毎に更新」にしてください。

```
[5*cos(deg2rad(targetPos)), 5*sin(deg2rad(targetPos))]
```

Builder での変更を終えたら条件ファイル exp05cnd.xlsx を開いてください。すでに 14 行分の条件が入力されていますが、これらの 14 条件を 8 種類 (0, 45, 90, 135, 180, 225, 270, 315) の targetPos すべてに対して実行するように変更してください。最終的に 14 × 8=112 行の条件ファイル (パラメータ名の行を除く) になるはずです。

以上で刺激の位置を試行毎に決定することができるようになりました。あとは時間の経過とともにだんだんコントラストが高くなる刺激を実現できれば実験は完成します。また新しい話が出て来るので、ここでいったん解説を区切ることにしましょう。

チェックリスト

- Builder で使用できる数学関数を挙げることができる
- 関数の引数に別の関数の戻り値を使うことができる。
- 条件ファイルから読み込んだパラメータ値を関数の引数に使うことができる。
- Python の算術演算子 8 つ挙げてその働きを説明することができる。

5.4 ルーチン開始後の時刻を取得して刺激を変化させよう

時間の経過とともに刺激のコントラストを上昇させるには、「現在の時刻に応じて **[前景色]** の値を変更する」ことと、「**[前景色]** の変化を刺激に反映させる」ことが必要です。第一の点については、t という Builder の内部変数を利用します。内部変数とは Builder が正常に動作するためにユーザーから見えない内部で自動的に作成する変数で、t には現在のルーチンが開始されてから何秒経過したかが保持されています。第二の点については、各プロパティの値を入力する欄の右側のプルダウンメニューで「フレーム毎に更新する」を選択することで実現できます。ごくシンプルな実験を作成してこれらのことを確認してみましょう。

一旦 exp05.psyexp を保存して、新規に実験を作成してください。trials ルーチンに Text コンポーネントをひとつ配置して、Text コンポーネントの **[終了]** を「実行時間 (秒)」にして値に 10.0 と入力して **[文字列]** に $t と入力してください。そして、 **[文字列]** の右端のプルダウンメニューを「フレーム毎に更新」にしましょう (図 5.6)。これだけで準備は OK ですので、適当な名前で保存して実行してみてください。スクリーン中央に、ものすごい速さで 0.0 から 10.0 に向かって上昇していく数値が表示されるはずです。試しに「フレーム毎に更新する」を今までの章で使ってきた「繰り返し毎に更新」に変更して、0 から全く値が変化しないことを確認しておいてください。なお、小数点以下 2 桁目まで表示したいなど、小数の表示を変更する方法は「 12.6: 残り時間を表示しよう 」をご覧ください。

一般論として、リアルタイムに刺激の色や位置といったプロパティを変更すると、PC のグラフィック機能への負担が大きくなります。ですから「更新しない」や「繰り返し毎に更新」に設定された刺激については、Builder はルーチン開始時に刺激を作成した後、一切プロパティ値を変更しません。近年の高性能な PC ならば、刺激数が数百、数千個でもない限り全て「フレーム毎に更新する」にしてもほとんど問題なく処理できるでしょうが、実験中は少しでも PC に無用な処理はさせないようにして PC の処理遅れなどを防ぎたいところです。実験の性質上どうしてもリアルタイムに値を変化させないといけないプロパティに関してだけ「フレーム毎に更新する」を設定するようにしましょう。

Builder の内部変数 t と「フレーム毎に更新する」を用いれば、時間と共にコントラストを変化させることは難しくありません。Builder で exp05.psyexp を開いて trial ルーチンの targetStim のプロパティ設定ダイアログを開いてください。ルーチン開始時に **[前景色]** が 0.0, 0.0, 0.0 で、一定の速度で上昇して 6.0 秒経過した時点で 1.0, 1.0, 1.0 になればよいのですから、t/6.0, t/6.0, t/6.0 とすれば目的を達成できるはずです。忘れずに「フレーム毎に更新する」の設定もしてください。これで実験が完成しました。

実験を実行したら、targetSF の値別に反応時間の平均値を計算してみてください。ひとつの targetSF の値に対して targetDir が 2 種類、targetPos が 8 種類で 16 試行あるはずです。平均値を計算したら、横軸に targetSF、縦軸に反応時間の平均値をプロットしてみましょう。実験に使用したモニターの平均輝度や部屋の照明などによって結果は変化しますが、targetSF が 1.6 か 3.2 辺りで反応が最も速く、最小値の 0.2 や最大値の 12.8 では反応が遅くなります (図 5.7 左)。反応時間が遅いという事はそれだけターゲットのコントラストが高くならないと刺激の傾きが判断できないということですから、ターゲットの空間周波数が高すぎても低すぎても視覚の感度が低下することがわかります。縦軸を反応時間の逆数にすると、一般的な空間周波数別の感度のグラフの形状に近くなります (図 5.7 右)。

ここでこの章の内容は一区切りですが、次の話題に移る前にひとつ補足します。Builder では現在のルーチンが開始してから経過した時間を t という変数に保持していると述べましたが、同時にルーチンが開始してからのフレーム数を frameN という変数に保持しています。 **[開始]** や **[終了]** をフレーム数で指定している場合は、t よりも frameN を使用する方が便利でしょう。フレーム数による **[開始]** や **[終了]** の指定については「 2.12.6: 時刻指定における frame について 」を参考にしてください。

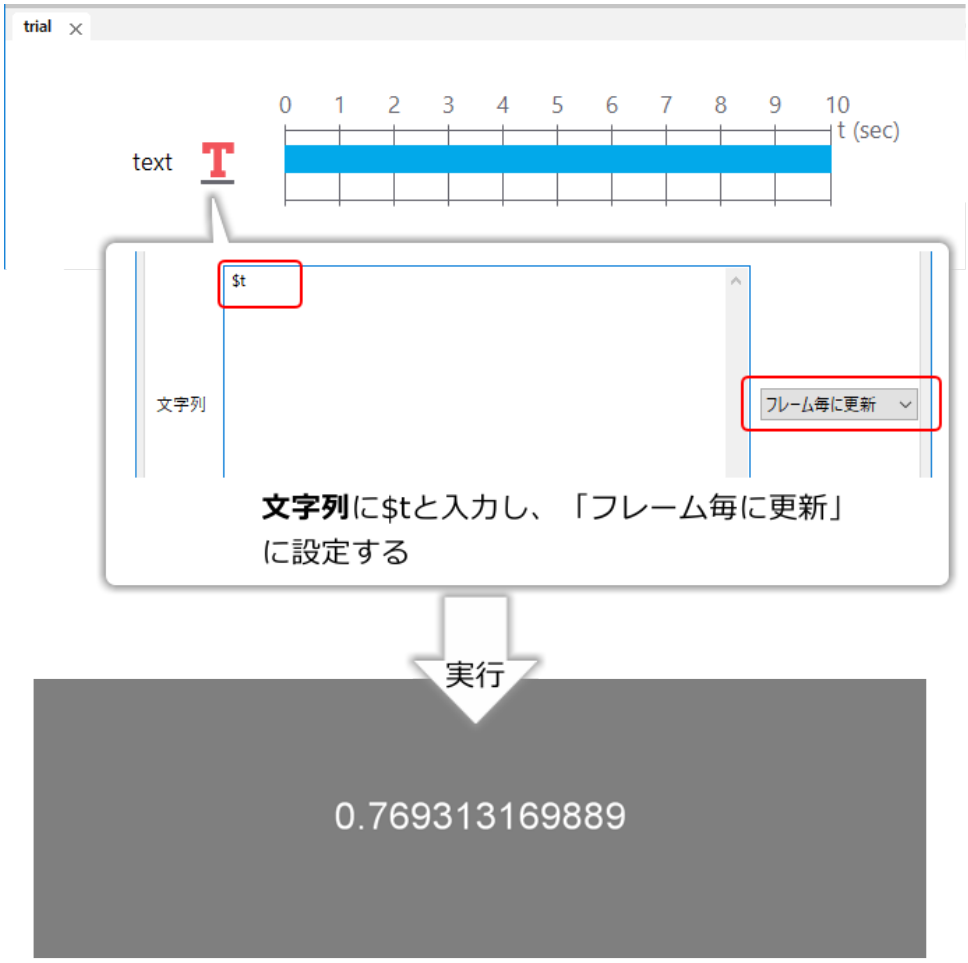


図 5.6　Builder の内部変数 t のテスト。画面上にルーチン開始から経過した時間が表示されます。

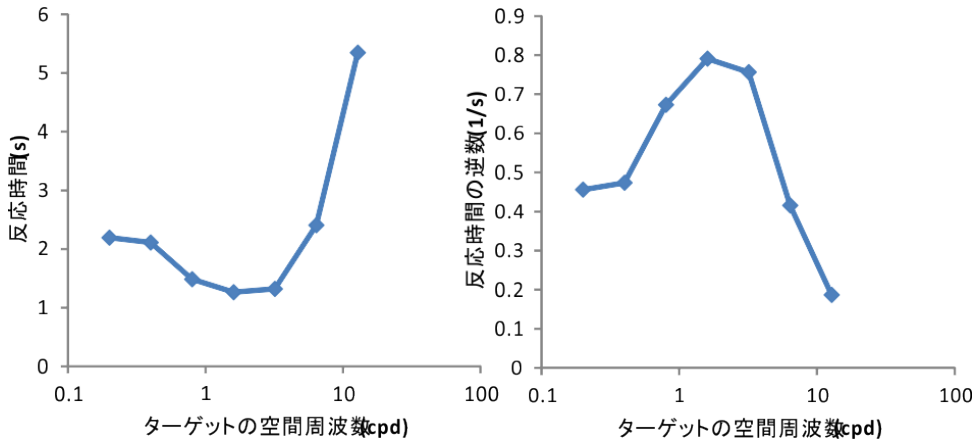


図 5.7　実験結果の例。左は反応時間、右は反応時間の逆数を縦軸にプロットしています。横軸が対数軸になっている点に注意してください。

チェックリスト

- 現在のルーチンが開始してから経過した時間を保持している Builder の内部変数数を答えられる。
- 現在のルーチンが開始してから描画したフレーム数を保持している Builder の内部変数数を答えられる。
- 現在のルーチンが開始してから経過した時間の値を用いて、時間の経過とともに色や位置が変化する実験を作成することができる。

5.5 実験情報ダイアログから実験のパラメータを取得しよう

この章で目指した実験はすでに完成していますが、応用問題ということで少し改造してみましょう。第 4 章で実験情報ダイアログから条件ファイル名を取得しましたが、同様に刺激の位置や色といったプロパティ値を取得することが可能です。ただ、実験情報ダイアログで数値を入力してプロパティ値として使う場合は少し工夫が必要です。

なぜ工夫が必要なのかを確認するために、まずは第 4 章と同様に作業してみましょう。まず、実験情報ダイアログに数値を入力するための項目を追加します。targetStim の大きさを入力する項目を eccentricity、固視点からの距離を入力する項目を target size という名前にすることにしましょう。ちなみに eccentricity とは、視覚刺激が視野中心からどれだけ外れた位置にあるかという意味です。また、target size という項目名にはスペースが入っていますが、実験情報ダイアログの項目名は Python の変数名でなくてもかまわないのでこのようにスペースが入っていてもエラーになりません。方法がわからない人は第 4 章を復習してください。

続いて targetStim のプロパティ設定ダイアログを開いて、**[位置 [x, y] $]** と **[サイズ [w, h] $]** で実験情報ダイアログの値を参照するように変更します。第 4 章の内容を思い出すと、expInfo['eccentricity'] と書けばその値が取り出せるはずです。取り出した値は **[位置 [x, y] $]** に入力済みの式の 5 に相当しますから、以下のように 5 を expInfo['eccentricity'] に書き換えればよいはずです (紙面の都合上改行していますが **[位置 [x, y] $]** に入力する際は 1 行で書いてください)。

```
[expInfo['eccentricity'] * cos(deg2rad(targetPos)),
 expInfo['eccentricity'] * sin(deg2rad(targetPos))]
```

同様に、target size の値は expInfo['target size'] と書けば取り出せます。この値は **[サイズ [w, h] $]** の 4.0 に対応しますから、**[サイズ [w, h] $]** を以下のように書き換えます。

```
[expInfo['target size'], expInfo['target size']]
```

これでうまくいくはずですが、実験を実行して実験情報ダイアログに eccentricity に 5、target size に 4.0 と今までどおりの値を入力して進めてみると、教示画面が終わってターゲットが提示される直前にエラーで停止してしまいます (図 5.8)。一般論としてこういう時にエラーメッセージを読んで何が起きたのかを考えることは上達への近道でありとても大切なのですが、ここのエラーは厄介なので結論を言うと (詳しい解説は「 5.8.3: 実験情報ダイアログに入力した数値をそのまま使用した際のエラーに関する踏みこんだ議論 (上級) 」をご覧ください)、eccentricity に入力した '5' という値が私たちの期待している 5 という数値ではなく、「5」という文字として扱われていることが原因です。意味がわからないと思われる方もいるかもしれませんが、数値としての 5 を表記する方法は何通りもあります。例えば漢字で「五」と書いてもいいですし、英語で"five"と書いてもいいでしょう。アラビア数字を使って「5」と書くというのもそれら

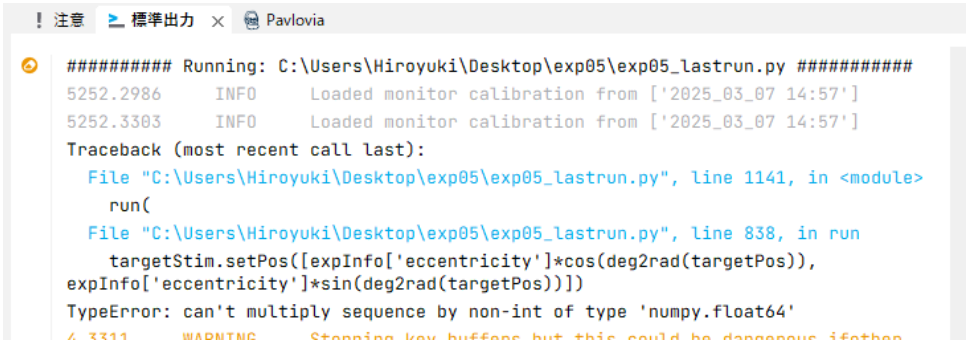


図 5.8　expInfo から取り出した値をそのまま使って刺激の位置と大きさを変更しようとするとエラーになります。PsychoPy のバージョンによってエラーメッセージが異なりますが (この画像は 2024.2.5)、エラーの原因と対策は同じです。

の方法の一つに過ぎません。実験情報ダイアログから取り出した値はすべて文字列として扱われるので、`expInfo['eccentricity'] * cos(deg2rad(targetPos))` という式を Python が実行しようとしたときに「5」という文字と cos(deg2rad(targetPos) の計算結果 (これは浮動小数点数となる) の「掛け算」をすることになってしまい、そんな計算はできないとエラーになってしまったというわけです。

エラーの原因がわかったので、次は解決策を考えないといけません。状況としては、「3.7: パラメータを利用して刺激を変化させよう」で取り上げた「パラメータに Target と入力されていた時、それは'Target' という文字列か、条件ファイルに定義されているパラメータ名か？」という問題と似ています。その時は$記号をつけることでどちらかを明示してやればよかったのでした。今回も「これは文字列ではなく数値として扱ってほしいんだ」と Builder に伝えることができれば解決するはずですが、ここで登場するのが Python のデータ型変換関数 (表 5.4) です。float( ) 関数は、引数に与えられたデータを浮動小数点数に変換して返します。データは整数型のように浮動小数点数ではない数値でも構いませんし、浮動小数点数として解釈できる文字列でも構いません。浮動小数点数として解釈不可能なデータが与えられるとエラーになります。この float( ) 関数を用いると、eccentricity の項目に入力された値を数値に変換することができます。

表 5.4　Python におけるデータ型変換関数。Builder で必要と思われるものだけを抜粋しています。

関数	概要
int(x)	x を整数に変換します。x が小数だった場合、小数点以下の値は切り捨てられます。
float(x)	x を浮動小数点数に変換します。
str(x)	x を文字列に変換します。
list(x)	x をリストに変換します。
tuple(x)	x をタプルに変換します

```
float(expInfo['eccentricity'])
```

これを **[位置 [x, y] $]** の式に当てはめると以下のようになります。長くて読みづらいですが頑張って先ほどエラーとなった式と見比べてください。

```
[float(expInfo['eccentricity']) * cos(deg2rad(targetPos)),
 float(expInfo['eccentricity']) * sin(deg2rad(targetPos))]
```

同様に、 **[サイズ [w, h] $]** の式も以下のように訂正しましょう。

```
[float(expInfo['target size']), float(expInfo['target size'])]
```

訂正が終わったら実行してください。今度こそ、実験情報ダイアログに入力した値でターゲットの大きさや固視点からの距離を変更できるはずです。

チェックリスト

- 実験情報ダイアログに入力されたデータの型を答えることができる。
- データを整数型、浮動小数点型、文字列型、リストに変換することができる。
- 実験情報ダイアログを用いて刺激の大きさや位置などの値を実験開始時に指定するように実験を作ることができる。

5.6 複雑な式には Code コンポーネントを使ってみよう

本章を終える前にあと一つ、今このタイミングで話しておきたいことがあります。本章で **[位置 [x, y] $]** に複雑な式を入力しましたが、入力欄が狭くて非常に作業しにくかったのではないでしょうか？ 特に、正しく入力したつもりなのに「Python の構文エラーがあります」と表示された場合、間違っている場所を探すのがとてもやりくかったのではないかと思います。このような複雑な式を扱う時に便利なのが Code コンポーネントです。Code コンポーネントはコンポーネントペインの「カスタム」というカテゴリにあります (図 5.9)。

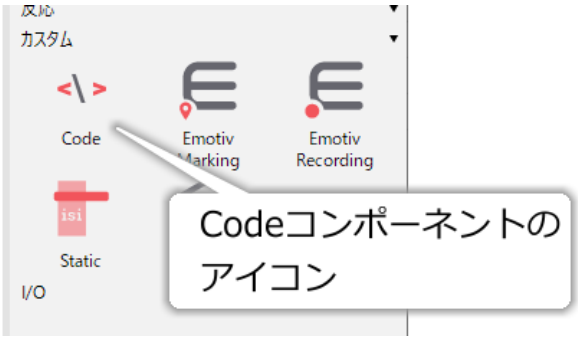


図 5.9　Code コンポーネントのアイコン。「カスタム」カテゴリにあるので注意してください。

Code コンポーネントは、実験が始まる時やルーチンが始まる時、各フレームの描画を行う時など、さまざまなタイミングで Python のコードを実行させることができるコンポーネントです。Code コンポーネントをルーチンに配置すると今までのコンポーネントと同様にプロパティ設定ダイアログが開きますが、その内容は大きく異なります (図 5.10)。まず、ダイアログ上部におなじみの **[名前]** があり、その右に **[コードタイプ]** という項目があります。さらにその右に **[無効化]** という項目がありますが、これは他コンポーネントの **[コンポーネントの無効化]** と同じ働きをするものです (「 4.9: 動作確認のために一部の動作をスキップしよう 」参照)。

[コードタイプ] は、PsychoPy 標準の設定から変更していなければ「Auto->JS」となっているはずです。もし他の値になっていれば、ダイアログのレイアウトが 図 5.10 と一致しない可能性がありますので、ひとまず「Auto->JS」してください。「Auto->JS」設定されていれば、ダイアログ中央には左右に分割された大きな入力欄があるはずです。そして入力欄の上には **[実験初期化中]** 、 **[実験開始時]** 、 **[Routine 開始時]** 、 **[フレーム毎]** 、 **[Routine 終了時]** 、 **[実験終了時]** というタブがあります。実はこのタブのそれぞれが Code コンポーネントのプロパティであり、他のコンポーネントと同様にプロパティ毎に入力欄を並べるとダイアログが大きくな

りすぎてしまうので、このようにタブで切り替えるようになっています。以後、Code コンポーネントについて「**[実験開始時]** に…と入力する」と書いた場合、「**[実験開始時]** タブを選んでその下の入力欄に入力する」ことを表すとします。

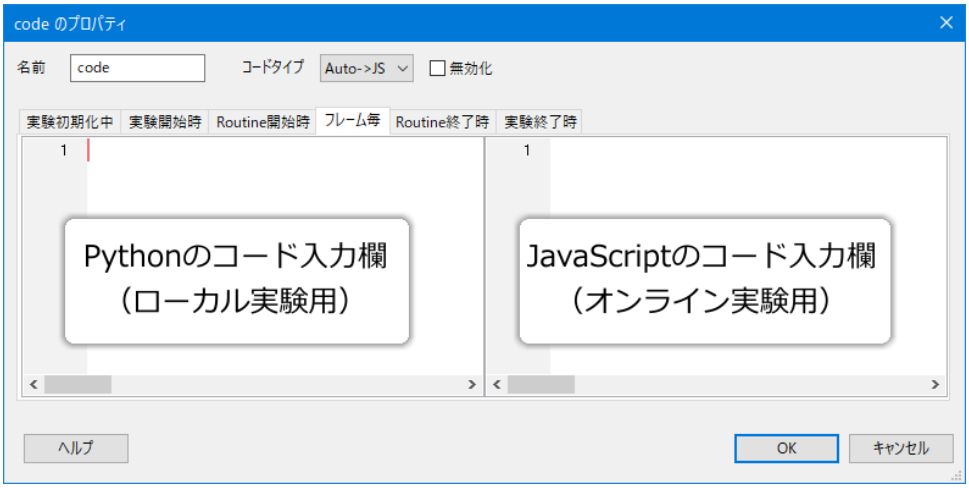


図 5.10　Code コンポーネントのプロパティ設定ダイアログ。

入力欄が大きく左右に分割されているのは、**オンライン実験** に対応するためです。Builder はもともとローカルな（つまり目の前にある）PC 上で動作する Python の心理学実験プログラムを人間の代わりに書いてくれるアプリケーションなのですが、現在のバージョンの Builder には **Pavlovia** (https://pavlovia.org/) というオンライン実験サービス用のプログラムを出力する機能も備わっています。オンライン実験用プログラムというのはインターネット上のページを閲覧するために用いる（Google Chrome や Mozilla FireFox などの）ブラウザで動作するのですが、ブラウザ上では Python よりも JavaScript という言語を使う方が有利なので Builder は JavaScript でプログラムを出力します。ユーザーが Code コンポーネントに Python で書きこんだプログラムが Builder によってどのように JavaScript に翻訳するかを確認したり、必要に応じて JavaScript のコードも手作業で編集したりできるように、入力欄が左右 2 つ用意されています。左側が Python コード用、右側が JavaScript コード用です。

本書では、オンライン実験のことはひとまず忘れてローカル PC 上で動作する実験の作成方法を解説しますので、左側の入力欄を使います。右側の入力欄は無視しても構わないのですが、小さな画面のノート PC で作業する場合などは邪魔に感じるかもしれません。そのような場合、ダイアログ上部の **[コードタイプ]** を「Py」にすると、Python 用の入力欄だけをダイアログの横幅いっぱいに表示させることができます (図 5.11 上)。この作業は Code コンポーネントを新たに置くたびに行わないといけないのですが、それが面倒な場合はメニューの「ファイル」から「設定」を選んで PsychoPy の設定ダイアログを表示し、「アプリケーション」ページの「Code コンポーネントの言語」を「Py」に変更してください。これで Code コンポーネントを配置する時の標準のコードタイプが Py になります。

Code コンポーネントの解説に戻りましょう。**[実験初期化中]** から **[実験終了時]** までの各プロパティは、コードを実験中のどのタイミングで実行したいかによって使い分けます。表 5.5 にコードが実行されるタイミングを示します。現時点でこの表を見てもよく意味が分からないと思いますが、本書ではこれからさまざまな目的で Code コンポーネントを使っていきますので、少しずつイメージをつかんでいただければと思います。

図 5.11　Code コンポーネントのプロパティ設定ダイアログ。

表 5.5　Code コンポーネントの各プロパティに記入したコードが実行されるタイミング

プロパティ	概要
[実験初期化中]	実験の初期化中、実験情報ダイアログを開く直前に実行されます。
[実験開始時]	実験情報ダイアログの OK がクリックされて刺激描画用ウィンドウが作成された直後、実験が実際に開始される前に実行されます。
[Routine 開始時]	Code コンポーネントを配置したルーチンが開始される直前に実行されます。ルーチンがループ内にある場合、繰り返しのたびに実行されます。
[フレーム毎]	Code コンポーネントを配置したルーチンの実行中、フレーム描画のたびに実行されます。つまり、毎秒 60 フレームで実験が実行されている場合、1 秒間に 60 回実行されます。
[Routine 終了時]	Code コンポーネントを配置したルーチンが終了した直後に実行されます。ルーチンがループ内にある場合、繰り返しのたびに実行されます。
[実験終了時]	フローのすべてのルーチンやループの実行を終えて、実験が終了する直前に実行されます。

それではさっそく、本章の「**[位置 [x, y] $]** の複雑で読みにくい式」を Code コンポーネントですっきりさせてみましょう。exp05.psyexp を開いて、trial ルーチンに Code コンポーネントを配置してください。そして **[実験開始時]** に以下のように入力します。

```
ecc = float(expInfo['eccentricity'])
```

おおよそ想像できると思うのですが、この式は ecc という変数に float(expInfo['eccentricity']) の値を割り当てることを意味しています。この操作を「代入する」といい、=記号を **代入演算子** と呼びます。 **[実験開始時]** にこのコードを実行することによって、実験の以後の部分では ecc という名前で float(expInfo['eccentricity']) の

値を利用できます。したがって **[位置 [x, y] $]** の式は以下のように短く書くことができるようになりました。

```
[ecc * cos(deg2rad(targetPos)), ecc * sin(deg2rad(targetPos))]
```

代入演算子にはいくつか注意してほしい点があるのですが、ここに書くと脱線が長くなるので「5.8.2: 代入演算子に関する注意点」をご覧ください。

さて、同じ要領で deg2rad(targetPos) という部分もすっきりさせてしまいましょう。

```
q = deg2rad(targetPos)
```

としてやれば、**[位置 [x, y] $]** の式は以下のようにシンプルにできます。余談ですが、変数を q としているのは角度を表す時によく用いられるギリシャ文字 θ に対応するアルファベットが q だからです。

```
[ecc * cos(q), ecc * sin(q)]
```

問題は、この式を実験中のどの時点で実行すればよいかです。先ほどの ecc と同じ **[実験開始時]** に追加すると、taregetPos の値がまだ Excel ファイルから読み込まれていないのでエラーとなってしまいます。targetPos は trials ループによる繰り返しのたびに値が更新されるのですから、trials ループの内部で式を実行しなければいけません。さらに、刺激を提示する前の時点で値が決まっていなければいけませんので、**[Routine 開始時]** が最も適しています。作業手順と解説が混じってわかりにくかったと思うので、ここまでの手順を整理しておくと以下のようになります。

- trials ルーチンに Code コンポーネントを配置する。
- **[実験開始時]** に ecc = float(expInfo['eccentricity']) と入力する。
- **[Routine 開始時]** に q = deg2rad(targetPos) と入力する。
- targetStim (Grating コンポーネント) の **[位置 [x, y] $]** を [ecc * cos(q), ecc * sin(q)] に変更する。

これで挿入するコードは完成ですが、あともう一つ大切なポイントがあります。ecc や q の値は、targetStim が描画される前に決定されていなければいけません。言い換えると、targetStim を描く処理を実行する前に Code コンポーネントのコードが実行されなければならないということです。この実行順序を決めているのは、ルーチンペインに並んでいるコンポーネントの順番です。「2.6: 刺激の重ね順と透明度を理解しよう」でルーチンペイン上でのコンポーネントの順番が刺激の重ね順に対応していることを学びましたが、正確にはこの順序は **コンポーネントの実行順序に対応しています**。視覚刺激の場合は「実行する＝画面に描画する」ということであり、先に描いたものの上に後から描いたものが上描きするので、下の方にあるものほど上に描かれるのです。したがって、targetStim の描画前に Code コンポーネントを描画するには、図 5.2 のように Code コンポーネントがルーチンペイン上で targetStim より上に置けばよいということです。

コンポーネントを並び替えたら実行してみましょう。Code コンポーネント導入前と同じように動作したはずです。以上で、動作を同じに保ったまま **[位置 [x, y] $]** の式をかなりシンプルに、意味を理解しやすくすることが出来ました。プログラムのコードというものは、書いている時は意味が分かっていても、数か月経ってから見直すとさっぱり意味がわからなくなりがちなものです。いま作成中の実験を将来利用する人のために (それは自分自身かも知れませんし、研究室の後輩や同じ分野の研究者かも知れません)、できる限り読みやすいコードを書くことをお勧めします。

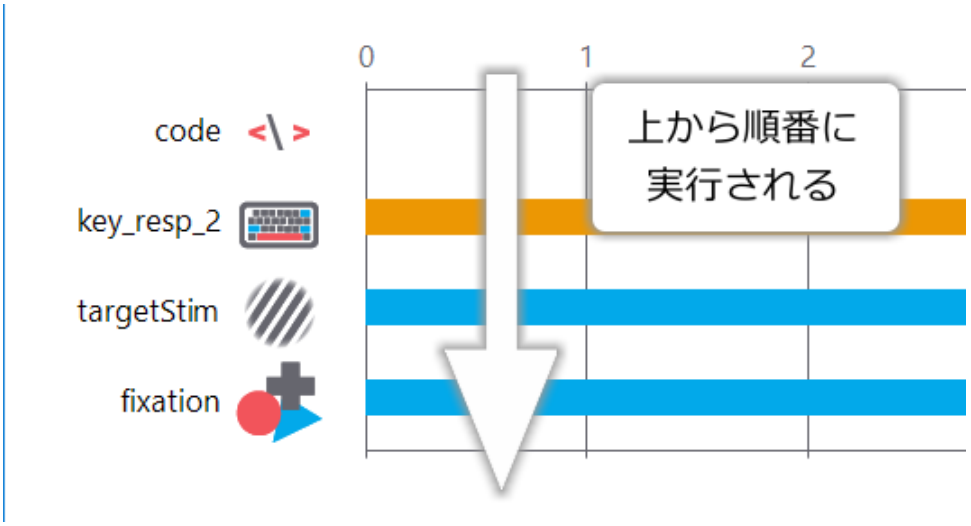


図 5.12　ルーチン上のコンポーネントは上から下の順に実行されます。刺激のパラメータを Code コンポーネントで設定するなら、Code コンポーネントは刺激描画用のコンポーネントより上になければいけません。

以上、Code コンポーネントを使って複雑なコードを読みやすくしましたが、実はここでこのテクニックを解説しておいたことにはもうひとつ理由があります。それは、 **将来的に Builder でオンライン実験を作成することを考えているなら、単純な数値の四則演算以上の複雑な式はコンポーネントのプロパティに書くべきではない** からです。先ほど述べたように、Builder が作成するオンライン実験は JavaScript という言語で書かれています。もしコンポーネントのプロパティに Python の式が書かれていると、オンライン実験として出力した時に JavaScript にうまく翻訳できない可能性があります。例えば今回の式の場合、現時点では cos() や sin() をうまく翻訳できません。 **[Routine 開始時]** のコードを

```
q = deg2rad(targetPos)
targPos_xy = [ecc * cos(q), ecc * sin(q)]
```

としておいて、targetStim の **[位置 [x, y] $]** を targPos_xy と書くようにしておけば、Code コンポーネントで JavaScript 用のコードを手作業で書くことで対応できます。Builder でのオンライン実験作成に興味がある人は覚えておいてください。

チェックリスト

- Code コンポーネントを用いて Python のコードをルーチンに配置することができる。
- Code コンポーネントのプロパティである **[実験初期化中]**、**[実験開始時]**、**[Routine 開始時]**、**[フレーム毎]**、 **[Routine 終了時]**、 **[実験終了時]** の違いを説明できる。
- Code コンポーネントに Python 用のコード入力欄だけを表示するように設定できる。
- ルーチンにおける Code コンポーネントと他のコンポーネントの実行順序を説明できる。
- 変数に値を代入する式を書くことができる。

5.7 練習問題：パラメータが適切な範囲を超えないようにしよう

この章では Builder の Python のコードを書いてみました。まだまだ解説したいことはたくさんありますが、ここで一区切りとします。最後に練習問題として、exp05.psyexp を改造して、trial ルーチンでキーが押されるまで刺激を提示し続けるようにしてください。ターゲット出現後 6 秒以上経過した後はキーが押されるまで、targetStim の **[前景色]** はずっと 1.0, 1.0, 1.0 の値を保ち続けるものとします。

キーが押されるまで trial ルーチンを継続し、刺激を提示し続けることはこの章まで進んだ皆さんなら問題はないはずです。この問題のポイントは、exp05.psyexp において、trial ルーチンが 6 秒以上継続してしまうと targetStim の **[前景色]** に入力した式 (t/6.0, t/6.0, t/6.0) の値が 1.0 を超えてしまう点です。一度皆さんも試してみていただければと思いますが、 **[前景色]** に入力した式を exp05.psyexp のまま変更せずに 6 秒以上刺激を提示すると、モノクロのグレーティング刺激を提示していたはずなのに突然カラーグレーティングが提示されてしまいます。そのまま放置していると実験が正常に動作しなくなる場合もありますので、現象を確認したらすぐにエスケープキーを押して実験を終了しましょう。 **[前景色]** に入力した式の RGB 各成分の値が 1.0 を超えないようにする方法を考えてください。練習ついでに、targetStim の **[位相 (周期に対する比) $]** の式を t にして「フレーム毎に更新する」にしてみましょう。グレーティング刺激を運動させることができます。運動知覚の実験ではよく使われる刺激ですので覚えておく価値があると思います。

どうしても答えがわからないという方向けに少しだけヒント。 表 5.2 のいずれかの関数を使えば式の値が一定値を超えないように制限することができます。先の章の話ですが「 8.3: ミューラー・リヤー錯視の実験をマウスで操作できるようにしよう 」も参考になるでしょう。

5.8 この章のトピックス

5.8.1 他の数学関数を使用する方法

本文中で述べたとおり、指数関数 e^x は標準の状態では Builder に読み込まれません。Python の膨大な数学関数を利用するためには Code コンポーネントを用いてそれらの関数を Builder に読み込む必要があります。Code コンポーネントを用いて実験の最初に以下のコードを実行すると、exp(x) という x の指数関数の値を得る関数が利用できるようになります。

```
from numpy import exp
```

一般に、foo というパッケージ（またはモジュール）に含まれる bar という名前の関数等を参照する時には

```
from foo import bar
```

と書きます。Code コンポーネントを用いて読み込んだ場合は使用済み名前のチェックが効かないので十分に注意してください。もっと踏み込んだ話をしますと、異なるパッケージで同一の名前の関数が存在する場合がありますので、上記の from を用いる方法よりも

```
import foo
```

と書いて foo パッケージを読み込み、

```
foo.bar(x)
```

という具合にドット演算子を使ってパッケージ名を明示する方が安全です。

5.8.2 代入演算子に関する注意点

Python における代入はかならず「右辺の値を左辺の変数へ」という方向で行われるので、以下の式はエラーとなります。

```
x+7 = 3
```

ちょっと注意が必要なのは、以下のような式です。これは右辺の「x+7 を計算して、その結果をまた x に代入する」という意味で、刺激の位置などを一定量だけ増減させるときなどに非常によく使われます。

```
x = x+7
```

数学では=演算子を「左辺と右辺が等しい」という意味でも使うので、この文をそちらの意味で解釈しようとして「意味が分からない」という方が時々おられます。 **Python では=演算子を「左辺と右辺が等しい」という意味で使うことはありません** 。このことはよく覚えておいてください。「左辺と右辺が等しい」ことを表す演算子は次章で登場します。なお、この「計算結果を元の変数に格納する」という式は非常に良く用いられるので、Python には二項算術演算子 (x+y のように二つの値をとる算術演算子) と代入演算子を組みあわせた演算子が用意されています (図 5.13 左)。x=x+7 を x+=7 に書き換えるといった具合に二項算術演算子と代入演算子を続けて書くと、右辺に変数名を書く必要がなくなります。変数名が x などのように短い時にはあまり意味がないのですが、 図 5.13 右のように変数名が長くなった場合に「target_leftside_length から 3 を引く」ということがとても読み取りやすくなります。

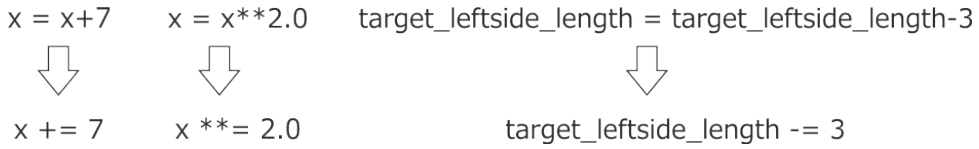


図 5.13　二項算術演算子と代入演算子の組み合わせ。変数名が非常に長い時に有効です。

5.8.3 実験情報ダイアログに入力した数値をそのまま使用した際のエラーに関する踏みこんだ議論 (上級)

第 6 版の改訂にあたって全体の内容を見直している際に、PsychoPy 2024.2.5 では第 5 版までに示していたものと異なるエラーメッセージが表示されることに気づきました。疑問に思ったので執筆時点で筆者の PC で動作確認可能な状態であった 2021.2.3、2022.2.4、2023.2.3 で試してみたところ、2021.2.3 から 2022.2.4 までの間に大きな仕様変更があり、その後も細かな変更が行われていることがわかりました。要点のみを書くと、2022 以降のバージョンで、 **[位置 [x, y] $]** や **[サイズ [w, h] $]** で数値であるべき部分が文字列であっても自動的に型変換が行われてエラーにならずに実行できてしまいます。つまり、 **[サイズ [w, h] $]** に

```
[expInfo['target size'], expInfo['target size']]
```

と入力して target size に 4.0 と入力した場合、

```
['4.0', '4.0']
```

という具合に幅と高さの値が文字列となりますが、エラーにならずに幅、高さとも 4.0 で描画されてしまいます。2021.2.3 やそれ以前のバージョンではこれはエラーとなっていました (第 5 版の記述もそれに基づいていた)。

この自動変換は要素に対してのみ行われるようで、 **[サイズ [w, h] $]** を

```
expInfo['target size']
```

として target size に (4.0, 4.0) と入力するとこれはエラーとなります。ややこしいのは target size に単に 4.0 と入力した場合で、これは自動的な型変換により数値となり、 **[サイズ [w, h] $]** スカラーが指定された場合は幅、高さとも同じ値が指定されたと見なす PsychoPy の仕様によってエラーとなりません。

このような自動的な型変換が導入された 2022 以降のバージョンでも、本文で述べたように eccentricity を含む式はエラーとなります。改めて問題の式を書くと

```
[expInfo['eccentricity'] * cos(deg2rad(targetPos)),
 expInfo['eccentricity'] * sin(deg2rad(targetPos))]
```

ですが、ここで eccentricity に 5 を入力すると

```
['5'*cos(deg2rad(targetPos)), '5'*sin(deg2rad(targetPos))]
```

という式になります。この式を **[位置 [x, y] $]** として扱おうとすると、まず式の評価が行われるので `'5'*cos(deg2rad(targetPos))` が計算されますが、この計算が不能なのでエラーとなるのです。注目すべきはこの時のエラーメッセージ (本文の 図 5.8) で、can't multiply sequence by non-int ですから **int であればエラーにならない** ということを意味しています。なぜかというと、Python ではシーケンス (文字列はシーケンスの一種) に整数をかけるとシーケンスを繰り返して結合したものが得られるからです。具体的に言うと、 `'5'*2` であれば `'55'` という文字列が得られます。この `'55'` は数値 (=55) に自動変換可能なので、恐らくコードの作者は 5 × 2=10 を意図していると思われますが 55 としてエラーにならずに動作してしまいます。

筆者の個人的な意見としては、このような自動型変換は特に初心者にとって混乱の原因となるだけだと思います (なぜあれはエラーにならないのにこれはエラーになるのか？)。そのため、本文では自動型変換については触れずに「expInfo から取得した内容を数値として使用する場合は関数を使って型変換する」と書くことにしました。

第 6 章

反応にフィードバックしよう―概念識別

6.1 この章の実験の概要

ずっと視知覚の実験ばかりが続いたので、この章では概念識別の実験を取り上げましょう。私たちが新しい概念を学ぶときには講義を聞いたり書籍を読んだりといった言語を通じた手段を用いることが多いでしょう。その他にも、ある概念について「この事例は当てはまる」、「あの事例は当てはまらない」といった事例を経験することによって、概念を獲得することがあります。例えば、幼児が言語を獲得する時に「これは『くるま』じゃないよ」、「いま『くるま』がみえたね」といった事例を通じて「くるま」という概念を獲得するといった具合です。ある概念が適用される事例を正事例、適用されない事例を負事例と呼びます。この章で取り上げる概念識別の実験は、この事例を通じた概念の獲得過程を単純化したものです。

実験では 図 6.1 に示す画像を刺激として使用します。眼鏡の有無 (かけている／かけていない)、顔の形 (丸い／四角い)、目の大きさ (大きい／小さい)、眉毛の形 (上がっている／下がっている)、口の形 (口角が上がっている／下がっている) の 5 種類の次元の組み合わせで合計 2 の 5 乗=32 種類の顔画像を用います。これらの画像ファイルは 図 6.1 下に示すように、'Face'+5 桁の数値 +'.png' という名前で保存され、数値の各桁の値が 5 種類の次元の値に対応しています。

実験の最初に無意味な単語をひとつ決定します。以後、この単語を「ターゲット語」と呼びます。例えば今、ターゲット語として「リニ」という無意味な単語を選んだとしましょう。この「リニ」の正事例として、5 種類の次元のいずれを選んでその値のどちらかを割り当てます。例えば「眼鏡の有無」の「かけていない」を選んだとしましょう。そうすると、これから実施する実験では「リニ」は刺激の顔が「眼鏡をかけていない」時に「当てはまる」、それ以外の時に「当てはまらない」ということになります。

以上のことを決めたうえで、実験に入ります。実験は 1 回から複数回のセッションから成っています (図 6.2)。各セッションの最初には、先ほど決定したターゲット語と、反応方法の教示が表示されます。反応方法は「当てはまる」ならばカーソルキーの左、「当てはまらない」ならカーソルキーの右を押すことにしましょう。実験参加者が教示画面でカーソルキーの左右いずれかを押すと最初の試行が始まります。各試行では、スクリーン中央にターゲット語と顔画像が提示され、スクリーン左下に「当てはまる」、右下に「当てはまらない」と提示されます。実験参加者はこの顔画像がターゲット語に「当てはまる」か「当てはまらない」か判断して、キーを押して反応します。反応に制限時間は設けず、刺激は反応があるまで提示し続けます。反応の検出後に、反応が正解であれば「正解です」、誤答であれば「不正解です」とスクリーンに表示して正誤を実験参加者にフィードバックします。32 種類の全ての画像に対して一回ずつ判断するまで試行を繰り返し、終了したら実験参加者に正事例の条件を口頭で回答させ、実験者は筆記します。以上の手続きを 1 セッションとします。本来なら参加者の回答をキーボードで文章として入力してもらいたいところですが、PsychoPy で文章入力を

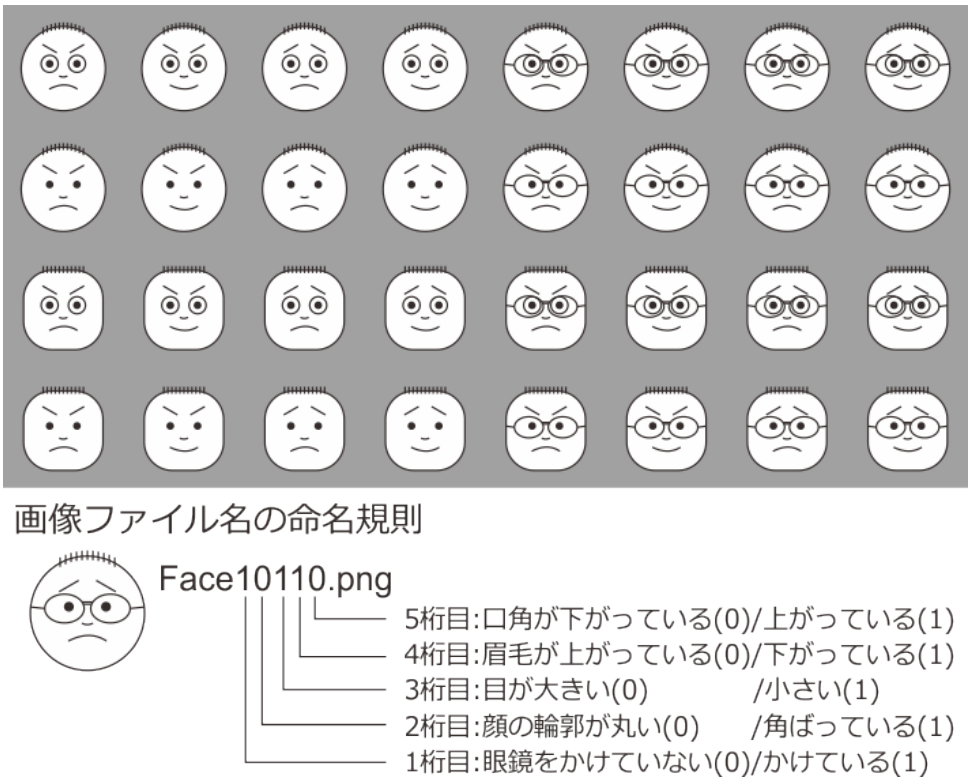


図 6.1　実験に用いる刺激。5 次元の特徴にそれぞれ 2 種類の値があり、合計 2 の 5 乗=32 種類の画像あります。

するのはちょっと難しいので今回はこういう形を取ります ((「 8.6.2: 日本語環境における Form コンポーネント (および TextBox コンポーネント) の文字入力機能 」も参照のこと)。実際の実験では複数セッション実施するでしょうが、複数セッションは 第 4 章 の多重ループで簡単に実現できますので、この章では 1 セッションのみを完成させることを目指しましょう。

以上が実験の概要です。実験の作成に入る前に、実験の作成に必要となる新しいコンポーネントの紹介をします。

6.2 Image コンポーネント

Image コンポーネントは **[画像]** で指定した画像ファイルをスクリーンに描画するコンポーネントです (図 6.3)。Grating コンポーネントと共通する部分が多く、Grating コンポーネントと同様に **[マスク]** 、 **[テクスチャの解像度 $]** や **[補間]** というプロパティがあります。これらのプロパティについては 第 4 章 を参考にしてください。

他コンポーネントと共通のプロパティのうち、注意が必要なのが **[サイズ [w, h] $]** と **[前景色]** です。まず **[サイズ [w, h] $]** ですが、ここを空欄にすると画像ファイルの本来の大きさで描画します。つまり、幅 640pix、高さ 480pix の画像であればそのまま幅 640pix、高さ 480pix で描かれるということです。これがどういう時に役に立つかというと、ループを使って縦横比が異なる画像ファイルを次々と表示するようなケースです。この時 **[サイズ [w, h] $]** を指定してしまうと、常に指定された縦横比で描かれるので、画像によって横方向か縦方向に引き延ばされてしまいます。ただ、この方法は height 単位のようにスクリーンの解像度に応じて刺激の大きさを調整してくれる機能と相性がよくありません。フル HD(解像度 1920 × 1080) のモニターではいい感じの大きさで表示されるのに、4K(解像度 3840 × 2160) のモニターだと小さすぎて刺激がよく見えないといったことが起こり得ます。実験室で、常に同じ解像度のモニターで実行するのでしたら問題にならないで

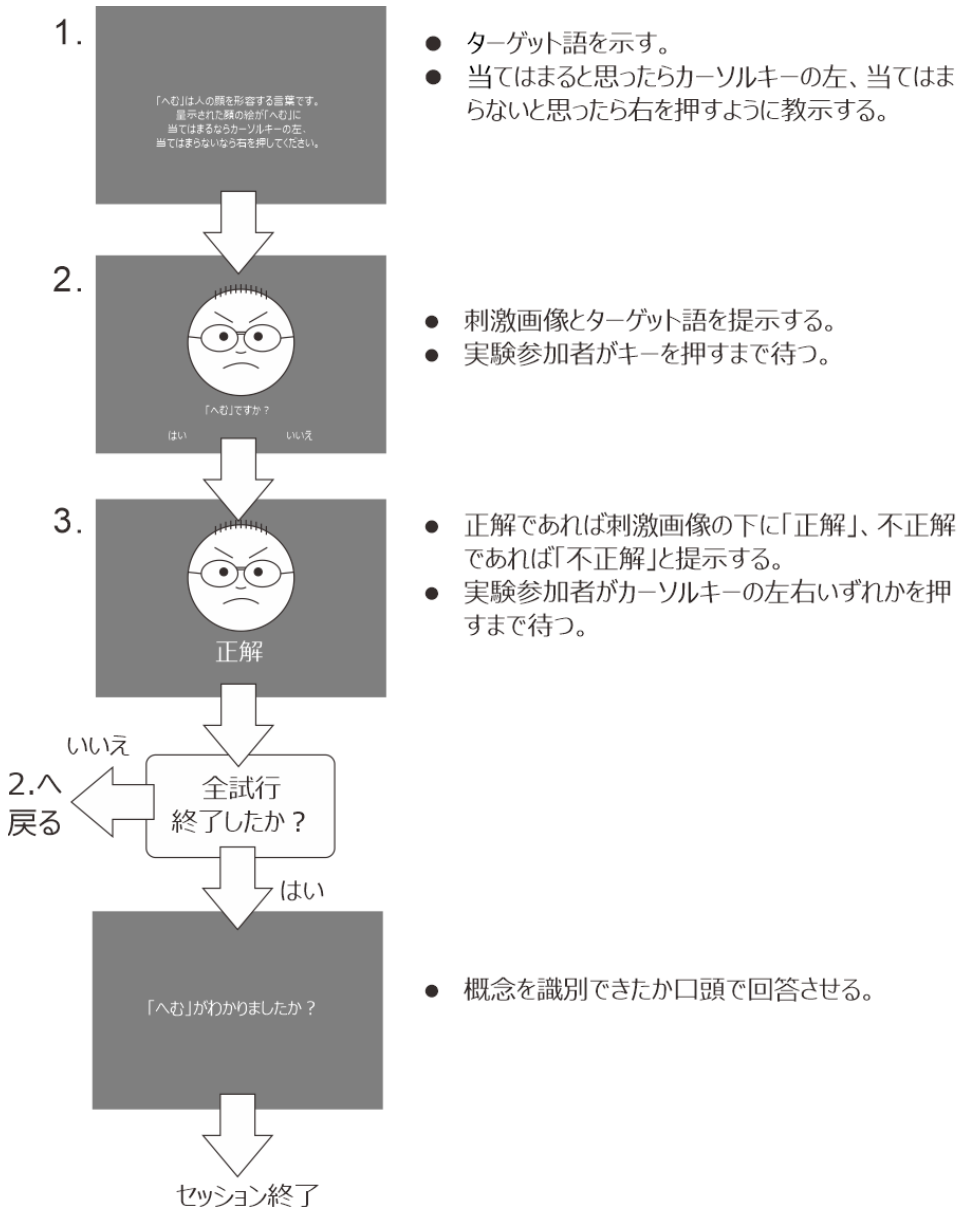


図 6.2　実験の概要。

図 6.3　Image コンポーネント

しょうが、デモ用に公開する実験やオンライン実験のように、どのような解像度のモニターで実行されるかわからない実験には不向きです。

height 単位の場合にスクリーンとの相対値で画像の大きさを指定しつつ、画像ごとに縦横比を変更したい場合は、条件ファイルを使って画像ごとに **[サイズ [w, h] $]** を指定する必要があります (実行中に画像の縦横比を計算することも可能ですが、本書で想定しているレベルを超えます)。あるいは、そういう面倒なことにならないように画像の縦横比がすべて同じになるように画像を用意するのもよいでしょう。

もうひとつの要注意プロパティである **[前景色]** ですが、画像を描画する際に、ここに指定した色と画像の色の乗算がおこなわれます。つまり、画像上のある位置の色が PsychoPy 風の表現で-0.3, 0.5, 0.7 であるときに **[前景色]** に-1.0, 0.0, 0.3 と指定されていたら、RGB 各成分をそれぞれ乗算して 0.3, 0.0, 0.21 として描かれるということです。これを使うと、R 成分だけを描画するとか G 成分だけを反転するといった簡単な画像処理だったら、前もって処理をおこなった画像ファイルを用意しなくても実行時に処理できるということです。 **[前景色]** の初期値は 1, 1, 1 ですので、1 を乗算しても RGB 値は変わらず画像の色そのままに描画されます。

Image コンポーネントで初登場のプロパティは **[水平に反転]** 、 **[垂直に反転]** 、 **[画像]** です。 **[水平に反転]** と **[垂直に反転]** にチェックを入れておくと、それぞれ画像が左右反転、上下反転された状態で提示されます。 **[画像]** には画像ファイル名を指定します。条件ファイルと同様に、ただファイル名だけが与えられた場合には、実行している psyexp ファイルと同じフォルダからファイルを探します。当然ファイルが見つからなければエラーとなり実験は停止してしまいます。psyexp と異なるフォルダにある画像ファイルを参照する場合は、絶対パスおよび相対パスによる指定が使えます。絶対パス、相対パスと言われてもピンと来ない方もおられると思いますので、節を改めて解説しておきましょう。

チェックリスト

- 画像ファイルをスクリーン上に提示することができる。
- 画像ファイルを上下、または左右に反転させて提示することができる。

6.3 絶対パスと相対パス

パス (path) とは「小道」のことで、PC ではプログラムを実行するときに開いたり保存したりしたいファイルへたどり着くための経路を表します。私たちの身の回りのものに例えるとすれば、「住所」によく似ています。「X 県 Y 市 Z 町 1 丁目 1-1」という住所があるとして、手紙の宛先にこの住所を書いておけば、全国どこから発送しても同じところへ配送されます。PC の場合でも同様に、PC のファイルシステム（ハードディスクや USB メモリ等）でファイルの位置を特定できる「住所」があります。この住所を絶対パスと呼びます。

Microsoft Windows の場合、USB メモリを差し込むと「ドライブ F:」などのようにアルファベットが割り当てられるのは御存知のことと思います。このアルファベットをドライブレターと呼びます。この USB メモリに「psychology」というフォルダがあって、さらにその中に「report」というフォルダがあって、その中の「report01.docx」というファイルがあるとします。この report01.docx を絶対パスで表すには、以下のようにドライブレターの後にコロンとバックスラッシュを書き、以後フォルダ名をバックスラッシュで区切って記述します。

```
F:\psychology\report\report01.docx
```

日本語 Windows ではバックスラッシュは半角の円記号 (¥) で表示されますのでご注意ください (「 3.12.5:$を含む文字列を提示する 」を参照)。日本での住所の表記が都道府県、市町村、と大きな区分から小さな区分に

向かって書かれるのと似ています。

Ubuntu などの Linux 系 OS では、バックスラッシュではなく以下のようにスラッシュでフォルダを区切ります。また、ドライブレターは使用されず、絶対パスの先頭はスラッシュです。

```
/home/user/Documents/Report/report01.txt
```

言うなれば先頭のスラッシュの前に「名前がない」フォルダがあることになりますが、この名無しフォルダの事を root と呼びます。ファイルシステムがこの名無しフォルダを根として枝が広がっていくように見えるところに由来する名称です。

絶対パスの良いところは、どの位置に目的のファイルがあるかを曖昧さなしに特定できることです。学期末のレポートの時期に「○○概論」や「△△特殊講義」といったあちこちのフォルダに report.doc という名前のファイルが散らかっていて訳が分からなくなることがありますが、絶対パスであればどのフォルダの report.doc であるかを見失うことがありません。しかし、この性質が逆に「融通の利かなさ」という欠点になる場面もあります。例えば自宅の PC で USB メモリが F: ドライブとして接続されていて、その中にある F:\Exp07\stim01.jpg という刺激画像を Builder から絶対パスで参照するようにしたとします。そして psyexp ファイルを保存して大学の実験室へ入り、実験室の PC に USB メモリを接続したら、E: ドライブとして認識されてしまったとしましょう。そうすると psyexp ファイルで参照している F:\Exp07\stim01.jpg の絶対パスは今や E:\Exp07\stim01.jpg に変わっていますので、それに合わせて psyexp ファイルを書き換えないといけません。これはあまりにも面倒です。このようなときに便利なのが相対パスによる指定です。

相対パスとは、住所の例えで言うならば、何県の話をしているのか文脈から明らかなときに県を省略して「Y 市 Z 町 1 丁目 1-1」のように書くことに似ています。しかし、住所の例えでは PC の相対パスは上手く説明できません。PC の相対パスでは、まず基準となる位置を決めて、そこから住所をどのように辿っていくかを記述します。基準となる位置のことをカレントフォルダと呼びます。今、F:\experiment\exp01 というフォルダに exp01.psyexp という Builder の実験ファイルがあるとしましょう (図 6.4)。この exp01.psyexp を実行する時には、F:\experiment\exp01 がカレントフォルダとなります。exp01.psyexp から他のファイルを探す時、絶対パスではないパス (ドライブレターや root から始まっていない書き方) が与えられると、カレントフォルダからファイルを探します。第 3 章 以降で条件ファイルを指定する時に exp01cnd.xlsx という具合にファイル名だけを書いたことを思い出してください。これは絶対パスではないので、カレントフォルダである F:\experiment\exp01 の exp01cnd.xlsx を指すことになります (図 6.4 (1))。image\stim01.png と指定されたら、カレントフォルダの中に含まれる image というフォルダの中にある stim01.png を指していると解釈されます (図 6.4 (2))。このようなカレントフォルダを基準にした指定方法を相対パスと呼びます。

普通の住所の表記と相対パスが大きく異なるのは、相対パスには階層をさかのぼる記法が用意されている点です。相対パスの中にピリオドが 2 文字だけのフォルダ (..\ ) が含まれていると、それは現在のフォルダを含んでいる上位のフォルダを指します。 図 6.4 の例では、..\が F:\experiment を指します。この記法は重ねて使用することができるので、..\..\と書くと F:\experiment のさらに上位のフォルダである F:\を指します。..\で指し示される上位フォルダのことを親フォルダと呼びます。この記法を用いることによって、カレントフォルダより上位のフォルダに含まれるファイルでも自在に指定することができます。 図 6.4 の (3) の exp02cnd.xlsx へ到達するためにはまず..\で親フォルダに移動し、そこから exp02 フォルダへ下ればたどり着けますから..\exp02\exp02cnd.xlsx と書きます。同様に (4) の face001.jpg にたどり着くためには、..\..\と書いて親フォルダを二つ遡って F:\まで移動し、そこから photo フォルダ、face フォルダと下ればたどり着きますから..\..\photo\face\face001.jpg と書けばよいということです。Builder で大量の画像ファイルを刺激として使う場合や、複数の実験で同じ画像を使いまわす場合などには、この相対パスの表記法を覚えておくと必ず役に

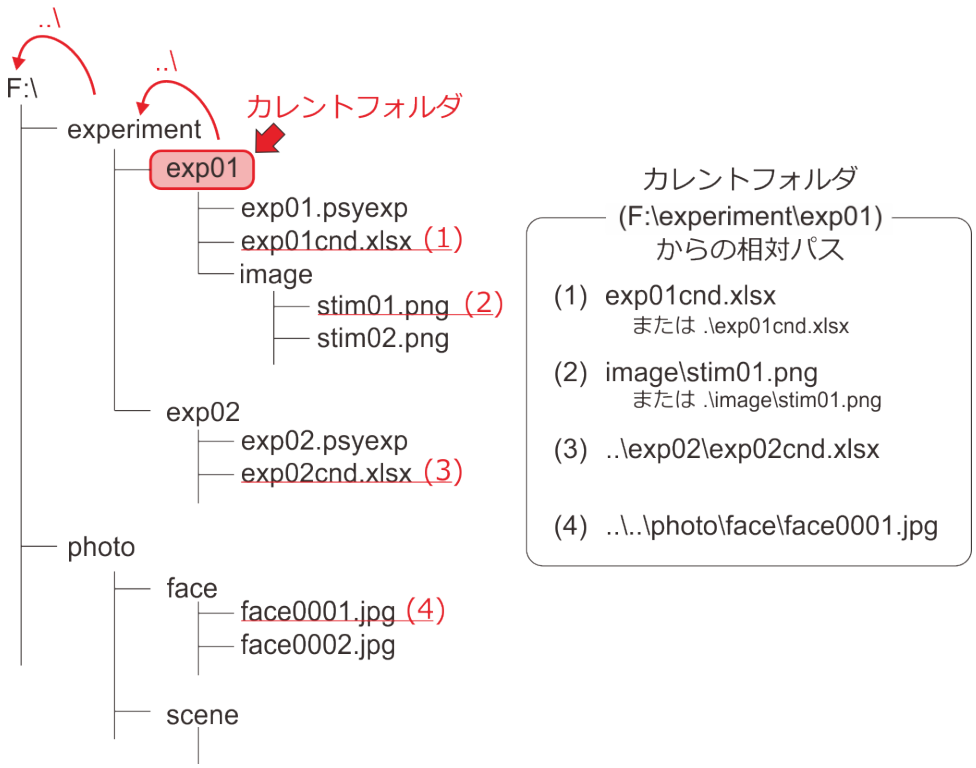


図 6.4　相対パスによるファイルの指定。

立ちます。しっかり理解しておきましょう。

相対パスのもう一つの利点は、「相対パスを使えば Python が OS によるパスの区切り文字の違いを吸収してくれる」という点です。先ほどから繰り返し例を示しているように、Microsoft Windows ではフォルダ名を区切る文字としてバックスラッシュを用います。ですから、Windows 上で動作するプログラムで相対パスを記述する時には image\stim01.png という具合に書かなければいけません。しかし、先述のように Linux ではスラッシュで区切りますので、Windows 上で動いたプログラムをそのまま持ってきても動作しません。この問題は「個人用 PC は Windows だけど大学の実験室では Linux」といった状況では困りものなのですが、ありがたいことに Windows 版の Python にはパスを Linux 流に image/stim01.png と書いても適切に解釈してくれる機能があります。ですから、Linux 流の相対パスを書いておけば Windows でも Linux でも動作する Builder の実験を作ることができるのです。「Windows 上でもスラッシュをパスの区切り文字と見なしてくれる機能は絶対パスの時にも有効なんじゃないの？」と思われる方もいるかも知れませんが、絶対パスの場合はドライブレターの問題が立ちはだかります。さすがの Python でもドライブレターを自動的に補ったり削除したりはできませんので、絶対パスで書くと OS 依存になってしまいます。

最後に、親フォルダの記法を紹介したついでに、カレントフォルダの記法も一応紹介しておきます。ピリオド 1 文字だけのフォルダ名はカレントフォルダとして解釈されます。ですから、 図 6.4 の (1) は.\exp01cnd.xlsx とも書くことができます。 (2) も同様に.\image\stim01.png と書くことができます。Builder を使うだけでしたら覚える必要はないのですが、今後本格的なプログラミングを学習したりする時には知っていると役に立つかもしれません。

チェックリスト

- 画像ファイルや条件ファイル等の位置を絶対パスで指定することができる。

- 画像ファイルや条件ファイル等の位置を相対パスで指定することができる。
- OS によるパスの書き方の違いを説明できる。
- 複数の OS で実行できる Builder の実験を作成するためにはどの記法でパスを記述したらよいか答えられる。

6.4 実験の作成

コンポーネントの解説が終わったので、実験の作成に入りましょう。

準備作業

- この章の実験のためのフォルダを作成して、その中に exp06.psyexp という名前で新しい実験を保存する。

実験設定ダイアログ

- 「基本」タブの **[実験情報ダイアログ]** に word という項目と、condition という項目を追加する。
- 「スクリーン」タブの **[単位]** が height でなければ height にする。
- 「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。

- trial ルーチン
  - Image コンポーネントをひとつ配置して、以下のように設定する。
    - 「基本」タブの **[名前]** を `faceImage` に、 **[終了]** を空白にする。 **[画像]** に `$imageFile` と入力し、「繰り返し毎に更新」にする。
    - 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.5, 0.5)` に、 **[位置 [x, y] $]** を `(0.0, 0.1)` に設定する。
  - Text コンポーネントをひとつ配置して、以下のように設定する。最終的に 3 つの Text コンポーネントが配置される。
    - 「基本」タブの **[名前]** を `textYes` 、**[終了]** を空白にし、 **[文字列]** に **はい** と入力する。
    - スクリーン左下に提示されるように、「レイアウト」タブの **[位置 [x, y] $]** を `(-0.2,-0.3)` に設定する。
    - 「書式」タブの **[文字の高さ $]** を `0.05` にする。
    - textYes をコピーして、 `textNo` 、 `textQuestion` という名前で貼り付ける。
    - textNo がスクリーン右下に提示されるように **[位置 [x, y] $]** を `(0.2,-0.3)` に設定する。 **[文字列]** に **いいえ** と入力する。
    - textQuestion が faceImage の下に提示されるように **[位置 [x, y] $]** を `(0,-0.2)` に設定する。textQuestion の **[文字列]** は後で変更するのでとりあえずそのままでよい。
  - Keyboard コンポーネントをひとつ配置し、以下のように設定する。

    * 「基本」タブの **[名前]** を key_choice とし、 **[終了]** が空白となっていることを確認する。 **[検出するキー $]** を 'left', 'right' にする。

    * 「データ」タブの **[正答を記録]** をチェックして、 **[正答]** に $correctAns と入力する。

- instruction ルーチン (作成する)
  - フローの先頭に挿入する。
  - Text コンポーネントをひとつ配置し、「基本」タブの **[名前]** を textInst に、 **[終了]** を空白にする。「書式」タブの **[文字の高さ $]** に 0.05 を入力する。
  - Keyboard コンポーネントをひとつ配置し、「基本」タブ **[終了]** を空白であることを確認する。 **[名前]** は key_resp のままでよい。 **[検出するキー $]** を 'left','right' にし、「データ」タブの **[記録]** を「なし」にする。
- feedback ルーチン (作成する)
  - フローの trial ルーチンの直後に挿入する。
  - trial ルーチンに配置してある faceImage をコピーし、 faceImage_2 の名前で貼り付ける。
  - Text コンポーネントをひとつ配置し、「基本」タブの **[名前]** を textFeedback に、 **[終了]** を空白にする。 faceImage_2 の下に提示されるように「レイアウト」タブの **[位置 [x, y] $]** を (0,-0.28) に設定する。「書式」タブの **[文字の高さ $]** を 0.08 にする。
  - instruction ルーチンに配置済みの key_resp をコピーして key_resp_2 の名前で貼り付ける。
- report ルーチン (作成する)
  - フローの末尾に挿入する。
  - Text コンポーネントをひとつ配置し、以下のように設定する。

    * 「基本」タブの **[名前]** を textReport に、 **[終了]** を空白にする。

    * 「書式」タブの **[文字の高さ $]** に 0.05 を入力する。
  - instruction ルーチンに配置済みの key_resp または feedback ルーチンの key_resp_2 をコピーして key_resp_3 の名前で貼り付ける。
- trials ループ (作成する)
  - trial ルーチンと feedback ルーチンを繰り返すように挿入する。
  - **[繰り返し回数 $]** を 1 にする。
- 刺激画像
  - こちらのリンク (http://www.s12600.net/psy/python/ppb/index.html#ppb-files) からダウンロードする。
  - exp06.psyexp の保存フォルダに image というフォルダを作成し、その中に保存しておく Windows エクスプローラの標準的な手順で ZIP ファイルを展開すると image というフォルダができてその

中に画像ファイルが作成されるはずなので、image フォルダをそのまま exp06.psyexp の保存フォルダに移動させるとよい。

- 条件ファイル (exp06cnd01.xlsx, exp06cnd02.xlsx)
  - 以下のように作成する。難しければこちらのリンク (http://www.s12600.net/psy/python/ppb/index.html#ppb-files) から ZIP ファイルをダウンロードして展開してもよい。
    * imageFile、correctAns の 2 パラメータを設定する。
    * imageFile に 32 種類の刺激画像のファイル名を **相対パスで** (つまり image/Face00000.png といった具合に) 入力する。
    * imageFile の列でファイル名の 5 桁の数字の左から 1 桁目が 1 である行の correctAns を left に、それ以外の行の correctAns を right にする。これで眼鏡をかけている画像ファイルに対応する correctAns が left となる。
    * exp06.psyexp の保存フォルダに exp06cnd01.xlsx という名前で保存する。Excel を閉じずにそのまま作業を続ける。
    * correctAns の列の right を left に、left を right に書き換える。これで眼鏡をかけていない画像ファイルに対応する correctAns が left となる。
    * Excel の **「名前を付けて保存」** で exp06cnd02.xlsx という名前で保存する。これで exp06.psyexp の保存フォルダに exp06cnd01.xlsx, exp06cnd02.xlsx という 2 つのファイルができる。

以上で準備完了です。条件ファイルは練習のために一度は自分で Excel を使って作成することをお勧めしますが、上記リンクからダウンロードできる conditions.zip に含まれる xlsx ファイルでは画像ファイル名や correctAns 列の値を Excel の関数を使って合成しているので、そういったテクニックをご存じない方は見ていただくと参考になると思います (図 6.5)。

	A	B	C	D	E	F	G	H	I
1	imageFile	correctAns	baseFilename	p_Glasses	p_Shape	p_Eye	p_Eyebrov	p_Mouth	
2	image/Face00000.png	right	Face00000.png	0	0	0	0	0	
3	image/Face00001.png	right	Face00001.png	0	0	0	0	1	
4	image/Face00010.png	right	Face00010.png	0	0	0	1	0	
5	image/Face00011.png	right	Face00011.png	0	0	0	1	1	
6	image/Face00100.png	right	Face00100.png	0	0	1	0	0	

図 6.5　条件ファイルの作成。Builder で使用する条件ファイルには実験に使用しない列があっても構わないことを利用して、右の方に画像の特徴 (眼鏡の有無や顔の輪郭など) を表す 0/1 の値を入力し、Excel の concat 関数で結合して画像ファイル名 (baseFilename) を作成している。さらにこの baseFilename に concat 関数でフォルダ名 (image/) を結合して今回の実験で必要な imageFile パラメータを合成している。correctAns パラメータに関しては、画像ファイル名の合成に用いた 0/1 の値に対して Excel の if 関数を適用して left, right の値を割り当てている。

いくつかの Text コンポーネントの **[文字列]** やループの **[条件]** などが未設定のまま残っています。これから、以下の作業に取り組みたいと思います。

- 実験実行時に実験情報ダイアログの word からターゲット語を取得し、instruction や trial ルーチン、report ルーチンに配置した Text コンポーネントの **[文字列]** で提示する教示文に組み込む。
- 実験実行時に実験情報ダイアログの condition から条件ファイル名の番号 (01、02) を取得し、前に

exp06cnd、後ろに.xlsx を結合して条件ファイル名を得る。条件ファイル名を全て入力する手間を省ける。

なお、本書の旧版では条件ファイルに画像ファイル名のみを (つまり image/をつけずに) 入力して、実行時に相対パスとして完成させるようにしていましたが、条件ファイルの時点で完全な相対パスを入力するように変更しています。旧版から活用いただいていて、この変更に興味がある方は「6.9.1: 条件ファイルに完全なパスを書くべきか？」をご覧ください。

6.5 文字列を結合して教示文を作成しよう

まず、Image コンポーネントの **[画像]** プロパティを設定しましょう。 **[画像]** には読み込む画像ファイル名を指定します。ファイル名は条件ファイルの imageFile というパラメータから読み込まれますが、単に$imageFile と書くと画像ファイルが exp06.psyexp と同じディレクトリにあると見なされてしまいます。そこでフォルダ名の image/をファイル名の前に補いたいのですが、$image/imageFile とか$"image/"imageFile とか書いてもエラーになります。ではどう書けばいいかと言いますと、+ 演算子を使います。数値と数値の間に + 演算子を書けば足し算になりますが、文字列と文字列の間に + 演算子を書くと両者を連結した文字列が得られます。今回の場合は、以下のように **[画像]** に記入すれば目的を達成できます。image/を文字列として認識させるためにシングルクォーテーションまたはダブルクォーテーションで囲むことを忘れないでください。

```
$'image/' + imageFile
```

trials ルーチンの faceImage と feedback ルーチンの faceImage_2 の両方の **[画像]** にこの式を入力してください。そして、忘れずに「繰り返し毎に更新」を設定しておきましょう。

教示文へのターゲット語の組み込みも同様の方法で実現できます。expInfo['word'] という式で実験情報ダイアログから文字列を取得できるのですから、以下のような式を用いれば「これは「○○」ですか？」という文字列が得られます。

```
$'これは「' + expInfo['word'] + '」ですか？'
```

trial ルーチンを開いて textQuestion の **[文字列]** にこの式を入力してください。実験の実行中には expInfo['word'] の値は変化しませんので、「繰り返し毎に更新」に設定する必要はありません。この式は少々複雑なので、+ 演算子の前後にスペースを入れて見易くしてみましょう。

```
$ 'これは「'    +    expInfo['word']    +    '」ですか？'
```

三つの文字列を 2 つの + 演算して結合していることがわかります。数値演算における + 演算子が 5+3+8 という具合に 3 つ以上の値に次々と適用できるのと同じことです。ただし、数値演算では 5+3+8 でも 8+3+5 でも同じですが、文字列の + 演算では常に演算子の左側の文字列の最後尾に右側の文字列の先頭が結合されます。

このテクニックを使って、condition へ入力された値から条件ファイル名を得る式も入力してしまいましょう。trials ループの設定ダイアログを開き、 **[条件]** に以下の式を入力してください。これで 01 とだけ入力すれば exp06cnd01.xlsx を指定することができます。

```
$'exp06cnd' + expInfo['conditionFile'] + '.xlsx'
```

続いて instruction ルーチンの textInst の **[文字列]** ですが、ここは少々解説が必要です。目標として、以下のような複数行にわたる教示文をひとつの Text コンポーネントで表示するものとします。〇〇には実験情報ダイアログの word から取得した文字列が入るものとします。

```
「〇〇」は人の顔を形容する言葉です。
提示された顔の絵が当てはまるならカーソルキーの左、
当てはまらないなら右を押してください。
```

〇〇にあたる部分を expInfo['word'] にして前後を + でつなげばよいだけのように思われるかもしれません。しかし実際に試してみると、 図 6.6 のように Builder に「構文エラーがある」と怒られて OK ボタンをクリックすることができません。

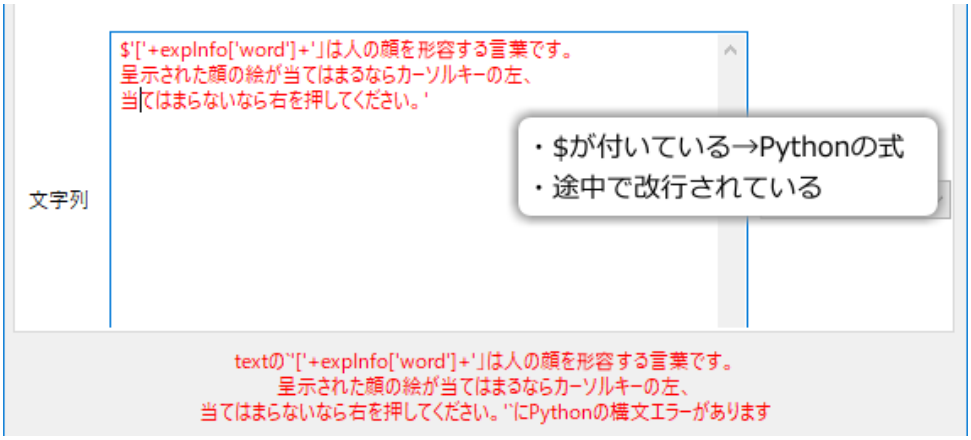


図 6.6　改行がある文と $の指定は両立しない

第 2 章 で述べたとおり、Text コンポーネントの **[文字列]** には改行を含むことができます。それなのになぜ 図 6.6 はエラーになってしまうのでしょうか。この問題の鍵は、Builder が **[文字列]** に入力されている値をどう解釈するかにあります。 **[前景色]** の設定を思い出してください (第 3 章)。 **[前景色]** には$が付いていないから、red と書けば red という文字列が入力されていると Builder は判断するのでした。一方、$red と書けば、変数 red に格納された値が指定されていると Builder は判断すると述べました。実は後者は不正確な記述で、正確には$が書かれていると、Builder は$を除いた部分が Python の式である判断するのです。$red から$を除くと red が残り、red という式を「評価」すると、変数 red に格納された値が得られるのです。ここで言う「評価」とは、式の中に + などの演算子が含まれていたらその計算をしたり、関数が含まれていたらその戻り値を計算したりといった作業を行って、最終的な式の計算結果を得ることです。この「評価」という考え方は Builder を理解する上でとても大切です。例えば 第 5 章 で出てきた時刻に応じて色を変化させる式 t/6.0, t/6.0, t/6.0 では、t/6.0 が評価されて t を 6.0 で割った値に置き換えられることによって、RGB 値として解釈できるリストとなるのです。

これを踏まえて 図 6.6 に戻ります。ここには$が入力されているので Builder はこれを Python の式と見なします。Python の式としてこれを解釈すると、1 行目の最後が文字列の途中で終わってしまっています。このような書き方は Python の文法で許されていません。同様に 2 行目、3 行目も文字列が適切にクォーテーションで囲まれていませんので Python の文法を満たしていません。これでは Builder に拒否されるのは当然です。

では目指している出力を得るにはどうすればよいでしょうか。ふたつ方法がありますが、簡単な方法を紹介しましょう。Python では、シングルクォーテーションまたはダブルクォーテーションを 3 個連ねることにより、複数行にわたる文字列を表記することができます。例えば以下のように書くと 2 行に表示されます。

```
'''必要に応じて休憩を取ってください。
準備ができたらスペースキーを押して実験を再開してください。'''
```

このテクニックを使うと、textInst の **[文字列]** は以下のように書くことができます。

```
'「'+expInfo['word']+'」は人の顔を形容する言葉です。
提示された顔の絵が当てはまるならカーソルキーの左、
当てはまらないなら右を押してください。'''
```

以上で **[文字列]** に$記号を使って Python の式を記述した場合でも複数行にわたる文字列を表示できました。この方法を知っていれば Builder を使う分には困ることはないと思いますが、今後さらにステップアップすることを考えるのならばもうひとつの方法も知っておくと役に立つかもしれません。「6.9.2: 改行文字を使った複数行の文字列の表現 (上級) 」に解説しておきますので、興味がある人は読んでください。

最後に残った report の教示もこの方法を活用しましょう。report ルーチンを開いて、textReport の **[文字列]** に以下の式を入力してください。

```
$'「' + expInfo['word'] + '」にあてはまる画像の条件を考えて、
 実験者に口頭で答えてください'''。
```

チェックリスト

- 複数の文字列を結合した文字列を得る式を書くことができる。
- 条件ファイルや実験情報ダイアログから読み込んだ文字列が組み込まれた文を提示することができる。
- Text コンポーネントの **[文字列]** に Python の式を書いた時に、表示する文字列を改行させることができる。

6.6 Python における比較演算子、論理演算子、条件分岐を学ぼう

これで残りの作業は「判断の正誤をフィードバックする」だけになりました。フィードバックの提示は feedback ルーチンの textFeedback を用いて行います。trial ルーチンでの判断が正しければ textFeedback の **[文字列]** に「正解」、誤っていれば「不正解」と設定したいのですが、当然実験参加者が正解するか否かは実験を実行する前にはわからないので、条件ファイルでは実現できません。この種の参加者の反応に応じた刺激や課題の変化は Builder が苦手とするところで、現状の Builder ではどうしても Python の文法知識、Python コードの記述が必要になります。

さて、これから「反応が正しければ『正解』、誤っていれば『不正解』と提示する」という作業を Python のコードへ変換するわけですが、このように条件に応じて行う処理を変更することを条件分岐と言います。条件分岐は

もし A が成り立つならば B を行う。さもなければ C をおこなう。

という文で表現できます。一般にプログラミング言語では「成り立つ」ことを「真 (True) である」と言い、成り立たないことを「偽 (False) である」と言います。この用語を用いると、先の文は

A が真であれば B を行う。偽であれば C をおこなう。

と書き直すことができます。Python では、この文を if, else という語を使って 図 6.7 のように書きます。if と else の後ろにコロン (:) がある点と、B、C が if や else より「字下げ」されている (行頭に空白文字がある) 点に注意してください。空白文字を何文字入れるかについての Python での文法上の取り決めは少々複雑なのですが、Python Enhancement Proposals (PEP) と呼ばれる Python の公式文書において「字下げには半角スペース 4 文字を用いる」ことが推奨されていますので、この文書では半角スペース 4 文字で統一します。第 7 章で詳しく触れますが、Python の文法では字下げが重要な意味を持っています。

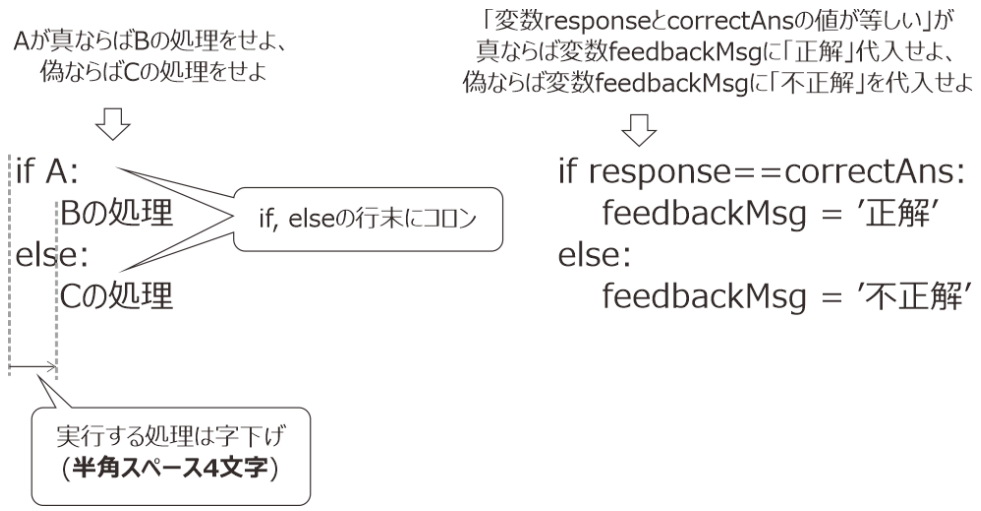


図 6.7　条件分岐 (if 文) の書式。右は実際のコードの例です。

反応が正しければ『正解』、誤っていれば『不正解』と提示する」という目標をこの if 文の形式に当てはめることができれば目的は達成されますが、どう当てはめたらよいでしょうか。この目標のままでは PC にとってはまだ抽象的すぎますので、もう少し書き直してみましょう。

・反応が正しければ
↓
・「押されたキーの名前が変数 correctAns の値と一致している」が真であれば

「押されたキーの名前」を Python のコードとして表現する方法はまだ解説していないので、とりあえず変数 response に押されたキーの名前が格納されているものとして書き換えを進めますと、以下の文が得られます。

・「押されたキーの名前が変数 correctAns の値と一致している」が真であれば
↓
・「変数 response の値が変数 correctAns の値と一致している」が真であれば

続いて「『正解』と提示する」という部分についても書き直してみましょう。提示には Text コンポーネントを使うのですから、以下のように書き直すことができます。

・『正解』と提示する
↓
・Text コンポーネントの「文字列」に' 正解' と設定する

第 3 章 以降の解説では、Builder のコンポーネントのプロパティ値を実行中に変更する時には変数を用いてきました。今回もこの方法が有効でしょう。feedbackMsg という変数を用いることにしましょう。

```
・Text コンポーネントの「文字列」に' 正解' と設定する
↓
・変数 feedbackMsg に' 正解' を代入する
```

「『不正解』と提示する」は「『正解』と提示する」と同様ですので省略します。ここまで書き直すことができれば、Python のコードへ変換することができます。 図 6.7 の右側が実際に Python のコードに置き換えてみた結果です。 図 6.7 左側と見比べて、 図 6.7 左側の A、B、C に対応する右側のコードを見てください。右側のコードの意味が何となく分かると思うのですが、ここで「なんとなく」で済ませると後で躓くのでしっかり理解しておきましょう。

まず if の後に続く response == correctAns ですが、response と correctAns はすでに何度も出てきている Python の変数であり、その中に値が保持されています。両者の間にある ==という記号ですが、これは比較演算子と呼ばれる演算子です。==の前後に置かれた値を比較して、両者が一致していれば True、一致していなければ False という「値」を返します。True や False を「値」と言われると奇妙に感じるかも知れませんが、そういうものだと思ってください。10+5 を評価すると 15 という値が得られるのと同様に、比較演算子を含む式を評価すると True や False という「値」が得られるのです (正確に知りたい方は「 6.9.3:True と False の「値」 」参照)。比較演算子には==の他にも 表 6.1 に示すものがあります。表中の X と Y が両方とも数値である場合は特に難しいことはないと思うのですが、どちらか一方に文字列やリストが含まれている場合は話が厄介です。詳しく知りたい方は「 6.9.4: 文字列、シーケンスに対する比較演算子 」を読んでいただきたいのですが、慣れないうちは以下の点を守って使用することをお勧めします。

- 文字列やシーケンス型の比較の場合は==(等しい) と!=(等しくない) 以外使用しない
- 異なる種類のデータ型の比較 (数値と文字列の比較) はしない

表 6.1　Python の比較演算子

X == Y	X と Y は等しい	X != Y	X と Y は等しくない
X < Y	X は Y より小さい	X <= Y	X は Y 以下
X > Y	X は Y より大きい	X >= Y	X は Y 以上

比較演算子は、二つ以上同時に使うこともできます。例えば以下の例では x が 1 以上で 5 未満の時に True、それ以外の時は False になります。数値がある範囲に収まっているか否かで処理を分岐させる時に便利です。

```
1 <= x < 5
```

比較演算子について学んだついでに、もうひとつ演算子を学んでおきましょう。「刺激の位置がスクリーンの左上だった場合」といった条件で分岐させたい場合には、X 座標が負の値であること、Y 座標が正の値であることの二つの条件を同時に満たす必要があります。このような場合は論理演算子 (表 6.2) を用います。X 座標と Y 座標の値がそれぞれ X、Y という変数に格納されているのであれば、この条件は以下のように記述できます。

```
X<0 and Y>0
```

逆に、「刺激の位置がスクリーンの左上ではなかった場合」とうい条件を指定したい場合は、X 座標が 0 以上、または Y 座標が 0 以下のどちらか一方が成立すればいいのですから、以下のように記述できます。

```
X>=0 or Y<=0
```

否定演算子を使うと以下のように書くこともできます。演算子を適用する順序を指定するために ( ) を使用していることに注意してください。まず ( ) の中が評価されて、その後に not が適用されます。

```
not ( X<0 and Y>0 )
```

表 6.2　Python における論理演算子

X and Y	X かつ Y (論理積)	X と Y がともに True の時に True
X or Y	X または Y (論理和)	X と Y のいずれかが True の時に True
not X	X の否定	X が True ならば False、False ならば True

比較演算子と論理演算子についての解説はこのくらいにして、 図 6.8 の B と C に対応するコードも見ておきましょう (#以下の部分は Python によって無視されるのでコメントを書く時に用いられます)。

```
# B にあたるコード
feedbackMsg = '正解'

# C にあたるコード
feedbackMsg = '不正解'
```

すでに前章で代入演算子を学んだ皆さんには、もうおわかりですね。feedbackMsg という変数に表示したい文字列を代入しています。

これで参加者の反応に応じて表示するメッセージを変更するための Python のコードの書き方がわかりました。残された問題は、この節の解説では「変数 response に格納されているものとする」と仮定した反応キー名をどうやって取得するかです。ここでまた新たな概念が登場するので節を改めましょう。

チェックリスト

- 条件に応じて処理を分岐させる Python コードを書くことができる。
- 数値の大小や一致・不一致に応じて処理を分岐させることができる。
- 文字列の一致・不一致に応じて処理を分岐させることができる。
- Python の比較演算子を 6 つ挙げてそれらの働きを説明することができる。
- Python の論理演算子を 3 つ挙げてそれらの働きを説明することができる。

6.7 オブジェクトのデータ属性を利用して反応にフィードバックしよう

いよいよこの章の実験の完成が近づいてきました。反応キー名を得るコードですが、これは知っていないといくら考えてもわからないことなので、解答から始めます。key_choice という名前の Keyboard コンポーネントから反応キー名を得るには以下のように書きます。

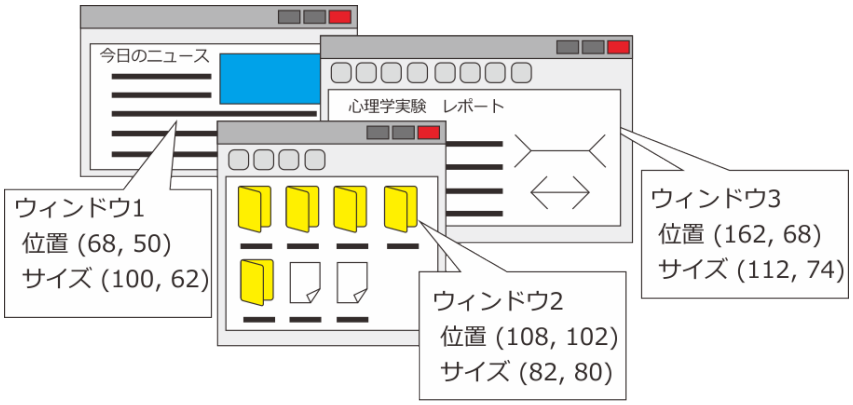
```
key_choice.keys
```

プログラミングの経験がない方には難しい話が続きますが、この部分を理解していないと次章以降の内容を理解できません。じっくり解説しますので、頑張ってついてきてください。

この key_choice.keys という式ですが、Python の文法では名前にピリオドを含むことはできませんので、key_choice と keys という名前がピリオドで結ばれているはずです。key_choice というのはルーチンに配置している Keyboard コンポーネントと同じ名前なので。恐らく Keyboard コンポーネントと関係がある「何か」だというのは容易に想像できるのではないかと思います。この key_choice というのは変数で、その中には Builder のキーボード反応計測用オブジェクト (psychopy.event.BuilderKeyResponse：以下 BuilderKeyResponse) というモノが入っています。オブジェクトとは、コンピュータ上で扱うさまざまな対象、キーボードやマウスといった物理的なものから各種ソフトウェアの動作のために用いられるデータなどをプログラミング言語上から扱いやすくするための仕組みです。

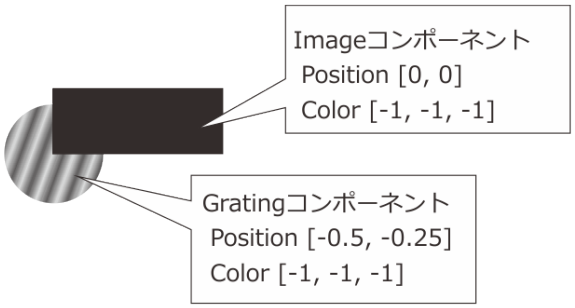
とても抽象的な概念でイメージしにくいと思いますので、具体的な例を挙げましょう。 図 6.8 は web ブラウザと文書作成ソフト、フォルダ内容表示の 3 つのウィンドウを開いている様子を示しています。これらのウィンドウは右上のボタンをクリックしてウィンドウを閉じたり最大化、最小化したりといった操作や、ウィンドウの枠をドラッグして位置や大きさを変更する動作は共通しています。しかし、ウィンドウを動かしてしまうと今まで別ウィンドウに隠されていた部分が見えるようになるので OS はウィンドウの描き直しを行うのですが、描き直しの方法は web ブラウザや文書作成ソフトなど、個々のウィンドウで異なるでしょう。このように、コンピュータ上で扱う対象には共通化できる部分と、固有の部分があります。この共通化できる部分を共通化するための仕組みがオブジェクトです。実は、オブジェクトという考え方はここまで解説してきた Builder のコンポーネントにも利用されています。例えば刺激に対応するコンポーネントにはいずれも **[位置 n [x, y] $]** や **[サイズ [w, h] $]** というプロパティがあり、これらを使ってスクリーン上のどの位置にどのくらいの大きさで描画されるかを指定することができました。 **[前景色]** プロパティで色を指定することもできましたが、Image コンポーネントと Grating コンポーネントでは同じ値を **[前景色]** に指定してもスクリーンに描かれる刺激の色は全く異なりました (図 6.8)。こういった刺激間の共通点や相違点を効率よくするために、Builder の内部ではオブジェクトが使用されています。

Python におけるオブジェクトでは、こういったさまざまな対象を表現するために、データ属性とメソッドと呼ばれるものを用います。Builder の刺激オブジェクトの例を用いると、データ属性とは **[位置 [x, y] $]** や **[サイズ [w, h] $]** のように、それぞれの刺激で固有の値をとるデータのことです。メソッドについては詳しくは次章で触れますが、それぞれの刺激をスクリーンに刺激を描画する手続きのように、そのオブジェクトに対する操作をおこなうものです。それぞれのオブジェクトのデータ属性やメソッドの定義をクラスと呼び、クラスに従って生成されたオブジェクトを該当するクラスのインスタンスと呼びます。 図 6.9 はデータ属性、メソッド、クラス、インスタンスの関係を示しています。Grating クラスはグレーティング刺激を描画するためのクラスで、 **[位置 [x, y] $]** 、 **[回転角度 $]** 、 **[テクスチャ]** に対応する Position、Orientation、Texture といったデータ属性を持っています。また、スクリーンに描画を行うための draw というメソッドを持っています。グレーティング刺激をスクリーンに 2 個描画するために、Grating クラスのインスタンスを 2 個生成して、それぞれ gratingPatch と stripePatch という変数に格納しました。各インスタンスのデータ属性に値を代入する時



共通点　ウィンドウの右上のボタンで閉じたり隠したりする動作
ウィンドウの枠をドラッグして位置や大きさを変える動作

相違点　位置やサイズの値
ウィンドウの内容を描き直す動作
ウィンドウ内部でマウスをクリックしたりキーを押したときの動作



共通点　Positionで指定した位置が刺激の中心となるように描画する

相違点　PositionやColorの値
Colorの値に応じて塗りつぶす方法

図 6.8　オブジェクトの例。個々のウィンドウはそれぞれ固有の位置や大きさを持ちますが、閉じたり移動させたりといった操作方法は共通しています。これまでに扱った Builder の刺激にも、それぞれに共通する点や異なる点があります。

には、gratingPatch.Position = (0,0) という具合に、変数名とデータ属性名をドット演算子 (.) で結合して記述します。ドット演算子を用いることによって、どちらのインスタンスに代入するのか混乱することがありません。draw メソッドを用いて刺激を描画する時には、gratingPatch.draw( ) という具合にやはりドット演算子を使って変数名とメソッド名を結合して記述します。メソッドは関数と同様に引数をとることができますので、( ) が draw の後に付きます。draw メソッドでは各インスタンスのデータ属性の値を用いて刺激の位置や波形が決定されて刺激が描画されます。

グレーティング刺激に追加して、多角形の刺激を描画する Polygon クラスを用いて三角形と五角形を 1 個ずつ描画するとします。Polygon クラスは Grating クラスと似ていますが Grating クラスには無い N_Vertices というデータ属性を持っています (**[頂点数]** に対応)。また、Grating クラスと同様に draw というメソッドを持っていますが、その処理内容は異なります。多角形を 2 個描画するのですから、Polygon クラスのインスタンスを 2 個生成して triangle と pentagon という変数に格納します。triangle と pentagon のデータ属性に適切な値を設定して、draw メソッドを呼びます。triangle に格納された Polygon を描画する時には trialgle.draw() と記述しますが、このように変数名とメソッド名をドット演算子で結合して記述することによって、この draw( )

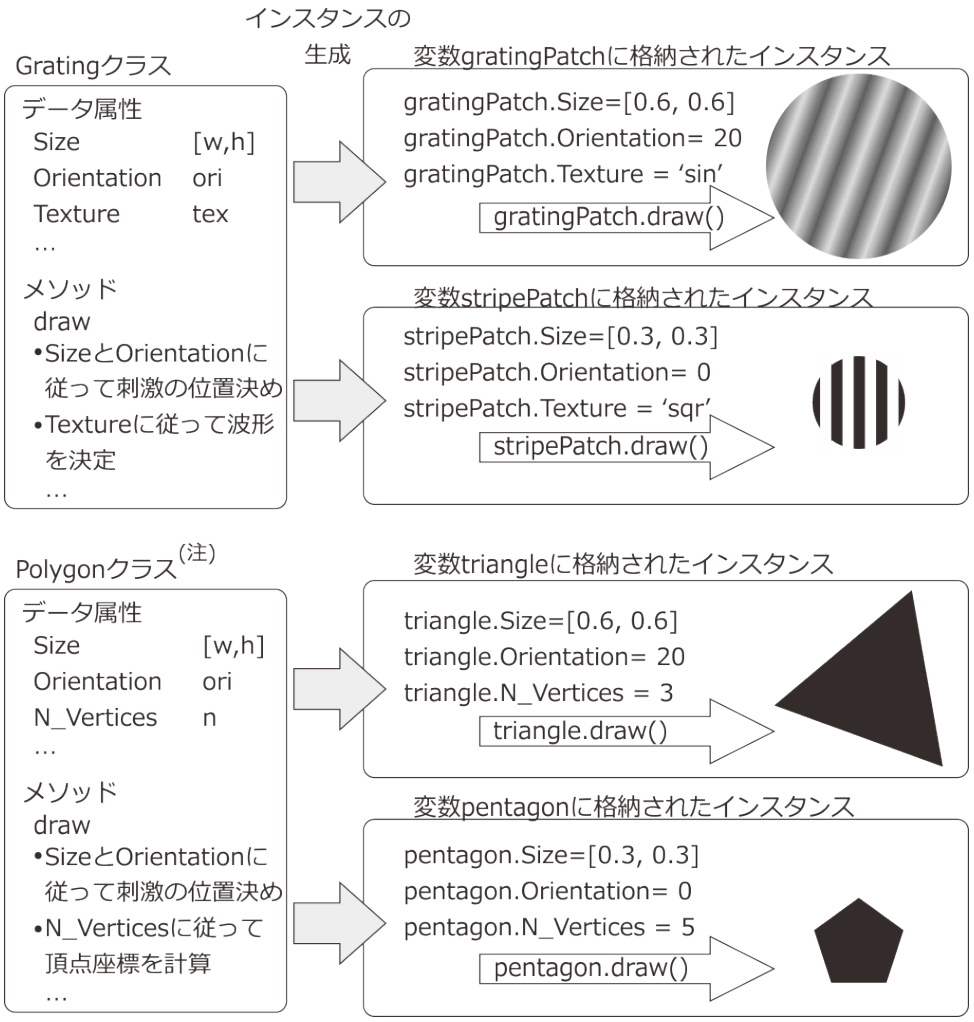


図 6.9　データ属性とメソッド、クラスとインスタンス。データ属性の値は個々のインスタンスで異なります。メソッドを呼ぶと自分のクラスで定義されているメソッドが呼び出されるので、異なるクラス間で同名のメソッドがあっても混乱しません。

は Polygon クラスのオブジェクトの draw メソッドであることが Python にはわかります。ですから、Grating クラスに同名のメソッドがあっても Python が両者を混同することはありません。

なお、ここで想定した Grating クラスや Polygon クラスおよびそのデータ属性は、実際の Builder で使用されているものと異なります。クラスおよびデータ属性の正確な名称および Builder との対応関係を「 6.9.5:Builder のコンポーネントと PsychoPy のクラスの対応 」に記しておきますので、詳しく学びたい方はご覧ください。Builder の通常の用途ではそこまで知っておく必要はありません。

ここまで説明して、ようやく key_choice.keys という式の意味を解説できます。key_choice には BuilderKeyResponse というクラスのインスタンスが格納されています。BuilderKeyResponse はキーボードの状態を記録するためのクラスで、 表 6.3 に示すデータ属性を持っています。この表によると、keys には押されたキー名が保持されています。key_choice という変数名は Builder の trial ルーチンに配置した Keyboard コンポーネントの **[名前]** に対応していますから、key_choice.keys という式は trial ルーチンに配置した Keyboard コンポーネントで記録したキー名です。具体的に言えば'left' か'right' のいずれかの文字列が格納されています。ですから、Code コンポーネントに入力した key_choice.keys == correctAns は実験参加者が押したキーのキー名が correctAns と一致すれば True、一致しなければ False が得られます。

表 6.3　BuilderKeyResponse クラスのデータ属性

データ属性	概要
keys	ルーチンで記録されたキー名が格納されています。キーが押されていなければ空のリスト、「最後のキー」または「最初のキー」を記録している場合は該当するキー名、「全てのキー」を記録している場合は押されたキーのキー名が順番に並んだリストが格納されています。
corr	反応が正答であれば 1、誤答であれば 0 が格納されています。キーが押される前は 0 です。
rt	キーが押された時刻が格納されています。キーが押されていなければ空のリスト、「最後のキー」または「最初のキー」を記録している場合は該当するキーが押された時刻、「全てのキー」を記録している場合は各キーが押された時刻が順番に並んだリストが格納されています。
clock	Builder が時刻を計測するための時計オブジェクトのインスタンスが格納されています。直接操作すべきではありません。

これで必要な要素はすべてそろいました。Builder で exp06.psyexp を開いて、trial ルーチンを開いて Code コンポーネントを配置してください。そして **[Routine 終了時]** に以下のコードを入力しましょう。

```
if key_choice.keys == correctAns:
    feedbackMsg = '正解'
else:
    feedbackMsg = '不正解'
```

ついに exp06.psyexp の完成です。さっそく exp06.psyexp を実行してください。実験情報ダイアログの condition には 01 とだけ入力し、word に「リニ」などの無意味語をターゲット語として入力して OK をクリックしてください。条件ファイルとして exp06cnd01.xlsx が読み込まれて、最初の教示文にはターゲット語が埋め込まれているはずです。そして、刺激が出てきたら適当に反応して、眼鏡をかけている顔画像に対して「はい」を選択する (カーソルキーの左を押す) と正解、それ以外の顔画像に対して「はい」を選択すると不正解になることを確認しましょう。実験が終了したらもう一度実験を実行して、今度は実験情報ダイアログの condition に 02 と入力して OK をクリックしてみましょう。眼鏡をかけている顔画像に「はい」を選択すると不正解、それ以外の画像に対して「はい」を選択すると正解になるはずです。

以上でこの章の解説はおしまいです。if 文についてはまだまだ説明したいことがたくさんありますが、この章もずいぶん長くなってしまったので、ここで一区切りにして次の章で取り扱うことにしましょう。この章で作成した実験は 第 8 章 で活用しますので残しておいてください。

チェックリスト

- クラスとインスタンスの違いを説明することができる。
- データ属性とは何かを説明することができる。
- 変数 x に格納されたインスタンスのデータ属性 foo に値を代入したり値を参照したりするときの Python の式を書くことができる。

- BuilderKeyResponse のデータ属性を参照して押されたキー名を知ることができる。

6.8 練習問題：データ属性 corr で正誤を判定しよう

この章の解説では、条件ファイルとして眼鏡をかけている顔画像が正事例となる exp06cnd01.xlsx と、眼鏡をかけていない顔画像が正事例となる exp06cnd02.xlsx の二種類しか作成しませんでした。しかし、実際に exp06.psyexp を使って概念識別の実験をするためには、他の特徴が正事例となる条件ファイルを準備する必要があります。練習問題として、「眼鏡をかけていて眉が上がっている (論理積)」顔画像が正事例となる条件ファイルと、「顔が丸い、または眉毛が下がっている (論理和)」顔画像が正事例となる条件ファイルを作成してください。

もうひとつ、if 文と BuilderKeyResponse の復習問題として、trial ルーチンの反応が正答であったか誤答であったかに応じて feedbackMsg に代入する文字列を変更する部分で、データ属性 keys を使わずにデータ属性 corr を使うように Code コンポーネントの内容を書き換えてください。 表 6.3 に示すように、データ属性 corr には反応が正答であれば 1、誤答であれば 0 が格納されています。これはほぼ答えを言っているようなものなので、これ以上はノーヒントで挑戦してください。

6.9 この章のトピックス

6.9.1 条件ファイルに完全なパスを書くべきか？

本文で述べた通り、第 6 版の改訂において条件ファイルの imageFile に完全な相対パスを書くように変更したのですが、これは Builder のオンライン実験作成機能を使用する際のことを考慮して決定しました。Builder で作成するオンライン実験では、実験で使用する画像ファイルや音声ファイル、条件ファイル等のリソースのすべての位置が JavaScript のコードに埋め込まれた形で出力されます。したがって、Builder はコードのコンパイル時に使用されるリソースをすべて把握しなければいけません。実験内に配置されているコンポーネントやループが直接参照しているリソースは自動的に把握されますし、把握された条件ファイル内で参照しているリソースもある程度は自動的に把握されますが、旧版の条件ファイルのように、条件ファイル内では Face00000.png としか書いていなくて実行時に初めて完全なパスとなるリソースはまず把握されません。把握されなかったリソースは、実験設定ダイアログの「オンライン」タブにある **[追加リソース]** に手作業で追加する必要があります。以上のことから、実行前に使用することが確定しているリソースは完全なパスで入力しておく習慣をつけておいた方が良いと思います。

オンライン実験のことを考慮しないなら、旧版のように条件ファイルに Face00000.png などとファイル名のみ書いて、Image コンポーネントの **[画像]** で $'image/'+imageFile といった式でパスを合成しても何も問題ありません。

6.9.2 改行文字を使った複数行の文字列の表現 (上級)

Microsoft Word などの文書作成アプリケーションを使ったことがある方は、この種のアプリケーションでは 図 6.10 文の末尾に矢印のような記号が (しばしば本文とは異なる色で) 描かれていることに気付いておられると思います。実は、アプリケーションの内部では、この「矢印」もひらがなや漢字、英数字と同じ「文字」の一種で、ここで「行が変わる」ことを表しています。文字として扱われている証拠（？）に、普通の文字と同様に BackSpace キーや Del キーで削除できますし、削除したら前後の文が 1 行につながりますよね。この「行が変わる」ことを表す文字を **改行文字** と呼びます。「 3.12.5:$を含む文字列を提示する 」で「コンピュータにおいて文字は『文字コード』と呼ばれる数値で表される」と書きましたが、改行文字にも数値が割り当てられ

ています。例えば Windows では 0x0D0A、Linux では 0x0A です。

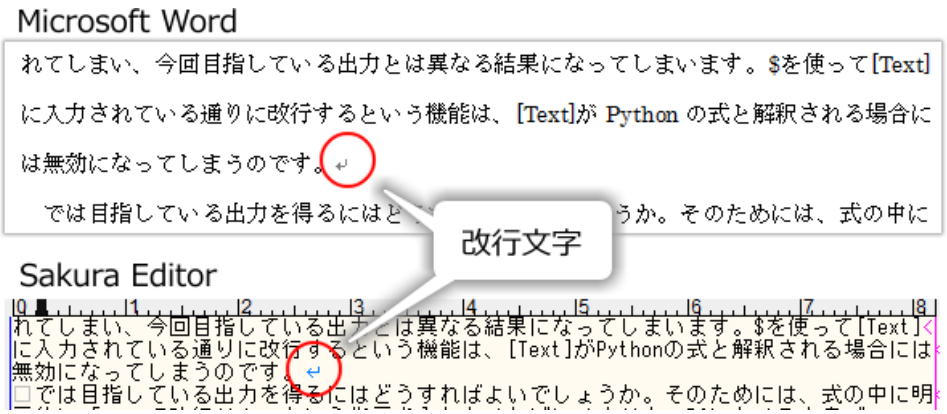


図 6.10　改行文字の表示例。行末の矢印のような記号が改行文字です。

Python を含む多くのプログラミング言語では、この改行文字をエスケープシーケンス (「3.12.5:$を含む文字列を提示する」参照) を使って\n と表すことができます。つまり

```
' 当てはまるならカーソルキーの左、\n 当てはまらないなら右を押してください。'
```

と書くと、

> 当てはまるならカーソルキーの左、
> 当てはまらないなら右を押してください

のように\n の位置で改行されて出力されます。\n を使う方法を知っていると、インターネット上で誰かが書いたプログラムを参考にして勉強するときなどにきっと役に立ちますのでぜひとも覚えておきたいところです。

さて、本章の教示も\n を使って記述することができますが、一行に書くには長すぎます。このような場合、Python の文法には

- 行末が \ のみで終わる場合は次の行につながっているとみなす
- ' 今日は' ' いい天気' のように二つの文字列の間に空白文字しかない場合は自動的に接続して ' 今日はいい天気' と同じとみなす

という規則があることを利用して

```
'「' + expInfo['word'] + '」は人の顔を形容する言葉です。\n' \
' 提示された顔の絵が当てはまるならカーソルキーの左、\n' \
' 当てはまらないなら右を押してください。'
```

と書くことができます。\は最後の行には不要（というか書いてはいけない）点にご注意ください。\n は各行の末尾以外に置いても構わないので、

```
'「' + expInfo['word'] + '」は人の顔を形容する言葉です。\n 提示された顔の絵が' \
' 当てはまるならカーソルキーの左、\n 当てはまらないなら右を押してください。 '
```

と書くこともできます。上の例とじっくり見比べて、同じ結果になることを確認してください。

6.9.3 True と False の「値」

Python では、if 文に True や False 以外の値を返す式を書くことができます。その場合、式を評価した結果が 0 であれば False、0 以外であれば True として処理されます。ですから、 図 6.11 上段に示した例ではいずれの場合も if 文に続く x=7 が実行されます。このことを積極的に利用したプログラムを書くことも可能ですが、わかりにくいのでお勧めしません。

お勧めしないと言えば、Python2 における True や False は、あたかも通常の変数であるかのように値を代入することができます。ですから、True=0 などと書いて True に 0 を代入することも可能です。ただ、このようにしてしまうと True を評価すると 0 が得られて、0 は False として機能するため、 図 6.11 下段の例のような非常にややこしい事態が生じてしまいます。True や False への代入は絶対に行うべきではありません。**Python3 では True と False が予約語となったため、このような代入はできなくなっています。**

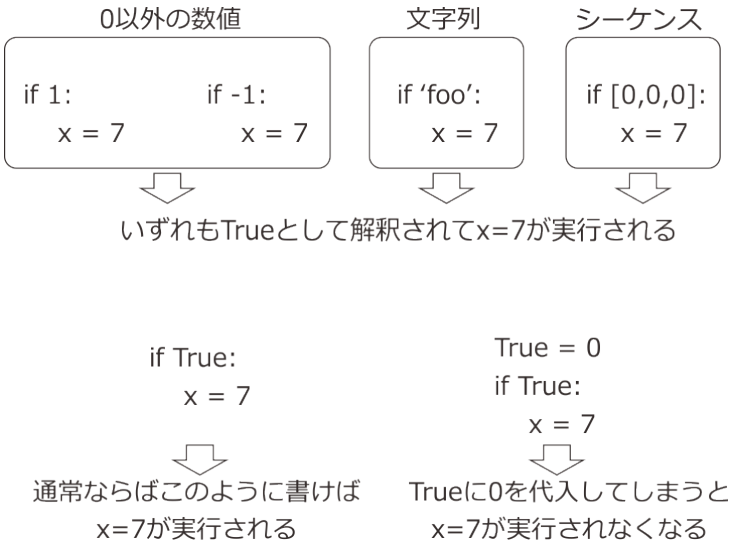


図 6.11　True と False に関する注意事項 (True/False への代入は Python2 のみ)。

6.9.4 文字列、シーケンスに対する比較演算子

本文では<や>=といった比較演算子を使って数値の大小を比較する方法について解説しましたが、Python では文字列やシーケンス型のデータに対しても比較演算子を使用することができます。これらのデータに対して比較演算子が適用された場合は「辞書順」に従って比較が行われます。

例えば memory と mind という文字列を比較するとしましょう。英和辞典の順番では、1 文字目から順番に比較していって、最初に異なる文字のアルファベット順でどちらが先に掲載されるかが決まります。memory と mind でしたら 1 文字目はどちらも m、2 文字目は e と i ですから、e の方が前です。Python では、辞書順で前にくる文字列ほど「小さい」と判定されますので、'memory' < 'mind' は True となります。'memory' > 'mind' は False です。

少し注意が必要なのは、数値や漢字が文字列に含まれる場合です。半角数字はすべてのアルファベットより「小さい」と判断されます。ですから'magical number' < '7' を評価すると False が得られます ('7' は'm' より小さいと判断される)。かな文字や漢字は文字コードに従って解釈されますので、文字コードを理解していなければ結果を予測するのは困難です。例えば Unicode で「心」は「記」より小さい値で表されますので（それぞれ 0x5fc3 と 0x8a18）' 記憶' < ' 心' を評価すると False が得られます。筆者の個人的な意見としては、このような比較は非常にわかりにくいので可能な限り使用するべきではないと思います。

シーケンスの場合は、要素を先頭から順番に比較していきます。[7,8,2] < [7,1,5,9] という比較でしたら、最初の要素の 7 は同一、2 番目の要素は 8 と 1 で左辺の方が大きい値ですので、左辺の方が大きいと判定されます。従って [7,8,2] < [7,1,5,9] は False です。シーケンスの要素に文字列が含まれている場合も、同様に個々の要素を先頭から順番に比較します。['theory', 7, 'mind'] > ['theory', 7, 'memory'] でしたら最初と 2 番目の要素は同一、3 番目の要素は memory より mind の方が「大きい」のでしたから True が得られます。

6.9.5 Builder のコンポーネントと PsychoPy のクラスの対応

図 6.12 に Builder の Grating コンポーネントと、それに対応する PsychoPy のクラスを示します。Grating コンポーネントに対応するクラスは psychopy.visual.GratingStim という名称 (以下 GratingStim) です。Grating コンポーネントの各プロパティは GratingStim クラスのデータ属性と対応しています。Builder において各プロパティに式や値を設定するという作業は、対応する GratingStim クラスのデータ属性にそれらを設定することと同義です。自分で実験用の Python コードを書く場合は、GratingStim クラスのデータ属性名やその設定方法を覚えなければいけません。Builder はプロパティ設定ダイアログという形でプロパティの一覧を見て設定ができるので、コードを書く方法を覚えるよりも手軽に自分の実験を作成できる段階まで学習できます。これが Builder を利用する最大の利点です。

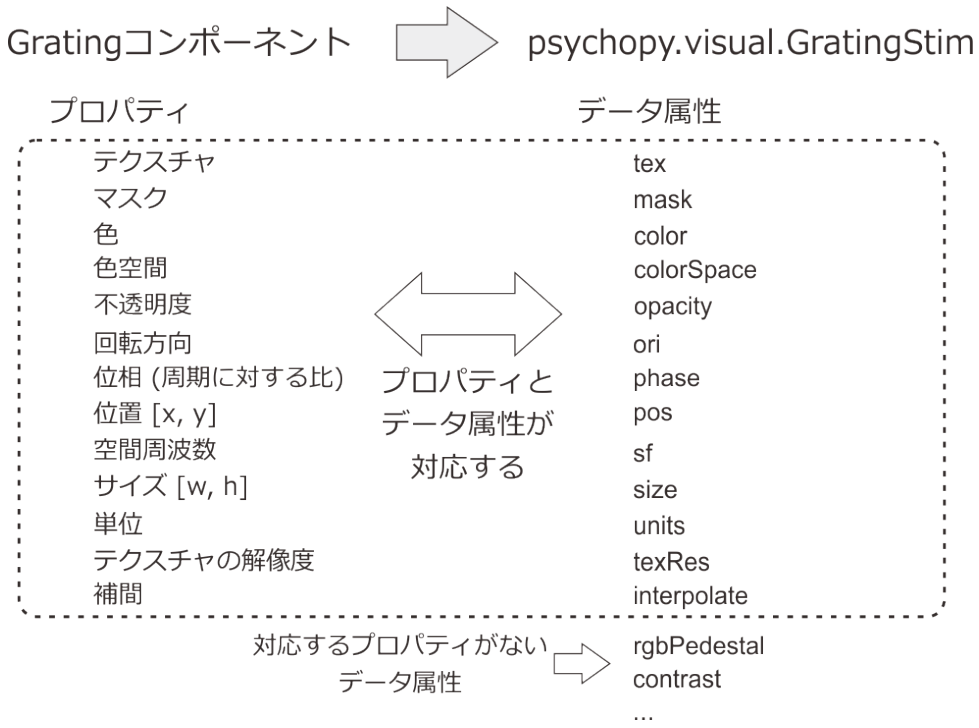


図 6.12　Builder の Grating コンポーネントのプロパティと、それに対応する PsychoPy のクラスのデータ属性。

ただし、の下の方に対応するプロパティが存在しないデータ属性があるように、Builder では GratingStim クラスの全てのデータ属性やメソッドを利用することができません。PsychoPy の機能を最大限に活かすためにはやはりコードを書く必要があります。

表 6.4 は、Polygon コンポーネントと PsychoPy のクラスの対応関係を示しています。Polygon コンポーネントの場合は、 **[形状]** に応じて最も効率がよいクラスを Builder が選択します。クラスによってデータ属性が異なりますので、Polygon コンポーネントのプロパティとデータ属性の対応も選択されたクラスに応じて変化します。

Builder は、背後にある PsychoPy のクラスやそのデータ属性についての知識がなくても使用できるように設計されています。実際、第 5 章 までは PsychoPy のクラスについて触れずに解説を進めてくることができました。しかし、この章の実験のように、参加者の選択に応じて刺激などが動的に変化する実験を作成するためには、残念ながら現状の Builder では PsychoPy のクラスについて言及せざるを得ません。

表 6.4　Builder の Polygon コンポーネントに対応する PsychoPy のクラス

形状	対応するクラス
直線	psychopy.visual.Line
三角形	psychopy.visual.ShapeStim
長方形	psychopy.visual.Rect
十字	psychopy.visual.ShapeStim
星	psychopy.visual.ShapeStim
正多角形...	psychopy.visual.Polygon

第 7 章

キーボードで刺激を調整しよう―ミューラー・リヤー錯視

7.1 この章の実験の概要

この章では有名な錯視のひとつであるミューラー・リヤー錯視を題材として、調整法の手続きを Builder で実現する方法を解説します。この章まで進んできた皆さんはそろそろ教示画面の作成は各自でできるでしょうから、重要な部分だけを取り上げましょう。 図 7.1 に実験に用いる刺激を示します。スクリーン上に左右に並んでテスト刺激とプローブが表示されます。テスト刺激はミューラー・リヤー錯視図形で、水平線 (以下主線と呼びます) の長さは 0.2、矢羽の長さは 0.05 です。主線と矢羽のなす角度 (以下夾角と呼びます) として 0 度から 30 度間隔で 150 度まで、6 種類の図形を用います。0 度の時は矢羽と主線がぴったり重なって長さ 0.2 の水平線だけに見える点に注意してください。プローブは水平な線分で、キーボードのカーソルキーの左右を使って長さを調節することができます。実験参加者は主線とプローブの長さが同じに見えるようにプローブの長さを調整して、スペースキーを押して報告します。この時のプローブの長さと主線の長さの差で錯視量を評価しようというのが本実験の狙いです。試行開始時のプローブの長さが毎回同じだと、参加者が何回キーを押せば主線とプローブが同じ長さになるかを学習してしまう恐れがありますので、試行開始時のプローブの長さは 170、190、210、230pix の中から無作為に選択します。テスト刺激、プローブともスクリーンの中央の高さで、水平方向の中心がスクリーン中央から 0.2 離れた位置に提示されるものとします。

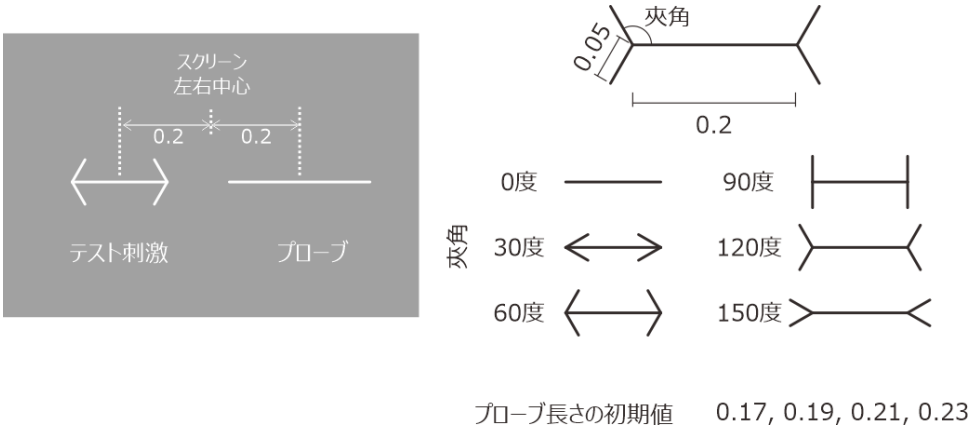


図 7.1　実験に用いる刺激。テスト刺激の水平線 (主線) の長さを 0.2、矢羽の長さを 0.05 で固定し、夾角を 0 度から 150 度まで変化させます。参加者は主線の長さとプローブの長さが等しく見えるようにプローブの長さを調整します。試行開始時のプローブの長さは 170、190、210、230pix のいずれかです。

図 7.2 に実験手続きを示します。教示画面は省略しましたので、いきなり最初の試行から始まります。各試行の最初に 0.5 秒間空白のスクリーンを提示した後、テスト刺激とプローブを提示します。テスト刺激は半数の試行で右側に、残り半数の試行で左側に提示され、順番は無作為に決定します。実験参加者がカーソルキーの右を押したら刺激の長さを 5pix 長く、左を押したら 5pix 短くします。参加者がスペース―を押したら試行は終了です。テスト刺激 (6 種類) ×テスト刺激の位置 (2 種類) ×プローブの初期長さ (4 種類)=48 通りの条件を 3 試行ずつ、合計 144 試行を行ったら実験は終了です。

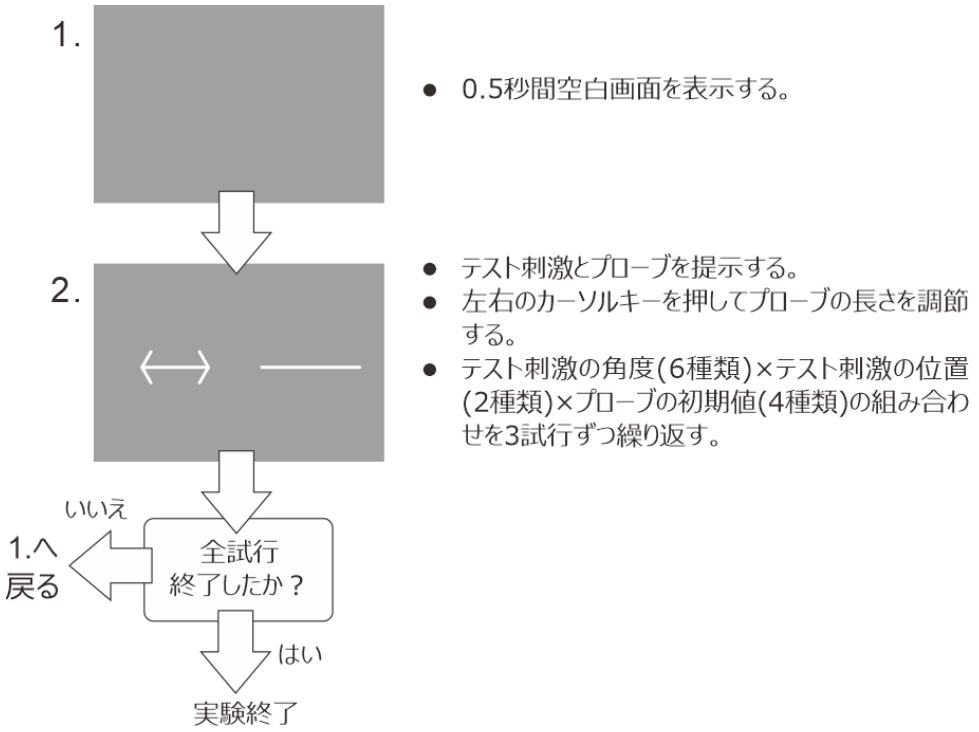


図 7.2　実験の流れ。

実験手続きは単純なので、前章までに解説したテクニックで十分実現できるはずです。問題は、「カーソルキーで刺激を調節してスペースキーで試行を終了する」という手続きをどうやって Builder で実現するかです。Keyboard コンポーネントでは、**[Routine を終了]** プロパティを用いてキーが押されたときにルーチンを終了させるか否かを指定できます。しかし、特定のキーが押されたときだけ終了させるといった指定はできません。第 6 章 で学んだ if 文を使うとキーに応じた処理を実行することが出来ます。

それでは実験を作成していきましょう。以下の解説では、Builder で実験を新規作成し、以下の作業を行って exp07a.psyexp の名前で保存したものとします。 **この章から既出のコンポーネントのプロパティについてはタブを省略しますのでご注意ください** 。

準備作業

- この章の実験のためのフォルダを作成して、その中に exp07.psyexp という名前で新しい実験を保存する。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
  - 「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。

- trial ルーチン

  ~~P~~olygon コンポーネントを 1 つ配置し、 **[名前]** を `testline` として以下のように設定する。

  ∗**PsychoPy 2024 以降の場合**

  - 「基本」タブの **[開始]** を「時刻 (秒)」の `0.5` とし **[終了]** を空白にして **[形状]** を直線にする。
  - 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.2, 0)` とする。 **[位置 [x, y] $]** を `(testPos, 0)` にして、「繰り返し毎に更新」に設定する。
  - 「外観」タブの **[枠線の色]** を `white` にする (標準で white のはずである)。 **[枠線の幅]** を 2 にする。

  ∗**PsychoPy 2024 より前のバージョンの場合**

  - 「基本」タブの **[開始]** を「時刻 (秒)」の `0.5` とし **[終了]** を空白にして **[形状]** を長方形にする。
  - スクリーンの解像度から、数 pix 程度の高さになる値を height 単位で計算する。この値を X として、「レイアウト」タブの **[サイズ [w, h] $]** を `(0.2, X)` とする。具体例を挙げると、1920 × 1080 のスクリーンであれば height 単位で 1.0 = 1080pix であり、X=0.002 なら 1080 × 0.002=2.16pix となりほぼ 2pix となるので、 **[サイズ [w, h] $]** を `(0.2, 0.002)` とする。
  - 「レイアウト」タブの **[位置 [x, y] $]** を `(testPos, 0)` にして、「繰り返し毎に更新」に設定する。
  - 「外観」タブの **[塗りつぶしの色]** を `white` にする (標準で white のはずである)。 **[枠線の色]** を `None` にする。

  ~~t~~estline をコピーして `probe` の名前で貼り付けて以下のように設定する。

  ∗ 「レイアウト」タブの **[サイズ [w, h] $]** を `(probeLen, 0)` にして、「フレーム毎に更新する」を設定する。 **[位置 [x, y] $]** を `(-testPos, 0)` にする。

  ~~t~~estline をコピーして `arrowTL` の名前で貼り付けて以下のように設定する (TL=Top Left)。

  ∗ 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.05, 0)` とする。 **[位置 [x, y] $]** を `(testPos-0.1, 0)` にして **[位置揃え]** を中央左にする。 **[回転角度]** を `-angle` にして「繰り返し毎に更新」に設定する (angle にマイナス記号がついていることに注意)。

  ~~a~~rrowTL をコピーして `arrowBL` の名前で貼り付けて以下のように設定する (BL=Bottom Left)。

  ∗ 「レイアウト」タブの **[回転角度]** を `angle` にする (angle にマイナス記号がついてない注意)。

  ~~a~~rrowBL をコピーして `arrowTR` の名前で貼り付けて以下のように設定する (TR=Top Right)。

  ∗ 「レイアウト」タブの **[位置 [x, y] $]** を `(testPos+0.1, 0)` にして **[位置揃え]** を中央右にする。

arrowTR をコピーして arrowBR の名前で貼り付けて以下のように設定する (BR=Bottom Right)。

* 「レイアウト」タブの **[回転角度]** を -angle にする (angle にマイナス記号がついていることに注意)。

– ここまでの作業で testline, probe, arrowTL, arrowBL, arrowTR, arrorBR の 6 個の Polygon コンポーネントが配置されることになる。Builder のメニューの「実験」の「実験内を検索...」から angle と testPos を検索して、 図 7.3 のように設定されていることを確認する。

Code コンポーネントを 1 つ配置して、以下の設定を行う。

* **[Routine 開始時]** に probeLen = initProbeLen と入力する。
* trial ルーチン内での順序を一番上にする。

Keyboard コンポーネントを 1 つ配置して、以下の設定を行う。

* **[名前]** を key_response にする。
* **[開始]** を「時刻 (秒)」の 0.5 とし、 **[終了]** を空白にする。
* **[Routine を終了]** のチェックを外す。
* **[検出するキー $]** を 'left','right','space' にする。
* 「データ」タブの **[記録]** を「なし」にする。
* trial ルーチン内での順序を一番上にする。結果として **Keyboard ルーチン、Code コンポーネント、その他の Polygon コンポーネントの順に並んでいる** ことを確認すること。

- trials ループ (作成する)
  – **[Loop の種類]** を fullRandom にする。
  – **[繰り返し回数 $]** を 3 にする。
  – **[条件]** に exp07cnd.xlsx と入力する。
- exp07cnd.xlsx(条件ファイル)
  – testPos、angle、initProbeLen の 3 パラメータを設定する。
  – 実験手続の内容を満たすように、2 種類の testPos(-0.2、0.2)、6 種類の angle(0、30、60、90、120、150)、4 種類の initProbeLen(0.17、0.19、0.21、0.23) の組み合わせを入力する。2 × 6 × 4=48 行の条件ファイルとなる (1 行目のパラメータ名を除く)。testPos はテスト刺激の中心の X 座標を表しており、プローブ刺激の位置は testPos の符号を反転すれば得られるのでパラメータとして用意する必要はない。

教示などを省略したので以上で単純なフローの実験となりました。また、古いバージョンの PsychoPy では Polygon コンポーネントの **[位置 [x, y] $]** は図形の中心を指していなければいけなかったので複雑な計算が必要だったのですが、 **[位置揃え]** が選択できるようになって非常に楽になりました。ただし、PsychoPy 2024 より前のバージョンでは線分を描画する際の **[位置揃え]** の処理がおかしいので非常に細長い長方形を使って線分を描画しています。

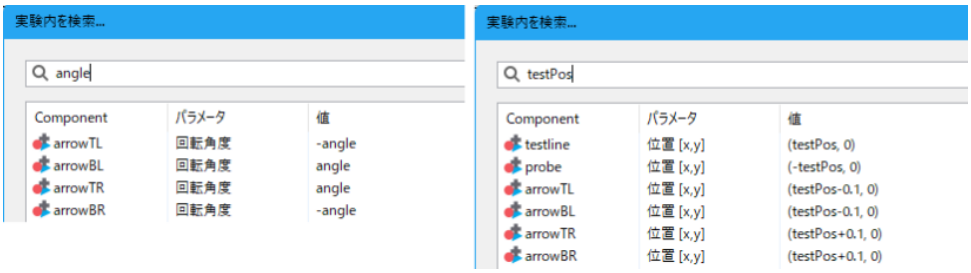


図 7.3　Polygon コンポーネントの設定が複雑で入力ミスしやすいが、「実験内を検索...」で angle や testPos を検索すると確認しやすい。

現状のままでは trial コンポーネントに置いた Keyboard コンポーネントは何の機能も持っていませんが、これから Code コンポーネントを使ってプローブ長の調整や決定の処理を組み込んでいきましょう。

7.2 Code コンポーネントを使って刺激のパラメータとルーチンの終了を制御しよう

Code コンポーネントでは、押されたキーを判別してカーソルキーの右であればプローブの長さを 5pix 長く、左であれば 5pix 短くし、スペースキーであればルーチンを終了します。第 6 章 を読んだ皆さんであれば「if 文を使うとよい」ということはすぐにわかると思いますが、今回は処理が複雑です。第 6 章 では

1. 押されたキーの名前が変数 correctAns の値と一致している
2. 押されたキーの名前が変数 correctAns の値と一致していない

の 2 通りに分岐しましたが、今回は

1. 押されたキーの名前が'left' である
2. 押されたキーの名前が'right' である
3. 押されたキーの名前が'space' である
4. 押されたキーの名前がいずれにも一致しない (つまりいずれも押されていない)

の 4 通りに分岐しなければいけません。このように 3 通り以上の分岐を処理するために、Python の if 文では elif という語を使うことができます。elif を使った if 文の書式は以下の通りです。n 個の式を連ねて書くことができます。最初は if、最後は else で、それ以外はすべて elif でなければいけません。

```
if 式 1:
    式 1 が真の時の処理
elif 式 2:
    式 1 が偽で式 2 が真の時の処理
（中略）
elif 式 n:
    式 1 から式 n-1 が偽で式 n が真の時の処理
else:
    式 1 から式 n がすべて偽であった時の処理
```

最初に真になった式に対応する処理だけが実行されますので、例えば式 1 が偽で式 2 が真であれば、「式 1 が偽で式 2 が真の時の処理」だけが実行されます。その後に続く式 3、式 4…が真であっても、対応する処理は一切実行されません。

押されたキーの名前が変数 key に入っているとすれば、今回の処理は以下のように書けます。上記の書式で n=3 の場合に該当します。

```
if key == 'left':
    プローブの長さから 0.005 を引く
elif key == 'right':
    プローブの長さに 0.005 を足す
elif key == 'space':
    ルーチンを終了する処理
```

最後の else はどこへいった？と思われるかもしれませんが、else に相当するのは「カーソルキーの左、右、スペースキーのいずれも押されていない場合」で、今回の実験ではこの場合なにもコードを実行する必要がありません。 **else 節で何もすることがない場合、else 節は省略されます。**

プローブの長さを変更する処理については、プローブ刺激に対応する Polygon コンポーネント probe の **[サイズ [w, h] $]** に (probeLen, 0) と書いているのですから、変数 probeLen の値を増減すればいいだけです。0.005 を加えるには `probeLen += 0.005` 、0.005 を引くには `probeLen -= 0.005` です。

ルーチンを終了させる処理については、まだ解説していない Builder の機能を使用する必要があります。Builder には、1 フレーム描画する毎に 1 回、continueRoutine という変数を確認し、値が False であれば直ちにルーチンを終了するという機能があります。if 文を用いて、ルーチンを終了させたい条件を満たした時に continueRoutine=False という文を実行すれば、ルーチンを終了させることができるわけです。

以上を踏まえて、if 文の処理内容を記述すると以下のようになります。

```
if key == 'left':
    probeLen -= 0.005
elif key == 'right':
    probeLen += 0.005
elif key == 'space':
    continueRoutine = False
```

残るは押されているキー名の取得です。第 6 章 の内容を踏まえると、Keyborad コンポーネントの **[名前]** が key_response ですから、key_response のデータ属性 keys を利用すればよいだけのような気がします。しかし、表 6.3 をよく読みなおしてほしいのですが、keys には押されたキー名が格納されているのではなく、そのルーチンで Keyboard コンポーネントが保存するキー名が格納されています。今回の実験では key_response の **[記録]** を「なし」に設定しているのですから、データ属性 keys にはキー名が格納されません。

ではどうにすれば良いかといいますと、Builder が自動的に用意する変数 theseKeys を利用します。theseKeys は最後に実行した Keyboard コンポーネントの結果を格納している変数です。 **[検出するキー $]** に記述されたキーがいずれも押されていなければ空の (要素数ゼロの) リスト、押されていたキーがあれば、それらの名前をすべて列挙したリストが格納されています。theseKeys と複数形になっているのは、同時に複数のキーが押さ

れる場合があるからです。キーの同時押しについての詳細は「7.6.1: 複数キーの同時押しの検出について」を参考にしてください。

theseKeys を利用するにあたって、ポイントが 2 つあります。まず、theseKeys は「最後に実行した Keyboard コンポーネントの結果」を格納するものですから、ルーチンが開始された直後にはまだ theseKeys という変数自体が存在しません。そこで、Code コンポーネントを使って **[Routine 開始時]** に theseKeys を用意しておく必要があります。まだキーは押されていないのですから、空のリストを代入しておけばよいでしょう。以下のコードを trials ルーチンの **[Routine 開始時]** に追加しておいてください (すでに `probeLen = initProbeLen` という文が入力されているはずです)。

```
theseKeys = []
```

第 2 のポイントは、押されたキーの確認方法です。theseKeys の中身はリストですから `theseKeys == 'left'` という具合に直接文字列と比較するのではなく、theseKeys に格納されたリストの中に'left' という文字列があるかを検査しなければいけません。Python にはこの用途にうってつけの in という演算子があります。in 演算子は x in a という形で使用して、a の中に x があれば True、なければ False となります。theseKeys と in 演算子を用いて if 文を書くと以下の通りになります。

```
if 'left' in theseKeys:
    probeLen -= 0.005
elif 'right' in theseKeys:
    probeLen += 0.005
elif 'space' in theseKeys:
    continueRoutine = False
```

これで if 文は完成しましたが、問題はこの if 文を Code コンポーネントのどこへ入力すれば良いのかという点です。ここで覚えておいてほしいのが、「あるルーチンを実行している最中に何かをするのであれば、コードを入力する欄は **[フレーム毎]** でなければいけない」ということです。今回のコードはルーチンの実行中に押されたキーを判別して処理を振り分けるのですから、 **[フレーム毎]** に入力しなければいけません。タイプミスしないように気を付けて **[フレーム毎]** にこの if 文を入力してください。left や right、space の前後のシングルクォーテーションや、if、elif が出て来る最後のコロン、if、elif 以外の行は行頭にスペースが必要な点などが間違えやすいポイントです。

以上で Code コンポーネントによる刺激パラメータとルーチン終了の制御ができるようになりました。作業内容を exp07.psyexp に保存して実行してみましょう。カーソルキーの左右を押すとプローブの長さが変化し、スペースキーを押すと次の試行へ進むはずです。残念ながらカーソルキーを押しっぱなしにしていても連続的に長さは変化しませんので、何度もカチカチとボタンを押して長さを調節する必要があります。

一見、これで 図 7.1 および 図 7.2 に示した実験が完成したように思えます。しかし、実験を最後まで実行するか Esc キーで中断して trial-by-trial 記録ファイルを確認してみるとわかりますが、probeLen の列に試行開始時の値が出力されていて、参加者が調整後の probeLen の値が記録ファイルのどこにも出力されていません。Builder は、Code コンポーネントで独自に使用した変数や、ルーチン開始後に変更されたパラメータの値を記録ファイルには出力しないのです。自動で出力してくれたらいいのにと思われるかもしれませんが、本当にそんなことをしたら記録ファイルが分析に不要な変数だらけで大変なことになりかねません。probeLen の値を出力させるためには以下の 2 つの方法があります。。

1. probeLen が保存すべき変数であることを Builder に教えるために Variable コンポーネントというコンポーネントを使用する。
2. Code コンポーネントを使って probeLen を実験記録ファイルに出力するコードを追加する。

筆者のお勧めは 2 の Code コンポーネントを使う方法です。次節でこちらの方法を解説します。1 の方法については「 7.6.4:Variable コンポーネントによる変数の値の出力 」を参考にしてください。

チェックリスト

- 3 通り以上の分岐を処理させる if 文を書くことができる。
- リストの中にある要素が含まれているか否かで処理を分岐させることができる。
- Code コンポーネントからルーチンを終了させることができる。

7.3 Code コンポーネント使って独自の変数の値を記録ファイルに出力しよう

Builder が実験記録ファイルに出力する変数はどのように管理されているのでしょうか。ここで思い出していただきたいのは、条件ファイルで定義した変数はすべて実験記録ファイルに出力されるという点です。条件ファイルはループのプロパティ設定ウィンドウで指定するのですから、実験記録ファイルに出力される値を管理しているのはループであるはずです。

では、Builder におけるループの「実体」とは何でしょうか。その答えは **[Loop の種類]** によって異なるのですが、この本で使用している random、sequential、fullRandom の場合はいずれも psychopy.data.TrialHandler (以下 TrialHandler) というクラスのインスタンスです。TrialHandler はその名が示す通り、試行を制御するためのクラスです。繰り返しの度にパラメータの値を更新したり、パラメータや反応を実験記録ファイルに保存したりする Builder の機能はこのクラスによって実現されています。

表 7.1 に TrailHandler の主なデータ属性を示します。これらのデータ属性を利用すると、画面上に「現在第 n 試行」、「残り n 試行」といったメッセージを提示することができます。例えば trials という名前を付けたループに対応する TrailHandler のインスタンスは Coder 上では変数 trials に格納されているので、trials.thisN と書くと現在 trials ループで何回繰り返しを終えているかが得られます。 表 7.1 に書かれているように 1 回目の繰り返しでは thisN=0 なので、1 を加えて trials.thisN+1 とすれば「第 n 試行」の n に当てはめる数値が得られます。

表 7.1　TrialHandler の主なデータ属性

データ属性	概要
thisIndex	条件ファイルの何行目の条件がこのループで用いられているかを示します。ただしパラメータ名の行は行数に数えず、1 行目を 0 と数えます。Trial-by-trial 記録ファイルの thisIndex と同じです。
nTotal	このループで実行される繰り返しの総数。
nRemaining	このループで実行される残りの繰り返し回数。1 回目の繰り返しの実行中に nTotal-1 で、繰り返しの度に 1 ずつ減少します。
thisN	このループで実行済みの繰り返し回数。1 回目の繰り返しの時に 0 で、繰り返しの度に 1 ずつ増加します。Trial-by-trial 記録ファイルの thisN と同じです。
thisRepN	Trial-by-trial 記録ファイルの thisRepN と同じです。
thisTrialN	Trial-by-trial 記録ファイルの thisTrialN と同じです。

ちょっと脱線なのですが、第 5 章 の復習がてら実際に「第○試行」と Text コンポーネントを使って提示する場合の注意点を述べておきましょう。trials.thisN+1 は数値ですので、そのまま文字列と結合することができません。ですから Text コンポーネントの **[文字列]** に$' 第' + (trials.thisN+1) + ' 試行' と書くと当然エラーになります。ここで 第 5 章 において実験情報ダイアログからターゲットの大きさや偏心度の値を得たときの処理を思い出してください。あの時は実験情報ダイアログの値は文字列で、刺激のパラメータとして利用する時には数値に変換しないといけないのでした。今回はこの逆で、数値である trials.thisN+1 を文字列にしないといけません。第 5 章 で紹介したように、この変換には関数 str( ) を用います。$' 第' + str(trials.thisN+1) + ' 試行' とすれば、Text コンポーネントの Text の値として使用できます。

話を元に戻しましょう。TrialHandler クラスのインスタンスに実験記録ファイルへ出力する変数を追加するには、TrialHandler のメソッドを使用する必要があります。 表 7.2 に主な TrialHandler のメソッドを示します。メソッドの引数の書き方が PsychoPy のヘルプドキュメントと異なりますが、この点について解説するには Python の文法に関する詳しい解説が必要です。興味がある方は「 7.6.2: メソッドの第一引数について (上級) 」をご覧ください。 表 7.2 の最初に挙げられている addData( ) が今回の目的を達成するためのメソッドです。第 6 章 でも少しだけ例が出ていたのですが、メソッドはデータ属性と同様にインスタンスが格納されている変数とメソッド名をドット演算子で連結して呼び出します。trials ループに出力する変数を追加する場合は trials.addData( ) と書くわけです。 表 7.2 に書かれている通り、addData( ) には引数が必要です。第 1 引数には、trial-by-trial 記録ファイルや xlsx 記録ファイルに置いてその値が出力される列の名前 (記録ファイルの 1 行目の見出し) を文字列として渡します。第 2 引数には、出力したい変数や値を記述します。今回の例では、response という列名で実験参加者が調整した後の probleLen の値を出力してみることにしましょう。記入すべき文は以下の通りです。

```
trials.addData('response', probeLen)
```

この文を Code コンポーネントに追加すればいいのですが、どの欄に追加すればいいでしょうか。参加者がスペースキーを押して反応を確定した後に保存しないと意味がありませんから、**[Routine 終了時]** に追加するのが正解です。追加して変更を保存してください。

表 7.2　TrialHandler の主なメソッド。

メソッド, 概要	
addData(thisType, value)	実験記録ファイルに出力する値を追加します。thisType に実験記録ファイルにおける列名、value に値を指定します。
getEarlierTrial(n)	n 回前に用いられたパラメータを得ます。1 回前であれば n=-1 という具合に負の整数で指定します。n が省略された場合は n=-1 と見なされます。n 回前が存在しない (2 回目で-5 を指定するなど) 場合は None、存在する場合は n 回前のパラメータが辞書オブジェクトとして得られます。
getFutuerTrial(n)	n 回後に用いられるパラメータを得ます。1 回後であれば n=1 という具合に正の整数で指定します。n が省略された場合は n=1 と見なされます。n 回後が存在しない場合は None、存在する場合は n 回後のパラメータが辞書オブジェクトとして得られます。

保存したら実験を実行してみましょう。終了後に trial-by-trial 記録ファイルと xlsx 記録ファイルを開くと、図 7.4 のように response という名前の列が存在していて、そこに調整後の probeLen の値が出力されているのがわかります。xlsx 記録ファイルには Keyboard コンポーネントの出力と同様に平均値や標準偏差も出力されています。

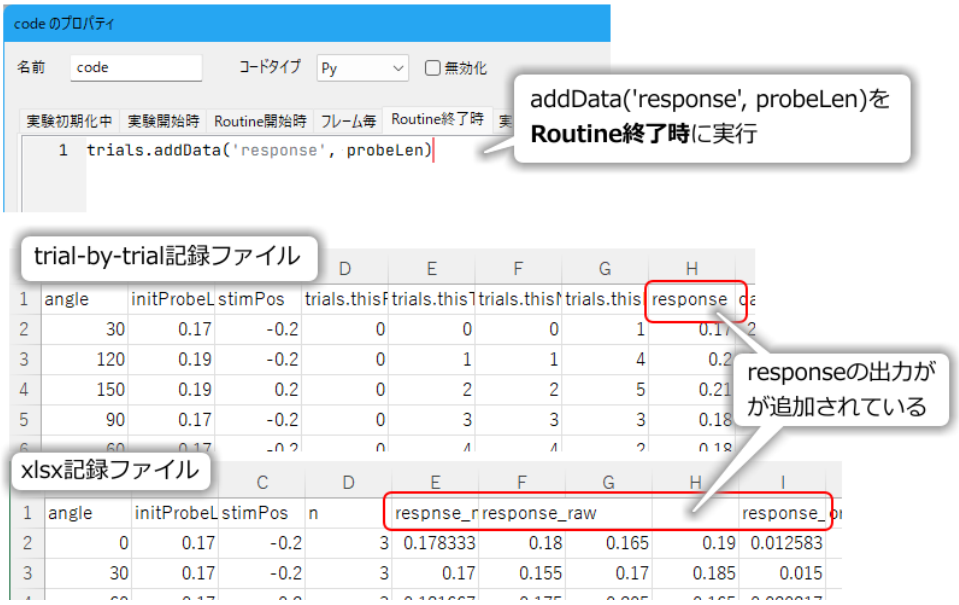


図 7.4　addData( ) メソッドによる変数の出力。

これでこの章の実験は完成です。ですが、せっかく 表 7.2 に addData( ) 以外の TrialHandler のメソッドを紹介しましたので、少し触れておきましょう。 表 7.2 に書かれている通り、TrialHandler には getEarlierTrial( ) と getFutuerTrial( ) というメソッドがあります。これらのメソッドを使うと、それぞれ現在のループの n 回前、および n 回後の繰り返しで条件ファイルから読み込まれたパラメータのどの値が用いられた（用いられる）かを知る事ができます。例えば trials ループの内部のルーチンに Code コンポーネントを配置して、**[Routine 開始時]**、**[フレーム毎]**、**[Routine 終了時]** のいずれかで以下の文を実行すると、2 回前の繰り返しで用いられたパラメータが変数 prevParam に格納されます。

```
prevParam = trials.getEarlierTrial(-2)
```

prevParam に格納されているのは辞書オブジェクトです。第 4 章 、第 5 章 でも触れたように、辞書オブジェクトとは実験情報ダイアログの値を保持するのにつかわれているデータ形式です。ですから、実験情報ダイアログと同様に、以下のように書くと angle というパラメータの値を取り出すことができます。

```
prevParam['angle']
```

記憶課題の一種に「左右の選択肢のうち、n 試行前に提示されていた刺激と一致する選択肢を選べば正解」という課題 (n-back 課題) がありますが、 **[Loop の種類]** に random や fullRandom を選んでいる場合、n 試行前に提示した刺激は無作為に決定されているので条件ファイルで正答を定義することができません。そのような場合に getEarlierTrial( ) メソッドは非常に有効です。

チェックリスト

- TrialHandler のインスタンスから現在ループの何回目の繰り返しを実行中かを取得できる。
- TrialHandler のインスタンスから現在ループの繰り返し回数が残り何回かを取得できる。
- TrialHandler のインスタンスから現在ループの総繰り返し回数を取得できる。
- 上記 3 項目の値を使って「現在第 n 試行」、「残り n 試行」、「全 n 試行」といったメッセージをスクリーン上に提示できる。
- Code コンポーネントを用いて、実験記録ファイルに出力するデータを追加することができる。
- 現在実行中の繰り返しの n 回前、n 回後に使われるパラメータを取得するコードを記述することができる。

7.4 プローブの長さが一定範囲に収まるようにしよう

すでにこの章で目的とする実験は完成しているのですが、if 文の練習を兼ねて少し改造してみましょう。exp07.psyexp では、実際にする人がいるかどうかは別として、テスト刺激に重なったりスクリーンからはみ出してしまったりするくらいプローブを大きくすることができてしまいます。また、プローブをどんどん小さくしていけばいずれ長さは 0 になり、負の値になってしまいます。長さは負の値をとることができませんので、そこで実験はエラーとなり停止してしまいます。このような事態を避けるために、if 文を使ってプローブの長さが 0.05 から 0.35 の範囲を超えて短くしたり長くしたりできないようにしてみましょう。

exp07.psyexp を Builder で開いて trial ルーチンの Code コンポーネントを開いてください。 **[フレーム毎]** に入力してあるコードによってプローブの長さが変わるのですから、ここのコードを書きかえると長さを一定の範囲に制限することができるはずです。ひとつの問題を解決するための方法を一度に何通りも紹介するのはよくないかも知れませんが、ここでは if 文の練習なので 2 通りの方法を考えます。

第一の方法は、長さを 0.005 増加させた時に 0.35 より大きくなっていないか確認して、なっていれば 0.35 に修正し、長さを 0.005 減少させた時に 0.05 未満になっていないか確認して、なっていれば 0.05 に修正するというものです。入力済みのコードに日本語で処理を書きこむと 図 7.5 の左のようになります。ご覧のとおり、if 文で分岐した後にまた「もし〜なら…」という分岐処理が含まれる形になっています。if 文では、このような「入れ子」になった条件分岐も書くことができます。「probeLen が 0.05 未満なら 0.05 にする」という部分

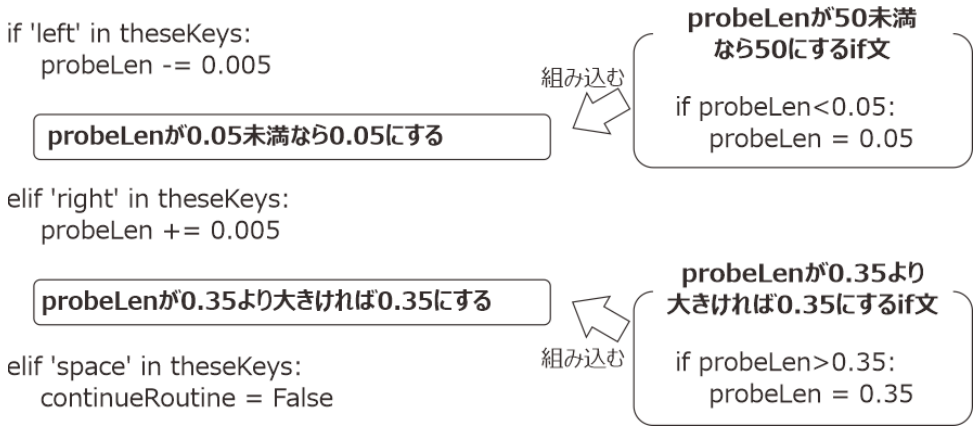


図 7.5　プローブの長さを制限する方法その 1。exp07.psyexp の Code コンポーネントに書いたコードに組み込む処理を日本語で記入したものを左側に、組み込む処理に対応するコードを右側に示しています。

だけを考えると、これは 0.05 以上なら何もしないということですから、else は省略できて 図 7.5 右上のように書けます。同様に「probeLen が 0.35 より大きければ 0.35 にする」という処理も 図 7.5 右下のように書けます。 図 7.5 左側のコードの日本語で記入した部分に、 図 7.5 右側の対応するコードを埋め込むと、 図 7.6 左に示すコードが得られます。これだけで完成です。

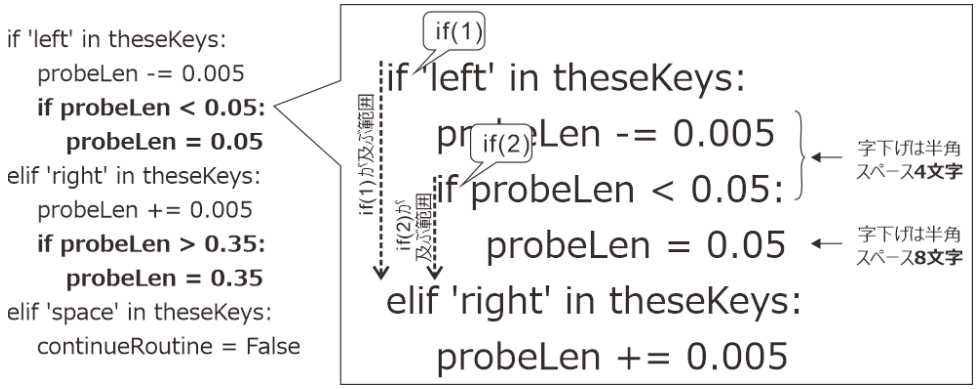


図 7.6　if 文を組み込んで得られたコード。if、elif、else の及ぶ範囲は、下方向に向かってこれらの語が出現した行と字下げ幅が狭いか同じ行に出会うまでです。if (2) が半角スペース 4 文字字下げされているので、if (2) の次の行は if (2) よりさらに 4 文字字下げして 8 文字字下げとなります。

図 7.6 左のコードをもう少ししっかり見ておきましょう。 図 7.6 左のコードの冒頭部分を拡大したのが 図 7.6 右です。if が及ぶ範囲は字下げの量で決まります。Python は if を発見すると、コードを下へ読み進めていって、これらの語が出現した行と字下げ量が同じか少ない行の直前の行までを if の条件式が及ぶ範囲と見なします。このことを念頭に置いて 図 7.6 右のコードを見ると、1 行目の if 文 (if (1) とします) の及ぶ範囲は 5 行目の elif 文の手前まで、すなわち 4 行目までであることがおわかりいただけると思います。if (1) の条件式が真であれば、4 行目までのコードが実行されます。一方、3 行目の if 文 (if (2) とします) の範囲はどこまでかと言いますと、これも 5 行目の elif 文の手前まで、すなわち 4 行目までです。ですから if (2) の条件式が真であれば、4 行目だけ実行されます。もちろん if (2) は if (1) の範囲に入っているので、if (2) が実行されるためには if(1) が真でなければいけません。if (1) が真で if (2) が偽であれば、2 行目だけが実行されます。elif、else が及ぶ範囲も if と同様に決まります。なお、本書では字下げをすべて 4 文字としているので 図 7.6 の説明で問題ないのですが、web 上で誰かが書いた Python のコードを流用する時にはそのコードが異なる字下げルールを使っているかもしれません。そのようなコードをコピーするときの注意点を「7.6.3:Python コードの字下げについて」に記しておきますので参考にしてください。

では、 図 7.6 左のコードを trial ルーチンの Code コンポーネントの **[フレーム毎]** に入力しましょう。すでに細字の部分は入力済みのはずなので太字部分を入力するだけでいいはずです。入力したら実験を保存して実行し、プローブの長さが一定以上伸びたり縮んだりしないことを確認してください。プローブ長が 0.05 や 0.35 に達するまでカーソルキーを連打するのが面倒くさい！という方は実験をいったん終了して、Code コンポーネントを編集してカーソルキーの左右を押したときに probeLen が増減する量を ±0.005 から ±0.05 などに変更して実行してみましょう。

続いて第二の方法の解説です。第一の方法では probeLen の長さを増減した後に範囲外に出てしまっていないかを確認しましたが、第二の方法では一連の if-elif-else が終わってから probeLen を確認します。 図 7.7 にこの方法を用いたコードを示します。ここでのポイントは、if に対応する elif、else が置ける範囲です。 図 7.7 に記した通り、1 行目の if 文 (if (1) とします) に対応する elif、else が置けるのは、if (1) と字下げが同じで elif、else 以外から始まる行が出てくるか、if (1) より字下げが少ない行が出てくる直前の行までです。 図 7.7 のコードでは、2 つ目の if (if (2) とします) が出現した時点で if (1) が終了していますので、if (1) から続く一連の if、elif の結果がどうであろうと必ず if (2) の条件式は評価されます。elif で条件式を列挙した場合は手前の if や elif で真になった時に全く評価されなかったのと対照的です。

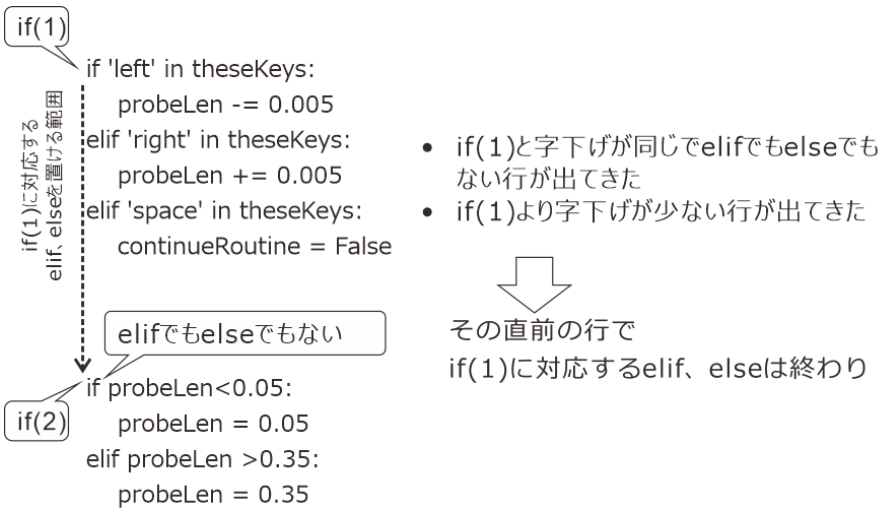


図 7.7　プローブの長さを制限する方法その 2。if に対応する elif、else を置ける範囲に注意。

exp07c.psyexp は exp07a.psyexp を別名で保存して作成したので、trial ルーチンの Code コンポーネントの **[フレーム毎]** に 図 7.7 のコードの if (2) の手前まですでに入力済みのはずです。そこへ、 図 7.7 のコードの最後の 4 行 (if (2) に対応する部分) を追加入力してください。continueRoutine = False という行と if (2) の最初の行の間は空白行を入れても入れなくても動作しますが、1 行空白を入れておいた方が後から見直した時にここで新たな if 文が始まることがわかりやすくてよいでしょう。入力を終えたら、exp07c.psyexp を保存して実験を実行してみてください。exp07b.psyexp の時と同様に、プローブの長さが一定以上伸びたり縮んだりしないはずです。キーを連打するのが面倒な方はやはり exp07b.psyexp の時と同様に、一回のキー押しで probeLen を増減する量を大きくして試してみましょう。

以上で if 文の練習は終わりですが、最後にひとつ補足しておきます。第一の方法は入れ子になった if 文の練習のためにまず「カーソルキーの左が押されたか」を判定して probeLen の長さを変更してから「probeLen が 0.05 未満か」を判定するという二段階の判定を行いましたが、第 6 章 で学んだ論理演算子を使えば一度に判定することができます。probeLen の初期値は 0.17、0.19、0.21、0.23 の 4 通りしかなくて、±0.005 ずつしか増減しないのですから、0.005 を引いて 0.05 未満になるのは probeLen が 0.05 の時のみです。ということは、「カーソルキーの左が押されていて、なおかつ probeLen が 0.05 より大きい」時には probeLen から 0.005 を引

いても 0.05 を下回ることはありません。したがって、論理演算子 and を使って一つの条件式として記述できます。

```
if 'left' in theseKeys and probeLen > 0.05:
    probeLen -= 0.005
```

同様に、probeLen に 0.005 を加えて 0.35 より大きくなるのは probeLen が 0.35 に達しているときだけですから、「カーソルキーの右が押されていて、なおかつ probeLen が 0.35 未満」の時には probeLen に 0.005 を足しても 0.35 を超えません。したがって、この条件は and を使って一つの式として記述できます。

```
elif 'right' in theseKeys and probeLen < 0.35:
    probeLen += 0.005
```

この方法の弱点は、probeLen の増減量が可変である時にはかえって複雑になってしまうことです。その場合は 図 7.6 や 図 7.7 に示した方法を用いた方がすっきりとしたコードが書けます。この「増減量が可変」な実験を作成することは練習問題としておきましょう。なお、この章で作成した実験は 第 8 章 で活用しますので残しておいてください。

チェックリスト

- if 文の中に入れ子上に if 文を組み込んだコードを記述することができる。
- if や elif の条件式が真であった時に実行されるコードがどこまで続いているかを判断することができる。if、elif の条件式が全て偽で else まで進んだときに実行されるコードがどこまで続いているかを判断することができる。
- 一連の if-elif-else の組み合わせがどこまで続いているかを判断することができる。
- 論理演算子を用いて複数の条件式をひとつの式にまとめることができる。

7.5 練習問題：プローブ刺激の伸縮量を切り替えられるようにしよう

exp07a.psyexp をベースにして、プローブの伸縮量 (probeLen の増減量) を切り替えられるようにしてみましょう。二通りの実現方法を挙げますので、ぜひ両方の方法の実現に挑戦してください。

- 実現方法その 1
  - Shift キーを押すと、伸縮量が ±2 と ±10 で切り替わる。余力がある人は Text コンポーネントを使って現在の伸縮量を画面上に提示すること。
    - ヒント 1：伸縮量の絶対値を保持する変数を一つ用意して、Shift キーが押されたら値を切り替える。
    - ヒント 2：伸縮量の絶対値を保持する変数には、ルーチン開始時に初期値を与える必要がある。
- 実現方法その 2
  - X キーを押すと-10、C キーを押すと-2、ピリオド (.) キーを押すと +2、スラッシュ (/) キーを押すと +10 伸縮する。

        * この方法についてはヒントなし。

「どちらもあっさりできてしまった」という方は、以下の問題にも取り組んでみてください。

- 実現方法その 1、その 2 共通
  - 図 7.7 の例のように、キー入力に関する処理を終えた後に probeLen が範囲を超えていないかを確認して必要があれば値を修正すること。ただし、その際 if 文を使わずに、第 5 章 に出てきた関数を使って「1 行で」処理を記述すること。
    * ヒント：二種類の関数を使う必要がある。

7.6 この章のトピックス

7.6.1 複数キーの同時押しの検出について

キーボードは製品によってテンキーと呼ばれる独立した数字や四則演算のキーがあったり、音量を調節するためのキーがあったり、いろいろなものがありますが、特殊なものを除けば 80 個以上のキーがあります。これらのキーは物理的には複数個同時に押すことはできますが、文書を書くなどの一般的な用途では「P と S と Y のキーを同時押しする」といった具合に複数の文字キーを押すことはありません。ですから、市販されているキーボードの中には、Shift や Ctrl といった特殊なキーを除いて、複数キーの同時押しを検出することを前提に設計されていないものがあります。例えば筆者が使用しているキーボードの中には、F、G、H、J のキーを同時に押すと F と J のみ、G と H のみといった具合にいずれか 2 つのみしか同時に認識しないものがあります。一方、これらの 4 個のキーの同時押しを認識させることができるキーボードもあります。

通常の文書入力では 4 つのキーの同時押しを検出できなくても困ることはまず無いのですが、キーボードを使用して操作するアクションゲームの場合は大きな問題になり得ます。ですから、ゲーム用と銘打って販売されているキーボードは多くのキーを同時押しできるように設計されています。中にはすべてのキーの同時押しを検出できる製品もあります。

Builder は、同時押しに対応したキーボード、対応していないキーボードのどちらが接続されていて同じコードで処理できるように、変数 theseKeys に押されたキー名を必ずリストとして保存するように作られています。複数キーを同時押ししているにも関わらず theseKeys に押したキー名が含まれていない場合は、そのキーの組み合わせが使用中のキーボードで検出できない組み合わせである可能性があります。

7.6.2 メソッドの第一引数について (上級)

TrialHandler の主要クラスメソッドの表 (表 7.2) において、getEarlierTrial( ) の引数は n の 1 個だけしか示されていません。しかし、Python インタプリタ上で psychopy.data を import して help(psychopy.data.TrialHandler) を実行して getEarlierTrial( ) のヘルプを見ると、self と n という 2 つの引数が記載されています。 表 7.2 で第一引数 self を省略している理由について簡単に解説します。

公式ヘルプに記載されている第一引数 self ですが、これは TrialHandler に限らずすべてのクラスのメソッドに必ず存在します。これは「インスタンス自身」を指し示す引数です。C 言語や C++ 言語を御存知の方には、「インスタンスへのポインタが渡される」と言えばわかりやすいかも知れません。クラスの定義に関する Python の文法を解説していないのでどうしても不正確な説明にしかならないのですが、この self はインスタンスが自分自身に格納されたデータ属性の値を知るために必要なもの、と思っておいてください。

Python の文法では、メソッドを呼び出す時に self は省略して記述すると定められています。ですから、引数が self のみしかない foo というメソッドを呼び出す場合は foo( ) という具合に括弧の中は空白にします。TrialHandler の getEarlierTrial( ) を呼び出す場合には、このメソッドには self と n という 2 つの引数があるので、self を省略して getEarlierTrial(n) と書きます。 表 7.2 では、実際にコードを書くときの表記と一致させることを重視して getEarlierTrial( ) の引数として n のみを記載しています。

なお、同じくヘルプの addData( ) メソッドの引数を見ると、self、thisType、value に加えてさらに position=None と書かれています。これはデフォルト値付き引数と呼ばれるもので、「引数 position が渡されなかった場合は None が渡されたと解釈する」ということを意味します。ですから、本文中で trials.addData('response', probeLen) としたように position に相当する引数を渡さなくてもエラーにならなかったのです。getEarlierTrial( ) の引数 n も n=-1 という具合にデフォルト値付き引数として定義されているので、本文中や 表 7.2 で述べたように n を省略することができるのです。

7.6.3 Python コードの字下げについて

第 6 章 において、Python Enhancement Proposals (PEP) という公式文書で半角スペース 4 文字の字下げが推奨されていることを紹介しました。PEP は Python の言語仕様や Python プログラマのコミュニティ向けの情報などを記述した文書の集合で、その中の PEP-8 という Python のコードの書き方を定めた文書に「半角スペース 4 文字を使いなさい」と記されています。

しかし、Python 以外のプログラミング言語では字下げに Tab 文字や 8 文字の半角スペースなど、さまざまな字下げが使用できるためか、Python でも半角スペース 4 文字以外の字下げを使用できるようになっています。図 7.8 は、2 文字や 6 文字の字下げが混在しているコードの例を示しています。 図 7.8 左のように字下げが混在していてもそれぞれのブロック内で字下げが統一されていれば動作します。例えば 図 7.8 の (3) は直前の if に対する字下げが 6 文字、(4) は直前の else に対する字下げが 4 文字であり、一連の if-else 文にも関わらず字下げが一致していません。しかし、(3)、(4) のブロック内でそれぞれ字下げが一貫しているので Python は適切にこのコードを解釈して実行することができます。それに対して 図 7.8 の (5) では、最初の 2 行の字下げが 4 文字であるにもかかわらず最後の i+=1 の字下げは 3 文字であり、(5) のブロック内で一貫していません。従って、 図 7.8 右のコードはエラーとなり実行できません。

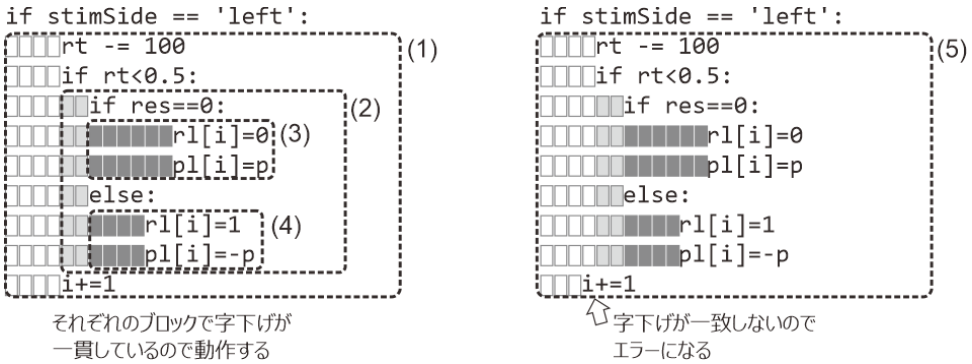


図 7.8　4 文字以外の半角スペースによる字下げ。それぞれのブロック内で字下げが一貫していればエラーにはなりません。

スペースと Tab 文字が混在しているとさらに事態は複雑になります。基本的には、Tab 文字は半角スペース 8 文字に置き換えられます。しかし、半角スペース 8 文字未満に Tab 文字が続く場合は、半角スペースと Tab 文字を合わせて半角スペース 8 文字と解釈されます。 図 7.9 の例をご覧ください。 図 7.9 左のコードの最終行は、4 文字の半角スペースより前に Tab 文字がありますから、Tab 文字が半角スペース 8 文字分に解釈され

て合計半角スペース 12 文字と解釈されます。従って、 図 7.9 左のコードはエラーとならず実行できます。一方、 図 7.9 右の最終行は、左と同じ Tab 文字と半角スペース 4 文字の組み合わせなのですが、半角スペースが Tab 文字より前にあります。この場合、半角スペースと Tab 文字を合わせて半角スペース 8 文字と解釈されますので、直前の if 文と字下げ量が同じとなってしまいエラーになります。

以上のように、Python のスクリプトでは字下げは半角スペース 4 文字でなくても動作します。しかし、混乱を避けるためにはやはり PEP-8 に従って半角スペース 4 文字で統一するべきだと思われます。

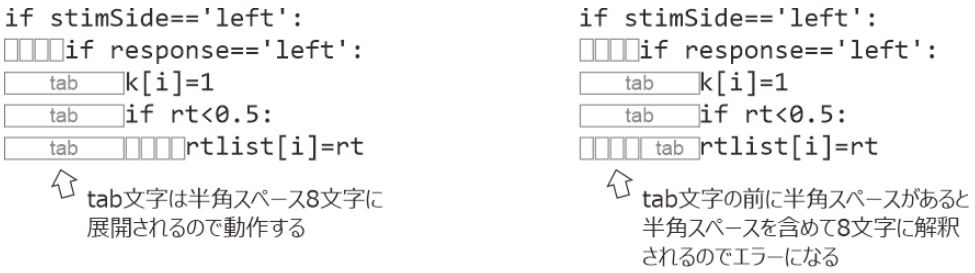


図 7.9　Tab 文字による字下げ。基本的には Tab 文字は半角スペース 8 文字に置換されると考えておけばよいですが、左の例のように半角スペースの後ろに Tab がある場合は半角スペースと Tab をまとめて 8 の倍数個のスペースとして解釈されてしまいます。

7.6.4 Variable コンポーネントによる変数の値の出力

Variable コンポーネントはコンポーネントペインの「カスタム」カテゴリにあります (図 7.10)。Builder はユーザーが Code コンポーネントで独自に追加した変数を把握できないので、実験記録ファイルに値を出力することもしませんし、コンポーネントの **[名前]** や条件ファイルのパラメータ名と重複していないかどうかも確認しません。いずれも初心者にはわかりにくく、特に後者 (重複を確認しない) は熟練者でもなかなか気づかないような「意図しない動作」の原因になりかねません。Variable コンポーネントを使うと、Builder がその変数の存在を把握できるので、 **[名前]** の重複も確認されますし、Code コンポーネントを使わずに値を設定、変更したり実験記録ファイルに出力させたりすることができます。

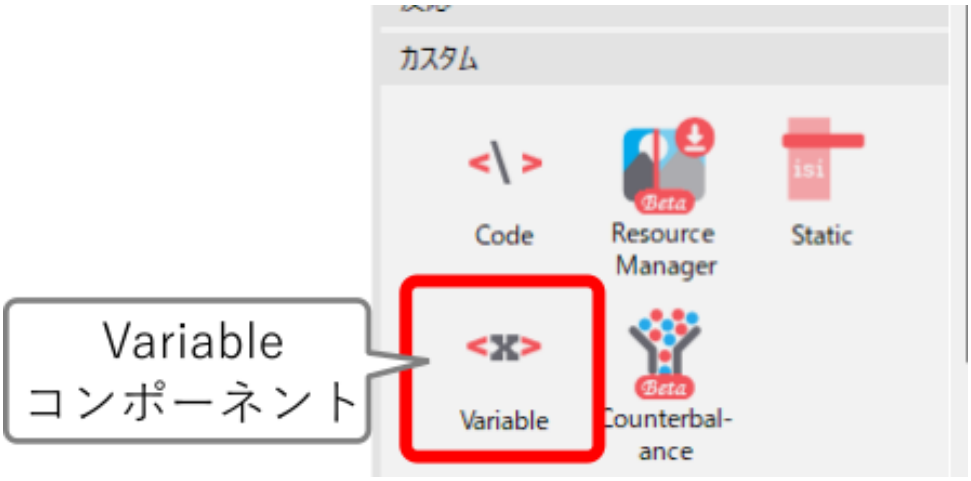


図 7.10　Variable コンポーネントは「カスタム」カテゴリにある。

表 7.3 に Variable コンポーネントの主なプロパティを示します。**[フレーム更新開始時の値 $]** と **[フレーム更新時の値を保存]** は直感的ではないので注意が必要ですが、それ以外は難しい点はないと思います。ルーチン上でのコンポーネントの並び順に従って処理されるので、Variable コンポーネントで値を設定した変数を利用するコンポーネントがある場合は、Variable コンポーネントより下になるように配置することに気をつけてく

ださい。本章の実験で使用するなら、「7.3:Code コンポーネント使って独自の変数の値を記録ファイルに出力しよう」の作業をする前の時点で trial コンポーネントに Variable コンポーネントを設置して、**[名前]** を probeLen、**[Routine 開始時の値 $]** に initProbeLen と設定すると良いでしょう。そして、調整後の値を実験記録ファイルに出力するには **[Routine 終了時の値を保存]** をチェックしてください。

表 7.3　Variable コンポーネントの主要なプロパティ。

[名前]	他のコンポーネントと同様、このコンポーネントの名前ですが、同時に変数の名前となります。つまり foo という **[名前]** で Variable コンポーネントを配置すると、実験内で foo という変数が使用できるようになります。
[実験開始時の値 $]	実験開始時の値を指定します。Code コンポーネントで定義するなら「実験開始時」のタブで初期化したい変数ならここで定義します。
[Routine 開始時の値 $]	Routine 開始時の値を指定します。Code コンポーネントで定義するなら「Routine 開始時」のタブで初期化したい変数ならここで定義します。
[フレーム更新開始時の値 $]	フレーム更新開始時の値を指定します。ここでいう「フレーム更新開始時」とは正確にはルーチンペイン上で当該 Variable コンポーネントの処理順がまわってきたタイミングを指します。もしルーチンペイン上で Variable コンポーネントより上に他のコンポーネントが配置されている場合は、そちらの処理が先に行われます。
[実験開始時の値を保存]	**[実験開始時の値 $]** に設定した値を実験記録ファイルに出力したい場合にチェックします。
[Routine 開始時の値を保存]	**[Routine 開始時の値 $]** に設定した値を実験記録ファイルに出力したい場合にチェックします。
[フレーム更新時の値を保存]	「最初」、「最後」、「すべて」、「なし」から選択します。保存しない場合は「なし」にしてください。Variable コンポーネントを使って値を設定していないと保存されないので注意してください。
[Routine 終了時の値を保存]	ルーチンが終了した時点のその変数の値を実験記録ファイルに出力します。Variable コンポーネントを使って値を設定していないと保存されないので注意してください。
[実験終了時の値を保存]	フローの最後まで実験が進行して終了処理をおこなう時点でのその変数の値を実験記録ファイルに出力します。Variable コンポーネントを使って値を設定していないと保存されないので注意してください。

本文の Code コンポーネントを使う方法と、Variable コンポーネントを使う方法のどちらを使うかは好みで決めてよいですが、それぞれに長所短所があります。Variable コンポーネントの長所は先ほど書いた通り、名前が重複していないか Builder がチェックしてくれることと、これだけのために Code コンポーネントを使う必要がないことです。ルーチンペインに Variable コンポーネントが配置されていれば、その名前の変数が使用されいていることが一目瞭然というのも利点です。自分が作った実験でも数か月経ったら詳細を忘れてしまったりするものですし、研究室で先輩から後輩へと引き継いでいくような場合には「わかりやすい」ということは特にありがたいでしょう。

Variable コンポーネントの短所は、複数のルーチンにまたがって使用しなければいけない変数の扱いが難しくなる場合があることです。例えば practice というルーチンに response という名前の Variable コンポーネントを置いたら、trial というルーチンに同じ response という名前の Variable コンポーネントを置くことはできません。Builder の実験においてコンポーネントの名前はグローバルなものなので、どちらか一方のルーチンに配置しておけばもう一方でもその変数にアクセスできますが、**[Routine 開始時の値$]** などのプロパティを使って値を設定したり、**[Routine 終了時の値を保存]** を使って値を保存したりできるのはコンポーネントが置かれているルーチンのみです。そういった処理が必要な場合は結局 Code コンポーネントを使用しなければいけません。Variable コンポーネントで「こういう名前の変数を使用している」とアピールしつつ、複数ルーチンにまたがる処理は Code コンポーネントを使うといった組み合わせも可能なので、うまく活用してください。

第 8 章

グラフィカルインターフェースを活用しよう

8.1 この章の実験の概要

第 7 章 ではキーボードを用いてプローブ刺激の長さを調節しました。しかし、実際に実行してみた方は「カーソルキーを何度もカチカチ押すのは面倒くさい」と思われたのではないでしょうか。Builder の Keyboard コンポーネントでは、基本的に「キーを押した」か「(既に押してある) キーを放した」のいずれかのイベントしか検出しません。ですから、「押しっぱなしにしていたらその間伸び縮みする」という動作を実現することが難しいのです。まあ Code コンポーネントを使うと可能ですので、Code コンポーネントの使い方を極めるという中級から上級者向けの練習問題としては面白そうですが、そんな難しいことをしなくてもぱぱっと実現できてしまった方がいいに決まっています。残念ながら現在の Builder(執筆時点で 2024.2.5) ではキーボードを使う限りどうしても難しくなってしまうのですが、マウスを使うとキーボードよりはずっと簡単に実現可能です。本章では、まず 第 7 章 の実験をマウスを使って操作するように改造します。その後、マウスの使い方をもう少し覚えようということで 第 6 章 章の実験にも少し改造を加えます。それぞれの章の実験を完成させたところから解説を始めますのでご注意ください。

8.2 Button コンポーネント

マウスを使うと言ってもいろいろな使い方がありますが、基本はマウスを操作してポインターを移動させてクリックして選択といった操作でしょう。Builder 上でこれらの操作を実現するためのコンポーネントが Button コンポーネントです。Button コンポーネントはコンポーネントペインの「反応」カテゴリにあります (図 8.1)。

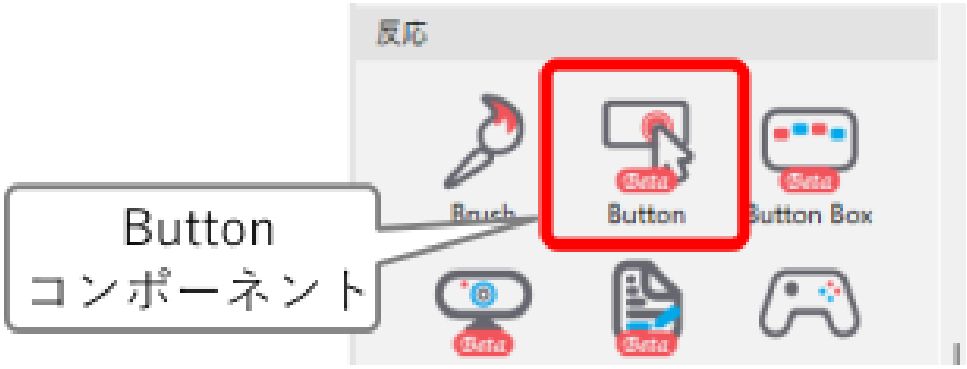


図 8.1　Button コンポーネント

表 8.1 に Button コンポーネントの主なプロパティを示します。このコンポーネントで作成されるボタンの外見 (色や大きさ、テキストの文字サイズなど) を調整するプロパティはは Polygon コンポーネントと Text コンポーネントと共通するものが多いため、それらは省略してあります。まず、Button コンポーネントは Keyboard コンポーネント同様、ルーチンを終わらせるために使いたい場合がありますので、それに対応するように **[Routine を終了]** というプロパティを持っています。使い方は Keyboard ルーチンと同じです。実験記録ファイルには button.numClicks (button は Button コンポーネントの **[名前]**) という項目にクリックされた回数が出力されるので、スクリーン上に複数のボタンがあってもどのボタンがクリックされたか判別できます。

表 8.1　Button コンポーネントの主なプロパティ

データ属性, 概要	
[Routine を終了]	Keyboard コンポーネントと同様、この項目がチェックされているとボタンがクリックされたときに現在のルーチンを終了します。
[ボタンのテキスト]	ボタンに表示するテキストを指定します。
[コールバック関数]	ボタンがクリックされたときに実行するコードを記入します (Python の文法に詳しい方向け：通常の関数ではなく、「ボタン上でクリックされた」を条件式とする if 文において条件式が真だったときに実行されるブロックを書きます)。
[クリック毎に 1 回実行]	**[コールバック関数]** に書かれたコードをクリックしたときに 1 回だけ実行するか、クリックした後ボタンが放されるまでの間繰り返し実行するかを選択します。 **[Routine を終了]** がチェックされている場合は (直ちにルーチンが終了してしまうので) 意味をなさないため無効になります。
thisTrialN, Trial-by-trial 記録ファイルの thisTrialN と同じです。	
[フォント]	ボタンに表示されるテキストのフォントを指定します。Text コンポーネントと異なる **初期値のままでは日本語などの非 ASCII 文字化けします** ので、非 ASCII 文字を表示したい場合は適切なフォントを指定する必要があります。
[文字の高さ]	ボタンに表示されるテキストのフォントを指定します。 **[サイズ [w, h] $]** で指定した **ボタンサイズに対して大きすぎる場合、テキストがはみ出したり表示されなかったりする** ので注意してください。

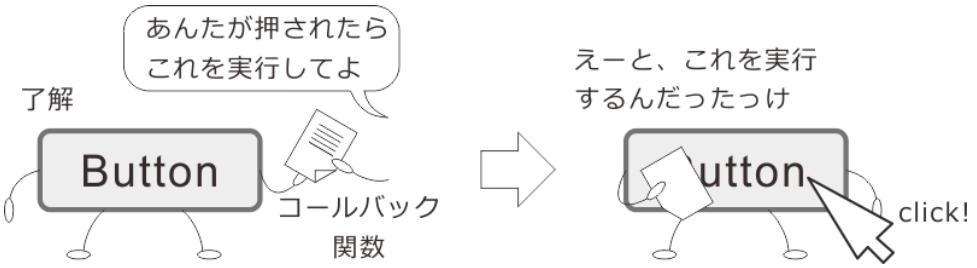


図 8.2　コールバック関数の仕組み

Button コンポーネントを活用するうえで非常に重要なのが **[コールバック関数]** です。コールバック関数とは

「関数から呼び出すために引数として渡しておく関数」のことで、ここでは Button コンポーネントに「もしクリックされたらその時はこれを実行してほしい」と渡しておくコードを意味しています (図 8.2)。プログラミングに詳しい人は却って混乱してしまうかも知れませんが「関数」という名前にもかかわらず、関数の定義 (Python なら def 文) を書く必要はありません。**[コールバック関数]** に入力したコードは、Code コンポーネントを使って **[フレーム毎]** のタブに

```
if (ボタンがクリックされた):
    [コールバック関数] の欄に
    記入したコード
```

というコードを記入したのと同じように解釈されます。「(ボタンがクリックされた)」の部分は Button コンポーネントのプロパティ設定によって実際の式が変化しますし、クリック時刻を記録するためのコードも出力されるのですが、そういった細かいことは抜きにして「そのような判定をするコードが書かれている」とだけ理解しておいてください。

具体例を挙げると、**[コールバック関数]** に

```
if probeLen < 0.35
    probeLen += 0.005
```

と書いておけば

```
if (ボタンがクリックされた):
    if probeLen < 0.35
        probeLen += 0.005
```

というコードを Code コンポーネントで **[フレーム毎]** のタブに挿入したのと同じように動作します。第 7 章の実験をどのように変更すればマウスで操作できるようになるか、イメージできたでしょうか。

続いて **[クリック毎に 1 回実行]** ですが、これが Keyboard コンポーネントと大きく異なるポイントです。この項目がチェックされていると、Keyboard コンポーネントによるキー押し検出のように、Button コンポーネント上でマウスのボタンが押されたその最初の瞬間に 1 度だけ **[コールバック関数]** の処理が行われます。チェックされていなければ、Button コンポーネント上でマウスのボタンが押されている間、フレーム毎に **[コールバック関数]** の処理が行われ続けます。第 7 章 の実験で「キーを押し続けている間プローブが伸縮してほしい」という要望が出てきましたが、この項目をチェックしていない時の動作がまさにこの要望に応えるものです。

逆に **[クリック毎に 1 回実行]** をチェックしておきたいのはどういう状況かというと、Button コンポーネントをクリックする度に視覚刺激の ON/OFF が切り替わるといった動作が求められるときです。普通の人がマウスのボタンをクリックしたとき、実際にボタンが押されている時間は 1 フレームの時間 (モニターが 60Hz なら約 16.7 ミリ秒) よりはるかに長くなります。ですから、**[クリック毎に 1 回実行]** をチェックしていない場合、ボタンを押している最中は 1 フレーム毎に ON/OFF が切り替わり、ボタンを放した瞬間切り替わりが止まることになります。狙った状態で止めるのは困難でしょう。Keyboard コンポーネントの動作はこちらと同じと言えます。

なお、表 8.1 に書かれているとおり、**[Routine を終了]** がチェックされている場合は Button コンポーネン

ト上でボタン押しが検出されたフレームでルーチンが終了してしまうので **[クリック毎に 1 回実行]** は意味を持ちません。そのため **[Routine を終了]** がチェックされている時には **[クリック毎に 1 回実行]** は無効になります。

その他のプロパティはおおむね Polygon コンポーネントと Text コンポーネントと同じですが、表 8.1 で挙げている **[フォント]** と **[文字の高さ]** は注意が必要です。Text コンポーネントと異なり、Button コンポーネントは **[フォント]** に欧文フォントが指定されているときに自動的に日本語のフォントを割り当ててくれません。したがって、日本語の文字を表示したい場合は **[フォント]** に Yu Gothic (Windows の場合) や Hiragino Kaku Gothic Pro W3 (MacOS の場合) などの日本語対応フォントを指定する必要があります。また、**[文字の高さ]** がボタンのサイズに対して大きすぎる場合、テキストがまったく表示されない場合があります。原因が直感的にわかりにくいので注意してください。

それでは、第 7 章 の実験をマウスで操作できるようにしてみましょう。

チェックリスト

- Button コンポーネントで作成したボタンをクリックすることでルーチンを終了することができる。
- Button コンポーネントで作成したボタン上でマウスのボタンを押し続けることでフレーム毎に処理を実行させることができる。
- Button コンポーネントで作成したボタン上でマウスのボタンを 1 回押すごとに 1 回処理を実行させることができる。
- Button コンポーネントで作成したボタンに日本語の文字を表示させることができる。

8.3 ミューラー・リヤー錯視の実験をマウスで操作できるようにしよう

第 7 章 の作業を最後までおこなった状態 (練習問題は除く) から作業する前提で手順を解説します。

準備作業

– 第 7 章 の実験をフォルダごとコピーして (つまり中の psyexp ファイルや xlsx ファイルもすべて含めて)、わかりやすい名前で貼り付ける。例えば 第 7 章 の実験を exp07 というフォルダにまとめている場合、まるごとコピーして exp07_mouse などの名前で貼り付ければよい。

実験設定ダイアログ

– 「スクリーン」タブの **[マウスカーソルを表示]** をチェックする。

trial ルーチン

– 配置済みの Keyboard コンポーネントを削除する。

– **Button コンポーネントを 1 つ配置し、以下のように設定する。**

* 「基本」タブの **[名前]** を `buttonOK` にして、**[開始]** を「時刻 (秒)」の `0.5` にする。**[Routine を終了]** がチェックされていることを確認し、**[ボタンのテキスト]** を `OK` に変更する。
* 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.1, 0.04)` に、**[位置 [x, y] $]** を `(0, -0.3)` にする。

    - 「書式」タブの **[フォント]** を `Yu Gothic` (Windows の場合)、`Hiragino Sans` (MacOS の場合) など、各自の PC で使用できる日本語フォントにする。 **[文字の高さ]** を `0.03` にする。
    - **ルーチンの一番上になるように並び替える。buttonOK が一番上にあり、そのすぐ下が Code コンポーネントになっていることを確認する。**

- **–buttonOK をコピーし、buttonOK のアイコンを右クリックしてメニューの「下に貼り付け (buttonOK)」を選択して `buttonLonger` の名前で貼り付ける。 ルーチンペイン上で一番上に buttonOK があり、そのすぐ下に buttonLonger、さらにその下に code になっていることを確認 してから以下のように設定する。**
    - 「基本」タブの **[Routine を終了]** のチェックを外す。 **[ボタンのテキスト]** に **長く** と入力し、 **[クリック毎に 1 回実行]** がチェックされていることを確認する。
    - 基本」タブの **[コールバック関数]** に `probeLen = min(probeLen+0.005, 0.35)` と入力する。
    - 「レイアウト」タブの **[位置 [x, y] $]** を `(0.15, -0.3)` にする。
- **–buttonLonger をコピーし、 buttonLonger のアイコンを右クリックしてメニューの「下に貼り付け (buttonLonger)」を選択して `buttonLonger_2` の名前で貼り付けてから以下のように設定する。**
    - 「基本」タブの **[ボタンのテキスト]** に **長く（長押し）**と入力し、 **[クリック毎に 1 回実行]** のチェックを外す。
    - 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.2, 0.04)` に、 **[位置 [x, y] $]** を `(0.35, -0.3)` にする。
- **–buttonLonger をコピーし、 buttonLonger_2 のアイコンを右クリックしてメニューの「下に貼り付け (buttonLonger)」を選択して `buttonShorter` の名前で貼り付けてから以下のように設定する。**
    - 「基本」タブの **[ボタンのテキスト]** に **短く** と入力する。
    - 基本」タブの **[コールバック関数]** に `probeLen = max(probeLen-0.005, 0.05)` と入力する。
    - 「レイアウト」タブの **[位置 [x, y] $]** を `(-0.15, -0.3)` にする。
- **–buttonLonger_2 をコピーし、 buttonShorter のアイコンを右クリックしてメニューの「下に貼り付け (buttonLonger_2)」を選択して `buttonShorter_2` の名前で貼り付けてから以下のように設定する。**
    - 「基本」タブの **[ボタンのテキスト]** に **短く（長押し）**と入力する。
    - 基本」タブの **[コールバック関数]** に `probeLen = max(probeLen-0.005, 0.05)` と入力する。
    - 「レイアウト」タブの **[位置 [x, y] $]** を `(-0.35, -0.3)` にする。
- – 配置済みの Code コンポーネントの **[フレーム毎]** に入力してあるコードをすべて削除する。

ここまで作業したら、trial ルーチンには上から Button コンポーネントが 5 つ並び、続いて Code コンポーネントがひとつ、さらに続いて Polygon コンポーネントが 6 つ並んでいるはずです。作業手順上、各 Button コンポーネントをどの位置に貼り付けるか指定しましたが、5 つの Button コンポーネント間の順序は異なっていても動作には影響ありません。

実行すると、 図 8.3 のように刺激とプローブの下に 5 つのボタンが表示されます。「長く (長押し)」と「短く (長押し)」にマウスカーソルを合わせてマウスの左ボタンを押しっぱなしにすると、長さ 0.05 から 0.35 の範囲内でプローブの長さが変化し続けることを確認してください。また、「長く」と「短く」のボタンは Keyboard コンポーネントの時のように、マウスのボタンを押したときに 1 回だけ長さが変化することも確認してください。 **[クリック毎に 1 回実行]** をチェックする、しないの違いがおわかりいただけると思います。「OK」と書かれたボタンをクリックすると次の試行へ進みます。

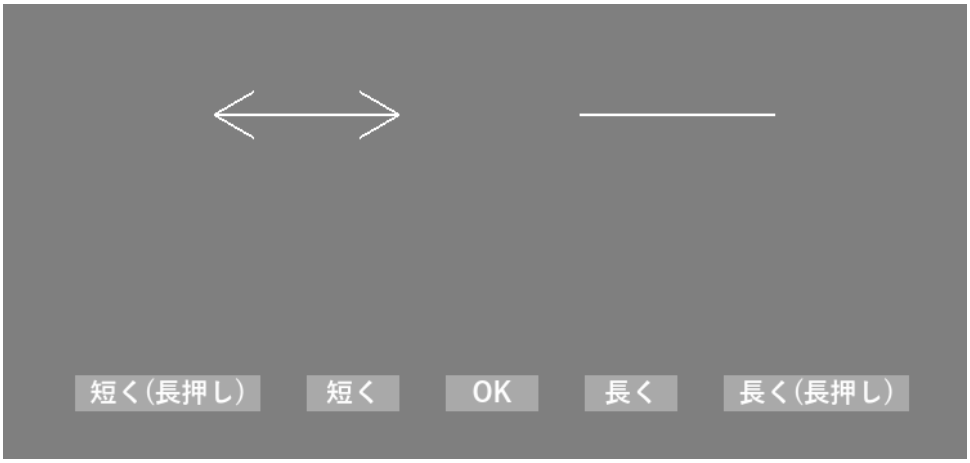


図 8.3　マウスでプローブの長さを調節する。

[コールバック関数] に入力した probeLen = min(probeLen+0.005, 0.35) と言う式は少し補足が必要でしょうか。min( ) は 表 5.2 で紹介した関数で、引数の中から最小の値を返します。ここでは probeLen+0.005 と 0.35 を引数として与えていますので、probeLen+0.005 が 0.35 より小さければその値が、0.35 より大きければ 0.35 を返します。これによって、「長さを 0.005 増加させるが 0.35 を越えないようにする」という処理が 1 行で実現できています。同様に、probeLen = max(probeLen-0.005, 0.05) は「長さを 0.005 減少させるが 0.05 未満にならないようにする」処理を実現しています。これは「 5.7: 練習問題：パラメータが適切な範囲を超えないようにしよう 」のヒントにもなっていますので、わからなかった方はこのテクニックを念頭において改めて「 5.7: 練習問題：パラメータが適切な範囲を超えないようにしよう 」を考えてみてください。

以上のコードにより、Code コンポーネントの **[フレーム毎]** で行っていた処理はすべて実現できてしまっているので、 **[フレーム毎]** のコードはすべて削除しました。しかし、 **[Routine 開始時]** と **[Routine 終了時]** の処理がまだ必要なので Code コンポーネント自体は残さないといけません。Variable コンポーネントを使用すれば完全に Code コンポーネントを削除してしまうことができますが、それは練習問題としておきましょう。Code コンポーネントでは probeLen に加えて theseKeys も初期化しなければいけないこと、theseKeys の値は実験記録ファイルに出力する必要がないことの 2 点がポイントとなるでしょう。

8.4 Slider コンポーネントと Form コンポーネント

本章を Button コンポーネントの話だけで終えてしまうのも物足りないので、マウスと相性がよい Slider コンポーネントと Form コンポーネントを紹介しておきましょう。いずれもコンポーネントペインの「反応」カテゴリにあります。

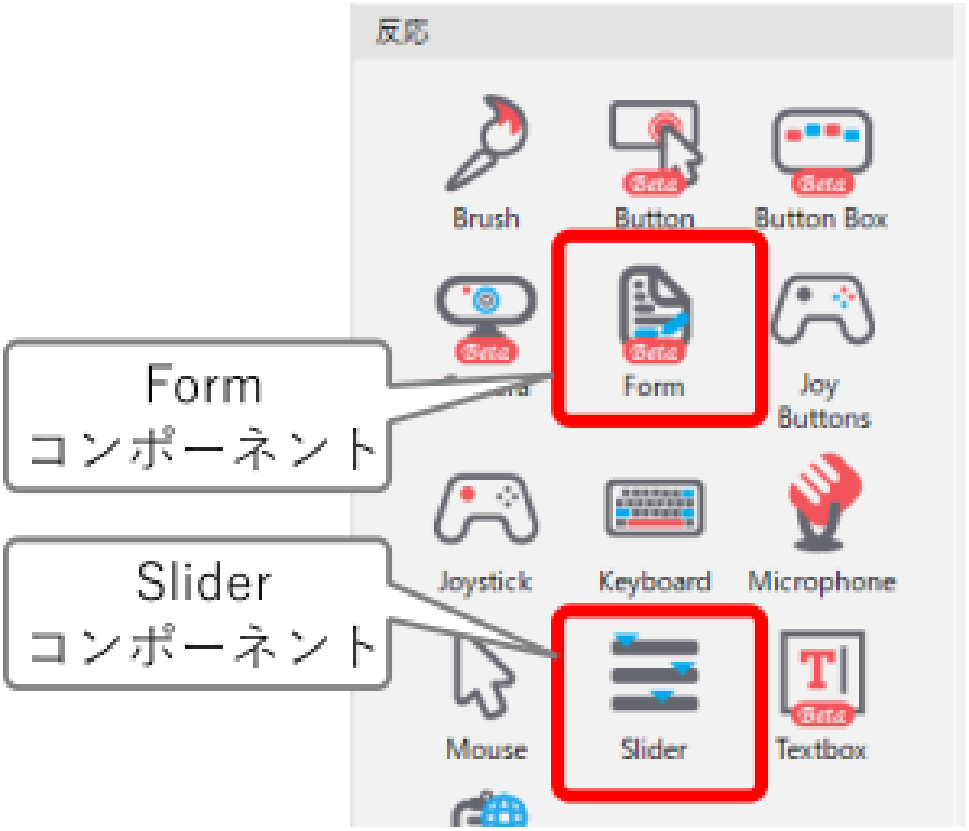


図 8.4　Slider コンポーネントと Form コンポーネント

Slider コンポーネントは、定規のような「尺度」をスクリーン上に提示して、尺度上の位置を参加者に選択させることによって反応を記録するためのコンポーネントです。心理学実験ではこの種の反応を求める課題は少なくないので、覚えておくときっと役に立つはずです。表 8.2 に主なプロパティを示します。**[Routine を終了]** やフォントの設定に関することなどは Button コンポーネントと同様なので省略してあります。

表 8.2　Slider コンポーネントの主なプロパティ

データ属性, 概要	
[スタイル]	尺度の見た目を決定します。slider, rating, radio, scrollbar, choice のいずれかを選択します。slider, rating, scrollbar は離散値と連続値の両方の記録に使用できますが、radio, choice は整数個の選択肢のなかからの選択にのみ使用できます。
[目盛 $]	尺度の目盛位置をカンマ区切りの数値で指定します。 **[スタイル]** が radio の場合は設定できません。
[ラベル $]	目盛につけるラベルをカンマ区切り指定します。 **[スタイル]** が radio の場合は必ず指定しないといけませんが、その他の場合は省略できます (ラベルは表示されません)。 **[目盛 $]** の要素数と **[ラベル $]** の要素数が異なる場合、 **[ラベル $]** が 2 個なら尺度の両端のラベルとなります。 **[ラベル $]** が 3 個以上で **[目盛 $]** の個数と一致しない場合、エラーになりませんがそのような設定は避けるべきでしょう。
[精度 $]	**[スタイル]** が slider, rating, scrollbar のとき、マーカー位置の最小間隔を指定します。例えば **[目盛 $]** が 1, 2, 3, 4, 5 で **[精度 $]** が 1 の場合、マーカー位置は 1 刻みでしか置くことができないので五段階の尺度となります。0 を指定すると実験の実験環境における最小間隔となり、実質的に連続値の尺度となります。radio ではこの項目は無効になるほか、choice では値の設定はできますが無視されます (2024.2.5 で確認)。
[初期値 $]	Slider コンポーネントが開始された時のマーカーの位置を指定します。空欄の場合、マーカーがどこにも置かれていない状態で開始されます。
[サイズ [w, h] $]	他のコンポーネントと同様ですが、高さよりも幅が大きければ横向き、幅よりも高さが大きければ縦向きの尺度が描かれます。
[回転角度 $]	他のコンポーネントと同様ですが、表示が乱れることが多いため 0 度以外での使用はお勧めしません。縦にする場合は **[サイズ [w, h] $]** で指定してください。
[反転]	通常、ラベルは尺度の下または左に表示されますが、この項目がチェックされていると下または右に表示されます。
[履歴を記録]	最終的なマーカー位置だけでなく、それまでのマーカー位置の変更履歴 (位置と時刻) も記録します。 **[Routine を終了]** がチェックされている場合は意味を持ちません。

[スタイル] は尺度の見た目を決めるもので、表 8.2 の中から選びます。さらに **[目盛 $]** と **[ラベル $]** で細かな設定をしていきますが、これらのパラメータと **[スタイル]** の関係が複雑なうえ、PsychoPy のバージョンアップの過程でいくつか仕様変更もおこなわれており、すべてを解説するのは大変です。多くの方にとってはそんな踏み込んだ話よりもお勧めの使用パターンを紹介した方が有益だと思いますので、以下にいくつか示します。

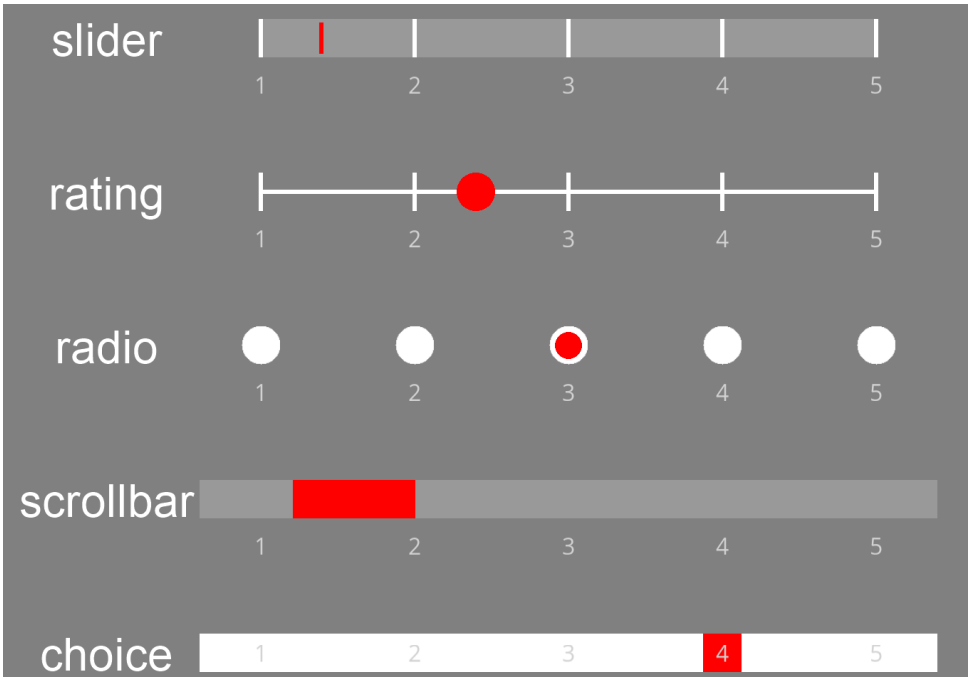


図 8.5　Slider コンポーネントのスタイル

- カテゴリカルな尺度では radio を使うのが良いと思います。radio では **[ラベル $]** を省略すると実行時にエラーになります (2024.2.5 で確認)。
- 離散的な順序尺度では rating を使うのが良いと思います。rating では **[目盛 $]** を 1, 2, 3, 4, 5 のように間隔 1 の整数にして、 **[精度 $]** を 1 にすれば目盛の位置しかマーカーを置けない尺度になります。 **[ラベル $]** は両端の 2 個を指定するか、 **[目盛 $]** の要素数と一致する個数を指定するかをお勧めします。ラベルの位置が気に入らない場合は **[ラベル $]** を空欄にして、Text コンポーネントを使って自分でラベルを配置するとよいでしょう。
- 連続的な値の尺度では rating, slider, scrollbar のいずれかが良いと思います。 **[精度 $]** を 0 にすれば実験の実行環境で最も細かな位置調整ができますが、 **[目盛 $]** を 0, 100 として **[精度 $]** を 1 としたり、**[目盛 $]** を 0, 1 として **[精度 $]** を 0.02 とするなど、実験の目的に応じた精度を積極的に指定するのもよいでしょう。

「レイアウト」タブでは他のコンポーネントと同様に尺度のサイズや位置、回転角度などを指定できますが、**ラベルの表示などがおかしくなることが多いので Slider、Slider コンポーネントを回転させるのはお勧めしません** 。 初期値では Slider コンポーネントは横向きの尺度を描画しますが、表 8.2 のように、 **[サイズ [w, h] $]** で高さの方が幅よりも長くなるように値を設定すると縦向きの尺度を描画します。ほとんどの用途に置いて、横向きまたは縦向きの尺度を描画できれば十分でしょう。

実験の実行中に Slider コンポーネントでマーカーが置かれている位置の値を得る必要がある時には、rating というデータ属性にアクセスします。例えば resp_slider という名前の Slider コンポーネントがあるとき、`resp_slider.rating` と書けば現在のマーカー位置の値が得られます。rating や slider、scrollbar のように連続量を扱えるスタイルでは、 **[精度 $]** で整数値しかとらないように制限していても浮動小数点数値 (float 型) となります。radio では、選択されている項目が **[ラベル $]** で数値として与えられているなら数値、文字列として与えられているなら文字列となります。あまり現実的な実験ではない状況でしょうが、 **[ラベル $]** の要素がそれぞれ違うデータ型の場合、そのラベルのデータ型がそのまま反映されます。具体的に言うと、 **[ラベル**

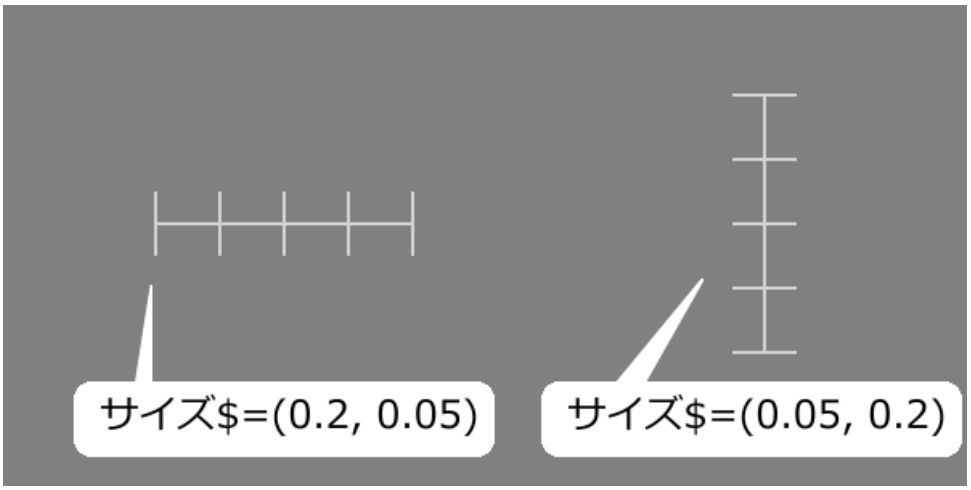


図 8.6　**[サイズ [w, h] $]** で高さの方が幅より大きな値にすると縦向きの尺度になる

$] に 1, 2.0, '3' と定義してあるなら、1 を選択すれば int 型、2.0 を選択すれば float 型、'3' を選択すれば str 型の値が得られます。注意が必要なのは、 **参加者が Slider 上をクリックする前は rating の値は None である** という点です。 **[初期値 $]** でマーカーの位置を指定していて既にマーカーが表示されていても、参加者がまだクリックしていなければ None ですので間違えないようにしてください。

現在どの位置にマーカーが表示されているかを知るには markerPos という属性を使います。例えば resp_slider という名前の Slider コンポーネントがあって、3.2 の位置にマーカーがあるなら `resp_slider.markerPos` の値は 3.2 となります。markerPos に値を代入すれば、コードでマーカー位置を変更することも可能です。以下のコードを Code コンポーネントを使って実行すると、マーカーの位置を 0.0 へ移動させることができます。

```
resp_slider.markerPos = 0.0
```

注意が必要なのは Slider コンポーネントの **[スタイル]** が radio のときで、markerPos では最初のラベルを 0.0、2 番目のラベルを 1.0、…という具合に浮動小数点数で「マーカーの位置」を返してきます。**[目盛 $]** が 0 から N-1(N はラベルの個数) の整数を並べたもの、 **[精度 $]** が 1 に設定されていると考えれば他のスタイルと統一的に扱えます。**[スタイル]** が radio の時に **[初期値 $]** を設定する場合も同様に考えて、最初のラベルなら 0、2 番目のラベルなら 1 という具合に指定してください。なぜ最初のラベルを 1 ではなく 0 として数えるのかは、おそらく Python におけるシーケンスの位置の数え方と密接な関係があります。 第 9 章 で説明する予定です。

この「参加者がまだクリックしていなければデータ属性 rating の値が None である」ということをうまく利用すると、スクリーン上に **[Routine を終了]** のチェックを外した Slider コンポーネントを複数配置して「スクリーン上のすべての Slider コンポーネントに反応すればルーチンを終了する」といった動作を Code コンポーネントを使って実現することが可能です (「8.6.1: 同一ルーチンに配置されている複数の Slider コンポーネントが回答済みか判定する 」参照)。しかし、コードを書くのが少々面倒ですし、なにより複数の Slider コンポーネントをラベルのことまで考慮しながらサイズと位置を調整するのはなかなか骨が折れる作業です。こういう時に非常に便利なコンポーネントが Form コンポーネントです。表 8.3 に主なプロパティを示します。

表 8.3　Form コンポーネントの主なプロパティ

データ属性, 概要	
[項目]	Form コンポーネントで管理する尺度などのパラメータを定義した xlsx ファイルや CSV ファイルを指定します。
[無作為化]	項目を xlsx ファイル等で定義したとおりの順序で配置するか、実行の度に無作為に並び替えるかを選択します。
[データフォーマット]	実験記録ファイルに結果を出力する際に、各 Slider の値を行方向に出力するか列方向に出力するかを選択します。
[項目間の余白]	項目間の余白を指定します。CSS などに詳しい方は table の padding に対応すると考えてください。
[スタイル]	配色を dark, light, カスタム... の中から選択します。カスタム... にした場合、「外観」タブの色に関する項目が設定可能になります。

Form コンポーネントを使いこなす鍵は **[項目]** に指定する設定ファイルです。具体例を見てみましょう。図 8.7 上は設定ファイルの例です。条件ファイルと同様、1 行目はパラメータ名で、2 行目以降は 1 行につきひとつの見出しや尺度などの項目に対応しています。この設定ファイルを実際に Form コンポーネントの **[項目]** に指定した実行した結果が 図 8.7 下です。Slider コンポーネントで作成できるような尺度が整然と並んで表示されています。もちろん、マウスを使ってそれぞれの項目を操作することが可能です。右端にある縦長の長方形はスクロールバーです。項目数が多くて Form コンポーネントの **[サイズ [w, h] $]** に入りきらなかった場合、このように自動的にスクロールバーが出現して上下にスクロールさせることができます。Slider コンポーネントを複数並べて自分で位置調整するのとは比べ物にならない簡単さです。

設定ファイルでは、図 8.7 で示したものだけでなくさまざまなパラメータを指定することができます (表 8.4)。type で Slider コンポーネントの rating, slider, radio, choice(ただし Form コンポーネントでは radio と同じ) を選べるほか、heading で見出し行、description で教示文を配置することもできます。Text コンポーネントの **[文字列]** を条件ファイルで変更するときと同様、複数行のテキストを入力することも可能です。一方、Text コンポーネントとは異なり、長い文字列を入力すると日本語であっても表示範囲に収まるように自動的に改行されます。ただし、改行位置の調整があまりスマートではないことや、英単語の途中で改行するときのようにハイフンが入ってしまったりしますので、手作業で改行した方がよい出力が得られるでしょう。xlsx 形式で設定ファイルを作成する場合、セル内で改行すればその通りに反映されますが、CSV 形式で設定ファイルをする場合は、 図 8.7 の「rating の例」の tickLabels のように改行する位置に\n と書いて改行を表すことができます (「6.9.2: 改行文字を使った複数行の文字列の表現 (上級) 」参照)。

表 8.4: Form コンポーネントの項目ファイルで指定する内容

index	項目の順序を整数で指定します。省略すると自動的に番号が割り振られます。刺激の描画には影響しませんが、 **[無作為化]** した時にデータファイルの出力を項目順に並び替えるときに役に立ちます。
itemText	表示したい文を指定します。

次のページに続く

表 8.4 – 前のページからの続き

type	項目の種類を指定します。rating,slider,radio は Slider コンポーネントと同じです。choice も指定できますが radio と同じになります (2024.2.5 で確認)。heading,description は教示などの文だけを表示したいときに使います (heading はやや大き目でボールド体になる)。heading, description を選んだ場合は options, ticks, ticklabels, layout, responseColor, responseWidth, granularity は意味を持ちません。ほかに文字列を入力できる free text を指定できますが、現状では日本語入力には使い物になりません (「8.6.2: 日本語環境における Form コンポーネント (および TextBox コンポーネント) の文字入力機能」参照)。
ticks	スライダーの目盛をカンマ区切りで指定します。
tickLabels	スライダーの目盛をカンマ区切りで指定します。
options	スライダーのラベルをカンマ区切りで指定します。ラベルに対応する値は自動的に決まります。 **旧バージョンでは ticks, tickLabels がなかったため options で指定していましたが、現在は ticks, tickLabels で指定する方がよいでしょう。**
layout	スライダーの方向を horiz、vert のいずれかで指定します。horiz なら水平方向、vert なら垂直方向です。省略すると horiz になります。
itemColor	質問文の文字色を指定します。省略すると white になります。Form コンポーネントの **[スタイル]** が custom... の場合のみ有効です。
itemWidth	Form コンポーネントの **[サイズ$]** で指定した幅のうち、質問文が占める幅の割合を 0.0 から 1.0 指定します。
responseColor	スライダーの色を指定します。省略すると white になります。Form コンポーネントの **[スタイル]** が custom... の場合のみ有効です。
responseWidth	Form コンポーネントの **[サイズ$]** で指定した幅のうち、スライダーが占める幅の割合を 0.0 から 1.0 指定します。itemWidth と responseWidth の合計が 1.0 を越えるとレイアウトが崩れますので注意してください。
granularity	Slider コンポーネントの **[精度 $]** に対応します。type が slider の時のみ有効です。rating, radio, choice の時は目盛の位置しか選択できません。
font	使用するフォントは Form コンポーネントのプロパティダイアログの **[フォント]** で指定しますが、特定の項目だけ別のフォントを指定したい場合はここでフォント名を指定します。

Slider の左側のテキストと Slider の幅のバランスは itemWidth と responseWidth で調整します。それぞれの項目で異なる値を指定できるので、 図 8.7 の例のように複数の種類の Slider を使い分ける際に種類ごとに異なるバランスにしたり、左側のテキストがどうしても長くする必要がある項目だけすこし itemWidth の幅を広げるといったこともできます。itemWidth, responseWidth は Form コンポーネントの幅に対する相対値ですので、合計で 1.0 を超えないように注意してください。1.0 を超えた場合、エラーにはなりませんが表示が崩れる場合があります。

項目ごとに色を指定したい場合のために itemColor, responseColor というパラメータが用意されていますが、

	A	B	C	D	E	F	G	H	I
1	index	itemText	type	ticks	tickLabels	layout	itemWidth	responseWidth	granularity
2		見出し	heading						
3		説明文の例です	description						
4		ratingの例	rating	1,2,3,4,5	あてはま¥nらない,あて¥nはまる	horiz	0.3	0.7	
5		sliderの例	slider	1,2,3,4,5	1,2,3,4,5	horiz	0.3	0.7	0.1
6		radioの例	choice		いいえ,はい	vert	0.3	0.7	
7									

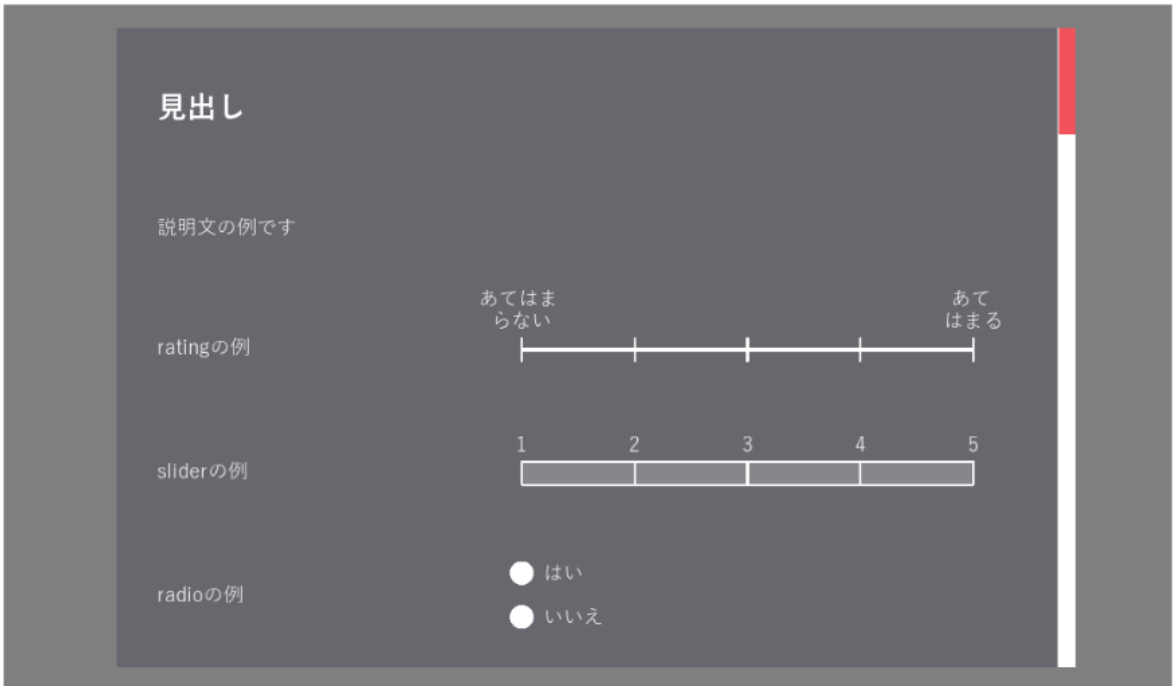
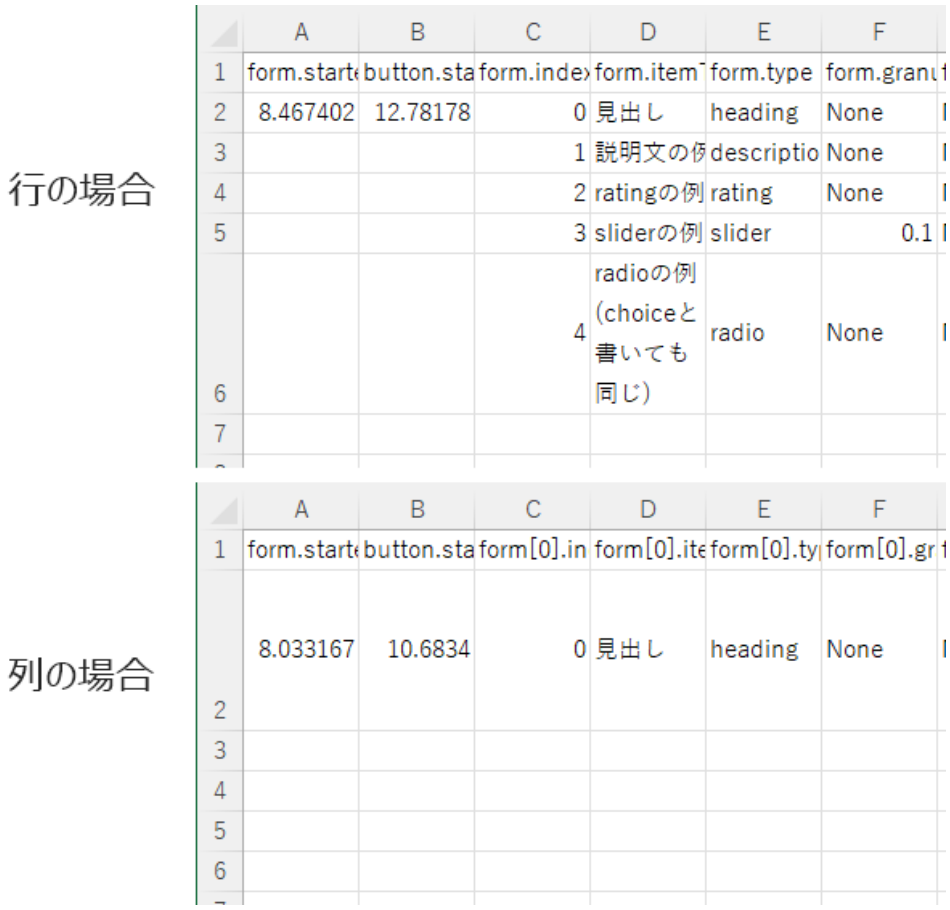


図 8.7　上：項目ファイルの内容。下：Form コンポーネントで実行した例。

これらのパラメータを有効にするには Form コンポーネントの **[スタイル]** を custom... にする必要があります。それ以外の場合、itemColor と responseColor は無視されます (書いてあってもエラーにはなりません)。font による項目ごとのフォント指定は **[スタイル]** の設定に関係なく有効です。

設定ファイルの説明はこのくらいにして、その他の注意点を挙げていきましょう。まず **[データフォーマット]** ですが、これは実験記録ファイルへのデータの出力形式を選択するパラメータです。言葉で説明するとわかりにくいので 図 8.8 をご覧ください。「行」であれば、Form コンポーネントに含まれる各 Slider の値が 1 行ずつ出力されていきます。「列」の場合は、実験内の他のパラメータと同様に、各 Slider の値が列方向にずらずらと出力されていきます。おそらく、人の目で確認するときに見やすいのは「行」です。「列」にした場合、Slider ひとつに対して複数列の値が出力されるため、実験記録ファイルの列数が非常に多くなり一画面におさまらない可能性が高いですし、評定値が出力されているセルが飛び飛びになるため Excel 上でマウスで選択して別のファイルにコピーするといった操作が非常に面倒になります。一方、本書では解説しませんが Python や R のスクリプトを書いてデータをまとめて処理する場合は、「1 試行＝ 1 行」の鉄則が守られる「列」形式の方がプログラムを書くのが楽かも知れません。実験が完成してしまえば何列目が目的の評定値なのかは変化しないのですから、その列が飛び飛びであろうとプログラムで処理する際には問題にならないでしょう。好みに応じて使い分けてください。

二つめの注意点は、**Form コンポーネントには [Routine を終了] がないので他の方法でルーチンを終了させる必要がある** という点です。この点は Keyboard コンポーネントや Button コンポーネントの **[Routine を終了]**



	A	B	C	D	E	F
1	form.start	button.sta	form.index	form.item	form.type	form.granu
2	8.467402	12.78178	0	見出し	heading	None
3			1	説明文の例	descriptio	None
4			2	ratingの例	rating	None
5			3	sliderの例	slider	0.1
6			4	radioの例(choiceと書いても同じ)	radio	None
7						

	A	B	C	D	E	F
1	form.start	button.sta	form[0].in	form[0].ite	form[0].ty	form[0].gr
2	8.033167	10.6834	0	見出し	heading	None
3						
4						
5						
6						

図 8.8　**[データフォーマット]** の設定による Form コンポーネントの出力の違い

を併用すれば解決できますが、何も考えずにこれらのコンポーネントを配置すると「(故意か誤操作かはともかく) 参加者がまだフォームに回答していないのに次のルーチンへ進んでしまう」ということが起こりかねません。実験の内容によってはそれでもいい場合もあるでしょうが、多くの用途では「すべて回答してからでなければ次へ進まない」方が望ましいのではないでしょうか。こういった時に便利なのが Form コンポーネントに対応する Form オブジェクトのデータ属性 complete です。このデータ属性は、当該フォームのすべての項目が回答済みであれば True、未回答の項目があれば False となります。これを利用して、

1. **[開始]** を「条件式」にして form.complete にすれば、form という名前の Form コンポーネントのすべての項目に回答した時点でそのコンポーネントが有効になる
2. **[終了]** を「条件式」にして form.complete にすれば、form という名前の Form コンポーネントのすべての項目に回答した時点でそのコンポーネントが終了する

といった動作が可能になります。1 は Keyboard コンポーネントや Button コンポーネントなど、**[Routine を終了]** を設定しているコンポーネントに適用すれば、すべての項目に回答し終えた後に参加者が自分でキーを押すなりボタンを押すなりしてルーチンを終了させるという使い方がよいでしょう。特に Button コンポーネントの場合、**ルーチンの開始時にはボタンが表示されていなくて、回答を終えた瞬間に次のルーチンへ進むためのボタンが出現する** という動作になるので参加者にとってもわかりやすいと思います (Keyboard コンポーネントだとキーが押せるようになったことがわかりにくい)。

2 は Form コンポーネント自身に対して適用して、すべての項目に回答し終えたら Form コンポーネントが終了するという使い方がよいでしょう。Form コンポーネント以外に教示用の Text コンポーネントなどが配置さ

れている場合、それらのコンポーネントがすべて終了するように設定しないとルーチンが終了しない点に注意してください。最後の項目に回答した瞬間にフォームが消えてしまうので参加者は最初少し驚くかもしれませんが、いちいち回答終了のためにキーやボタンを押すといった操作をしなくて済むのは楽かもしれません。最後の項目に回答した後にも参加者に回答の変更を許可したい場合は、1 の方法がよいでしょう。

最後の注意点は、**Form コンポーネントを配置したルーチンをループで繰り返す場合、2 回目以降の繰り返しでは前回の回答が残ったままになっている** という点です。回答済みを表すデータ属性 complete も True のままです。おそらくフォームへの入力内容を後続のルーチンで使用することを考えての仕様なのでしょうが、繰り返し毎に未入力の状態に戻ってほしい場面も多いでしょう。そういった場合は Form オブジェクトの reset( ) というメソッドを使用します。Code コンポーネントを配置して **[Routine 開始時]** に

```
form.reset( )
```

というコードを実行するとよいでしょう (form の部分は初期化したい Form コンポーネントの **[名前]** にする)。

コンポーネントの解説はこのくらいにして、次は Slider コンポーネントと Form コンポーネントを使用して 第 6 章 の実験をマウスで反応できるようにしてみましょう。

チェックリスト

- Slider コンポーネントで縦向き・横向きのスライダーを描画できる。
- Slider コンポーネントでラジオボタンを用いた多肢選択を描画できる。
- Slider コンポーネントで数値を回答させる際にマーカーの最小間隔を設定できる。
- Slider コンポーネントで参加者の回答を取得するコードを書ける。
- Form コンポーネントで複数の質問項目とスライダーのペアをルーチンに配置することができる。
- Form コンポーネントのすべての項目に回答したらルーチン終了のボタンを表示させることができる。
- Form コンポーネントのすべての項目に回答したらフォームを終了させることができる。
- ループの中で Form コンポーネントを使う時に繰り返しのたびに反応を初期化することができる。

8.5 概念識別の実験をマウスで反応できるようにしよう

第 7 章 の作業を最後までおこなった状態 (練習問題は除く) から作業する前提で手順を解説します。

準備作業

- 第 7 章 の実験をフォルダごとコピーして (つまり中の psyexp ファイルや xlsx ファイルもすべて含めて)、わかりやすい名前で貼り付ける。例えば 第 7 章 の実験を exp06 というフォルダにまとめている場合、まるごとコピーして exp06_form などの名前で貼り付ければよい。

実験設定ダイアログ

- 「スクリーン」タブの **[マウスカーソルを表示]** をチェックする。

instruction ルーチン

- 配置済みの Keyboard コンポーネントを削除する。

–Button コンポーネントをひとつ配置し、以下のように設定する。

* 「基本」タブの **[名前]** を `buttonInst` にして、**[終了]** を空欄にする。**[ボタンのテキスト]** を `OK` に変更する。
* 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.8, 0.04)` にする。**[位置 [x, y] $]** を `(0, -0.4)` にする。
* 「書式」タブの **[フォント]** を日本語に対応したフォントにする。**[文字の高さ $]** を `0.03` にする。
* 「データ」タブの **[クリックを記録]** を「なし」にする。

trial ルーチン

- 配置済みの textYes, textNo (それぞれ Text コンポーネント)、key_choice (Keyboard コンポーネント) を削除する。

–Slider コンポーネントを配置し、以下のように設定する。

* 「基本」タブの **[名前]** を `sliderResponse` にする。**[Routine を終了]** がチェックされていること、**[スタイル]** が rating であることを確認する (いずれも初期値でそうなっているはず)。
* 「基本」タブの **[目盛 $]** を `(0, 1, 2, 3)` にする。**[ラベル $]** に **`' いいえ',' 多分いえ',' 多分はい',' はい'`** と入力する。区切りのカンマや各ラベルを囲む`' '`は日本語入力を OFF にして (つまり半角で) 入力すること。
* 「基本」タブの **[精度 $]** を 1 にする。
* 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.8, 0.03)` にする。**[位置 [x, y] $]** を `(0, -0.3)` にする。**[反転]** をチェックする。
* 「書式」タブの **[フォント]** を日本語に対応したフォントにする。**[文字の高さ $]** を `0.03` にする。

- 配置済みの Code コンポーネントを一番下に移動させて (通常、sliderResponse を追加したことで sliderResponse が一番下になっている)、**[Routine 終了時]** のコードを以下のように変更する。

```
res = sliderResponse.getRating()
if res>1.5 and correctAns=='left':
    feedbackMsg = '正解'
elif res<1.5 and correctAns=='right':
    feedbackMsg = '正解'
else:
    feedbackMsg = '不正解'
```

feedback ルーチン

- 配置済みの Keyboard コンポーネントを削除する。
- instruction ルーチンに配置済みの buttonInst をコピーして、 `buttonFeedback` という名前で feedback ルーチンに貼り付ける。

report ルーチン

- 配置済みのコンポーネントを全て削除する。

–Form コンポーネントを配置して、以下のように設定する。

* 「基本」タブの **[名前]** を `formReport` にする。 **[項目]** は後で設定する。
* 「レイアウト」タブの **[サイズ [w, h] $]** を `(1.3, 0.7)` にする。 **[項目間の余白]** を `0.01` にする。
* 「書式」タブの **[テキストの高さ $]** を `0.03` 、 **[フォント]** を日本語に対応したフォントにする。

–instruction ルーチンに配置済みの buttonInst をコピーして、 `buttonReport` という名前で feedback ルーチンに貼り付けて以下のように設定する。

* 「基本」タブの **[終了]** を「条件式」にして、 `formReport.complete` と入力する。

項目定義ファイル

- 実験の psyexp ファイルと同じフォルダに 図 8.9 に示す内容の xlsx ファイルを作成し、report ルーチンの **[項目]** に設定する。

	A	B	C	D	E	F	G
1	index	itemText	type	options	itemWidth	responseWidth	
2		「はい」が正解となる条件は？	description				
3		眼鏡	radio	かけている,かけていない,無関係	0.2	0.7	
4		顔の輪郭	radio	丸い,四角い,無関係	0.2	0.7	
5		眼	radio	黒点のみ,白い円で囲まれている,無関係	0.2	0.7	
6		眉毛	radio	上がっている,下がっている,無関係	0.2	0.7	
7		口角	radio	上がっている,下がっている,無関係	0.2	0.7	
8							

図 8.9　実験用の項目定義ファイル

以上で作業は終了です。実行すると、 図 8.10 左のように、顔画像を見て判断する画面でスクリーンの下の方に「いいえ」「多分いいえ」「多分はい」「はい」の 4 段階の尺度が表示され、尺度上をクリックすることで回答できます。選択肢の中間をクリックすると、最も近い位置の選択肢が記録されます。すべての画像について回答を終えると、 図 8.10 右のように「はい」が正答となる条件はなんだったかを選択式で回答するフォームが表示されます。すべての項目に回答すると画面下に OK ボタンが表示されて、この OK ボタンをクリックすると実験が終了します。

おおむね前節の解説でおわかりいただける作業内容だと思いますが、OK ボタンの位置と大きさについて少し補足しておきます。sliderResponse は (0, -0.3) の位置、buttonFeedback は (0, -0.4) の位置に置かれているため、参加者は刺激画像を判断して尺度をクリックした後、フィードバックを見て少しマウスを下に動かして OK ボタンをクリックしないといけません。そして次の画像について反応するためにまた少しマウスカーソルを上に動かす必要があります。この実験を実行してみて、この上下移動を面倒に感じた方もおられるのではないでしょうか。両者の Y 座標を揃えればこの上下移動をせずに済みますが、もし **sliderResponse をクリック**

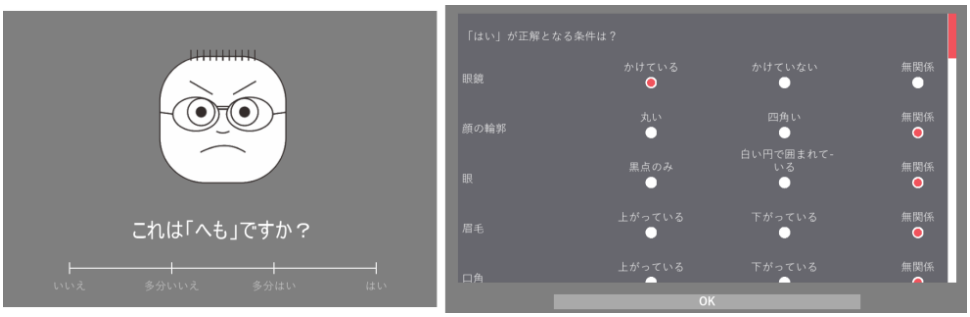


図 8.10　実験の実行画面。左は各画像について回答中。右は最後の質問フォーム。すべての項目が回答済みで OK ボタンが表示されている。

したときのマウスカーソルの位置が、フィードバック画面での buttonFeedback と重なっていたら、フィードバック画面に切り替わった瞬間に buttonFeedback をクリックしたと判定されてフィードバック画面が終了する という問題が生じます。本節で作成したように、sliderResponse と buttonFeedback が重ならないように位置をずらしておけば、この問題を避けるためなのです。少しでもマウスカーソルを動かす負担を軽減できるよう、buttonFeedback の幅を sliderResponse と同じ幅にしておいて、sliderResponse のどの位置をクリックしても左右方向にほとんどカーソルを動かさずに buttonFeedback をクリックできるようにしてあります。

どうしても上下移動を避けたい場合は、buttonFeedback の **[開始]** を 0.5 秒や 1.0 秒などに設定して、画面が切り替わった直後にはボタンが表示されないようにしておきましょう。**[開始]** を遅らせるほど問題が生じる危険性が低下しますが、参加者を待たせることになります。フィードバック画面の OK ボタンの表示を遅らせるのではなく、次の試行に入る時に完全に画面がブランクになる時間を設ける (つまり trial ルーチンのすべてのコンポーネントの **[開始]** を遅くする) 方が違和感が少ないかも知れません。「こうすれば間違いない」といった絶対的な方法はないと思いますので、各自でいろいろと試してみられるとよいと思います。

8.6 この章のトピックス

8.6.1 同一ルーチンに配置されている複数の Slider コンポーネントが回答済みか判定する

trial ルーチンに slider1, slider2 という Slider ルーチンが配置されているとします。参加者が slider1 に回答していれば、slider1.rating の値は None 以外になっています。None は Python の文法において特殊な値で、変数 x の値が None であるか判定するには x is None という式を書きます (None であれば True, なければ False)。None ではないときに真になるようにしたければ、否定演算子 not を使って x is not None と書きます。従って、以下のような式を if 文を書けば、slider1 が回答済みの時に処理を行わせるコードが書けます。

```
if slider1.rating is not None:
    # slider1 が回答済みのときにしたい処理
```

slider1, slider2 の両方が回答済みという条件にしたい場合は、論理演算子 and を使って

```
if slider1.rating is not None and slider2.rating is not None:
    # slider1 と 2 が回答済みのときにしたい処理
```

と書くことができます。さらに Slider コンポーネントの数が増えると、いちいち is not None と書くのは面倒ですし、式が長いです。そんな場合は in 演算子を使って

```
if None not in (slider1.rating, slider2.rating, slider3.rating, slider4.rating):
    # slider1 から 4 が回答済みのときにしたい処理
```

という具合に「( ) 内に None がない」という条件として書くと、is not None を繰り返さなくていい分簡潔になります。

8.6.2 日本語環境における Form コンポーネント (および TextBox コンポーネント) の文字入力機能

第 6 章 の実験では最後に「答えが『はい』になる条件は何か？」ということを口頭で自由報告させていましたが、本章ではこれを選択問題に変更しました。Form コンポーネントの使い方を解説するためとはいえ、これは実験の結果に影響しかねない大きな変更です。「自由報告させたいけど、実験者が横で聞き取って記録しなくても済むようにしたい」という場合、Builder にこだわらなければ色々方法があると思いますが (例えば紙に書かせて後から紙を回収するなど)、本書は Builder の解説本ですので、なんとか Builder で実現できる方法を考えたいところです。

とりあえずぱっと思いつくのは、PC に接続されたマイクを利用して音声を録音する Microphone コンポーネントを使うことです。ただ、音声データはファイルサイズが大きいですし、後で再生して文字起こしする手間もかかりますので、参加者が多い実験ではあまり使いたくありません (筆者の個人的な意見ですが)。他に良さげな候補は、ということで挙がってくるのが Form コンポーネントの free text です。これはフォーム上にキーボードを使って文字を入力できるボックスを設けるもので、自由に回答できるという点はクリアしていますし、録音のようにファイルサイズが大きくなる問題も後で文字起こしをするという手間もありません。第 6 章 の実験の改良として非常に魅力的なのですが、残念ながら現在 (執筆時点でバージョン 2024.2.5) の Form コンポーネントでは日本語入力のように IME(Imput Method Editor: ローマ字入力やかな入力、漢字変換などの機能を提供するソフトウェア) を使った文字入力画面をうまく表示できないのです。

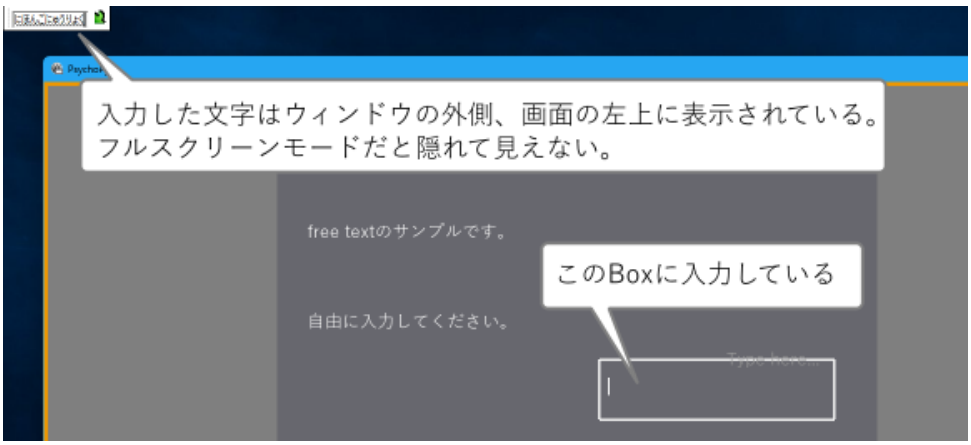


図 8.11　Form コンポーネントの free text で日本語を入力すると IME の表示がおかしくなる。

図 8.11 は free text を持つ Form コンポーネントを Pilot モードで実行して日本語を入力しようとした様子ですが (OS は Windows 11)、タイプした文字が表示されるべき右下の白い枠内には文字が表示されず、画面左上の角に小さなウィンドウが開いてその中に表示されてしまっています。この状態は不便ではありますが、「入力注の文字列が左上に表示されている」ということに気づきさえすれば、通常の操作で文字の入力は一応可能です。問題はフルスクリーンモードで実験を実行した場合で、左上に出現するウィンドウが隠れて全く見えなくなってしまいます。ひらがな程度ならなんとか入力できなくもありませんが、漢字変換は変換候補が見えな

いためお手上げです。本文で「free text は現状では日本語入力には使い物になりません」と書いたのはこのためです。

PsychoPy はマルチプラットフォームであり、Windows、MacOS、Linux などさまざまな環境をサポートしなければいけないため、IME のように OS に大きく依存する機能に対応するのは難しいです。残念ながら、おそらく数年内にこの問題が解決されて自然な日本語入力が可能になることはないでしょう。本実験のように、実験の最後に一度だけ文字入力を求めるのであれば、Builder での実験をいったん終了させてしまって何か別の方法で文字入力してもらう方が簡単でしょう。実験手続きの途中で入力が必要な場合、ちょっと抜け道的な使い方ですが「オンライン実験」をローカルブラウザで実行するという方法もあります。詳しくは「13.7: 実験中に日本語文字入力をさせたい方のために：「オンライン実験」をローカル環境でで実行する」をご覧ください。

第 9 章

マウスをさらに活用しよう―鏡映描写課題

9.1 この章の実験の概要

第 8 章 では Button, Slider, Form というマウスを使って操作する 3 つのコンポーネントを解説しました。これら 3 つのコンポーネントだけでもかなりの用途をカバーできると思いますが、長方形以外の形状のボタン (というかクリックできるオブジェクト) を使用したいとか、マウスカーソルの動きに応じて刺激を制御したいとか、もっといろいろな形でマウスを利用したい場面があると思います。本章では、鏡映描写課題を題材として、Builder での高度なマウスの活用方法を学びます。

鏡映描写課題とは、鏡に映った自分の手の像を見ながら図形をペンでなぞる課題です (図 9.1 左)。ペンでなぞる図形の上には遮蔽版が置かれていて図形を直接見ることができないようになっていて、奥に立てられた鏡に映った像を見ながら手を動かさなければいけません。鏡を見ながら描画するために前後方向（鏡に近づく、離れる）に手を動かしたときの視覚像の動きが逆転してしまうので、うまく図形をなぞるのはかなり困難です。しかし、何度も練習を繰り返していると、次第に手とその鏡像の動きの関係が学習され、素早く間違わずになぞることができるようになってきます。状況にふさわしい知覚と運動の関係を学習することを知覚運動学習といいます。鏡映描写課題は、知覚運動学習の課題として用いられます。

鏡映描写課題と類似の課題を PC で実現するために、この章では 図 9.1 右のような課題を考えます。スクリーン上に何カ所か折れ曲がった白い太線 (パス) と、小さな円 (プローブ) が提示されています。実験参加者がマウスを動かすとプローブが動きますが、スクリーンの上下方向のみ通常のマウスカーソルとは逆向きに動きます。つまり、マウスを奥に向かって動かすとプローブはスクリーンの下方向へ移動し、マウスを手前へ動かすとプローブはスクリーンの上方向へ移動します。マウスを動かすテーブルに箱か何かを置いて手元が見えないようにすれば、鏡映描写に類似した状況が得られるはずです。

もう少し実験の詳細を検討していきましょう。まず、パスをどのような形にするかを決めなければいけません。 図 9.1 は実験のアイディアを記しただけですので適当なパスを描いてありますが、実際に実験を作成するとなるときちんとした形状を決定する必要があります。特に Builder で作成するにはその大きさや中心位置を座標として計算できないといけません。また、パス間で移動すべき距離が異なるのは避けたいところです。そこで、この章では 図 9.2 のように星形の図形の辺を時計回りまたは反時計回りに進んで半周するパスを採用します。星形の中心から鋭角の頂点までの距離は 0.3 で、星形の中心がスクリーンの中心に一致するように配置します。スタート地点は鈍角の頂点から選び、星形の中心をはさんでスタート地点と向かい合う鋭角の頂点がゴール地点となります。スクリーン上でゴール地点をすぐに判別できるように、直径 0.03 のディスク (円) をゴール地点に提示します。パスは幅 0.02 の長方形を用いて描画します。スタート・ゴールの組み合わせが 5 種類、進行方向が反時計回りか時計回りかの 2 種類ありますので、合計 10 種類のパスが得られます。

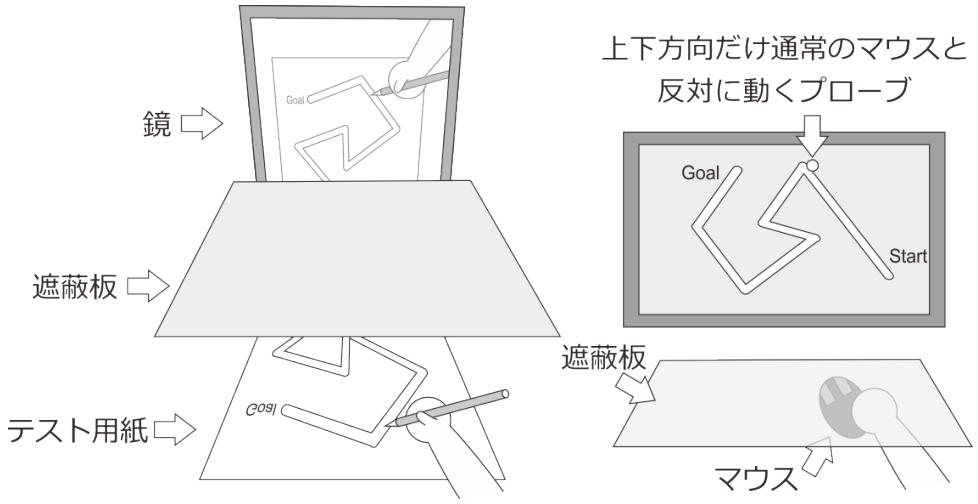


図 9.1　鏡映描写課題。遮蔽版の下に置かれたテスト用紙に描かれた図形を、鏡に映った像を見ながらペンでなぞります。スクリーンとマウスを用いてこの実験装置をシミュレートします。

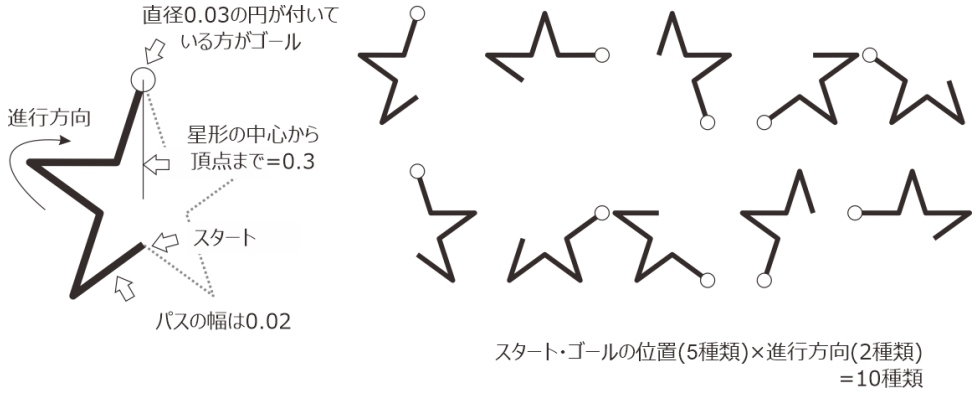


図 9.2　実験で使用するパス。星形の辺を時計回りまたは反時計回りに進んで半周します。5 種類のスタート・ゴールの組み合わせと 2 種類の進行方向で合計 10 種類のパスが得られます。

パスを描画するための座標値を計算してみましょう (図 9.3)。座標値の計算は、Builder でマウスを操作する方法を学ぶという本章の目的とは直接関係ありませんので、よくわからなければ次の段落まで読み流していただいて構いません。まず、長方形を用いてパスを描画するのですから、Builder では Polygon コンポーネントを使うのが適切です。Polygon コンポーネントで星形の辺を描画するには、その辺の長さと中点の座標が必要です。 図 9.3 では、辺 AE の長さと、辺 AE の中点である点 F の座標が必要な値です。点 A と B の座標がすぐに求まること、B と E の Y 座標が等しいこと、辺 AC が y=-tan(72°) x+300 上にあることを利用すれば、F の座標は計算できます。点 F の座標が得られれば Y 軸を挟んで向かい合った点 G の座標が直ちに得られます。点 F と G の座標がわかれば、原点を中心としてこれらの座標を 72 度ずつ回転させればすべての辺の中点が得られます。点 F の座標を求める途中で点 A と E の座標が得られますので、辺の長さも計算可能です。

座標値を計算して、図 9.2 の 10 通りのパスを描けるように Builder の条件ファイルの形で並べたものが 図 9.4 です。pathXpos(X=1 から 5) が各辺の中心の座標で、 **[位置 [x, y] $]** の値として使用します。pathXori は **[回転角度 $]** に使用する回転角です。startPos と goalPos はそれぞれスタート地点とゴール地点の座標です。このファイルを exp09cnd.xlsx という名前で保存して使用します。

パスにおける各辺の長さと幅はすべて等しいので、条件ファイルを利用せずに Builder 上で直接 **[サイズ [w, h] $]** に値を入力することにします。 図 9.3 から計算した辺の長さは 0.218 なのですが、ぴったり 0.218 にする

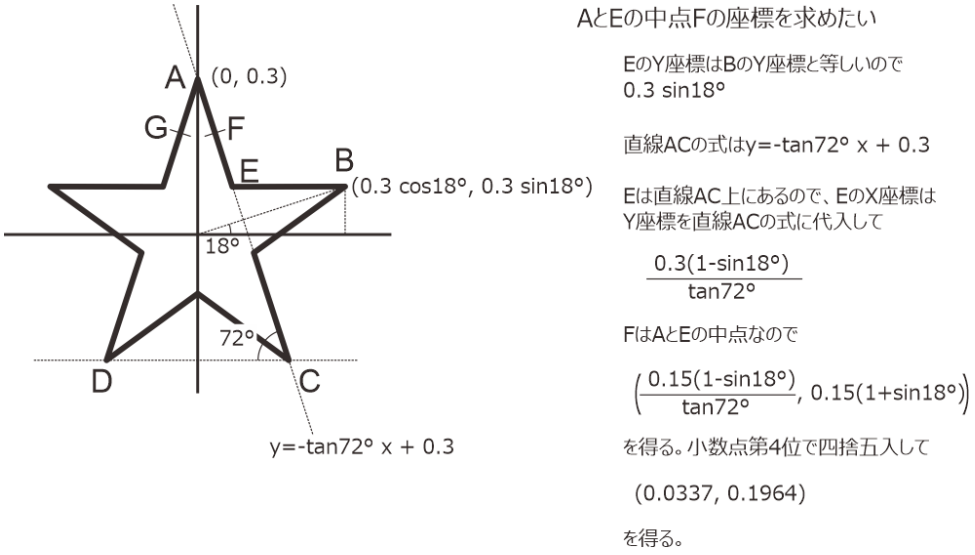


図 9.3　星形を描画するための座標値の計算。F の座標が得られれば G の座標も直ちに得られます。F と G の座標を 72 度ずつ回転させればすべての辺の中点が得られます。

	A	B	C	D	E	F	G	H	I	J	K	L
1	path1pos	path1ori	path2pos	path2ori	path3pos	path3ori	path4pos	path4ori	path5pos	path5ori	startPos	goalPos
2	(0.1763,0.0927)	0	(0.1971,0.0286)	-36	(0.1427,-0.1391)	-108	(0.0882,-0.1786)	-144	(-0.0882,-0.1786)	-36	(0.0674,0.0927)	(-0.1763,-0.2427)
3	(0.1427,-0.1391)	-108	(0.0882,-0.1786)	-144	(-0.0882,-0.1786)	-36	(-0.1427,-0.1391)	-72	(-0.1971,0.0286)	-144	(0.1090,-0.0354)	(-0.2853,0.0927)
4	(-0.0882,-0.1786)	-36	(-0.1427,-0.1391)	-72	(-0.1971,0.0286)	-144	(-0.1763,0.0927)	0	(-0.0337,0.1964)	-72	(0.0000,-0.1146)	(0.0000,0.3000)
5	(-0.1971,0.0286)	-144	(-0.1763,0.0927)	0	(-0.0337,0.1964)	-72	(0.0337,0.1964)	-108	(0.1763,0.0927)	0	(-0.1090,-0.0354)	(0.2853,0.0927)
6	(-0.0337,0.1964)	-72	(0.0337,0.1964)	-108	(0.1763,0.0927)	0	(0.1971,0.0286)	-36	(0.1427,-0.1391)	-108	(-0.0674,0.0927)	(0.1763,-0.2427)
7	(0.0337,0.1964)	-108	(-0.0337,0.1964)	-72	(-0.1763,0.0927)	0	(-0.1971,0.0286)	-144	(-0.1427,-0.1391)	-72	(0.0674,0.0927)	(-0.1763,-0.2427)
8	(-0.1763,0.0927)	0	(-0.1971,0.0286)	-144	(-0.1427,-0.1391)	-72	(-0.0882,-0.1786)	-36	(0.0882,-0.1786)	-144	(-0.0674,0.0927)	(0.1763,-0.2427)
9	(-0.1427,-0.1391)	-72	(-0.0882,-0.1786)	-36	(0.0882,-0.1786)	-144	(0.1427,-0.1391)	-108	(0.1971,0.0286)	-36	(-0.1090,-0.0354)	(0.2853,0.0927)
10	(0.0882,-0.1786)	-144	(0.1427,-0.1391)	-108	(0.1971,0.0286)	-36	(0.1763,0.0927)	0	(0.0337,0.1964)	-108	(0.0000,-0.1146)	(0.0000,0.3000)
11	(0.1971,0.0286)	-36	(0.1763,0.0927)	0	(0.0337,0.1964)	-108	(-0.0337,0.1964)	-72	(-0.1763,0.0927)	0	(0.1090,-0.0354)	(-0.2853,0.0927)

図 9.4　本章で用いる条件ファイル (http://s12600.net/psy/python/ppb/index.html#ppb-files からダウンロードできます)

と頂点のところでパスが狭くなってしまうので、余裕を持たせて辺の長さは 0.24 にしましょう。パスの幅は 0.02 なので、 **[サイズ [w, h] $]** に指定する値は (0.24, 0.02) です。

これで提示する図形についてはほぼ決まりましたが、まだプローブの大きさと色を決めていません。プローブの直径は 0.01 とします。プローブの色は、パスを白色で描画することを考慮すると、パスと重なっていても区別できるように白以外の色にする必要があります。パスや背景と区別が付けば何色でも構わないのですが、ここでは白い輪郭に赤色の塗りつぶしで描画することにしましょう。ついでに、ゴール地点に提示する円もわかりやすいように、白い輪郭に緑色の塗りつぶしで描画することにしましょう。

続いて実験の手続きを決定しないといけませんが、以下の解説では Builder の使い方を学ぶという目的を最優先して、ごくシンプルな手続きだけを作成します。 図 9.5 をご覧ください。各試行の最初に、プローブと共に、ゴール地点を示す円と色と大きさが等しい円をスタート地点に提示します。実験参加者がマウスの左ボタンをクリックすると、スタート地点の円がゴール地点へ移動すると同時にパスが提示されます。実験参加者はマウスを操作して、できる限りプローブがパスからはみ出ないように注意しながらプローブをゴール地点まで動かします。プローブの中心がゴール地点の円内に入ったことが検出された時点で試行は終了します。試行終了後は直ちに次の試行へ進みます。以上の手続きを、exp09cnd.xlsx に定義されている 10 種類の条件に対して 1 回ずつ、合計 10 試行を無作為な順序で実施したら実験は終了です。

この課題を実験実習などの授業で使用するならば、鏡映描写課題を練習した群としなかった群で比較したり、

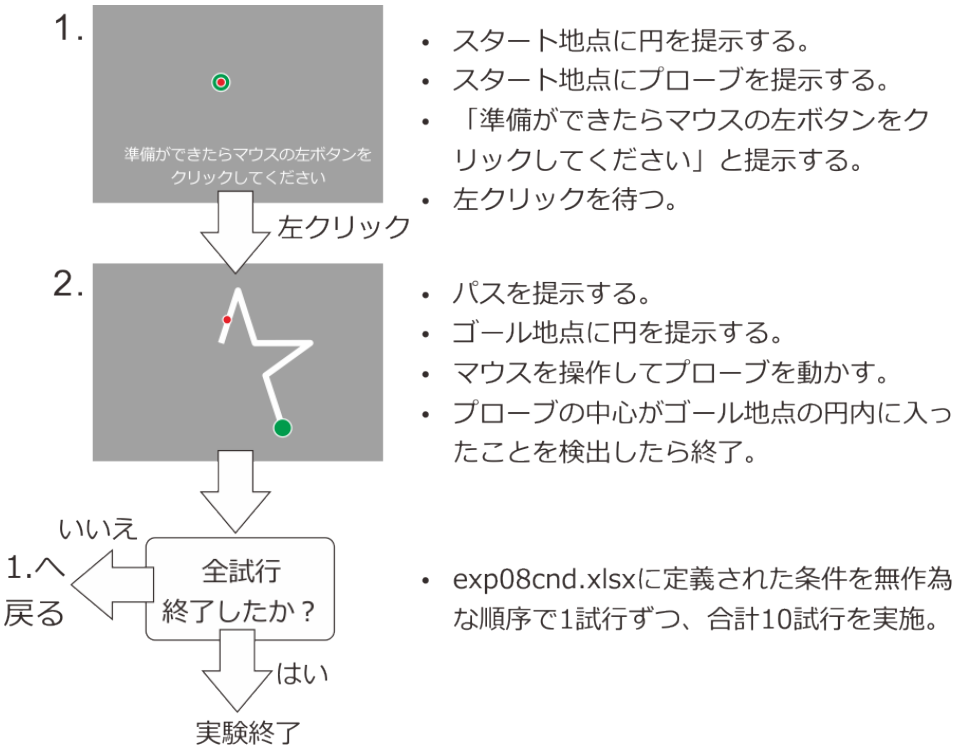


図 9.5　実験の流れ。

非利き手で練習して学習の効果が利き手にどの程度転移するかを調べたりするなどの工夫をする必要があります。この辺りは各自で工夫してみてください。

さて、以上で作成する実験の内容が決まりました。作成方法の解説を始めたいと思いますが、その前に本章で初登場の Mouse コンポーネントについて解説しておきましょう。

9.2 Mouse コンポーネント

Mouse コンポーネントは、その名の通り Builder からマウスを利用するためのコンポーネントです。3 ボタンのマウスを想定しているので、4 つ以上ボタンがあるマウスの場合は押されたことを検出できないボタンが出てきてしまいます。一般的なマウスの場合、3 つのボタンはそれぞれ「左クリック」と「右クリック」に使うボタンと、これらのボタンの間にあるボタンに対応します。 Mouse コンポーネントのアイコンは「反応」カテゴリにあります (図 9.6)。

Mouse コンポーネントのプロパティ設定ダイアログのうち、「基本」タブの **[名前]**、**[開始]**、**[終了]** は今までのコンポーネントと共通ですので解説は必要ないでしょう。 **[ボタン押しで Routine を終了]** はボタン押しでルーチンを終了するか否かを指定します。「なし」はボタンを押してもルーチンを終了しないということですが、ルーチンを終了させたい場合は「全てのクリック」と「有効なクリック」のどちらかを使用します。「全てのクリック」は文字通り、マウスボタンがクリックされたらどのような状況でも終了します。「有効なクリック」は **[クリック可能な視覚刺激 $]** で指定した刺激上でクリックされた場合のみルーチンを終了します。 **[クリック可能な視覚刺激 $]** は解説が長くなるのでトピックで扱います。「 9.14.4:**[クリック可能な視覚刺激 $]** の活用 」をご覧ください。**[新たなクリックのみ検出]** がチェックされていれば、マウスのボタン押し検出が始まった時点ですでにマウスのボタンが押されていた場合に、いったんボタンを離してから押し直さないと検出しません。実験参加者がマウスのボタンを押しっぱなしにしていた場合にいきなりルーチンが終了してしまうことを防げます。

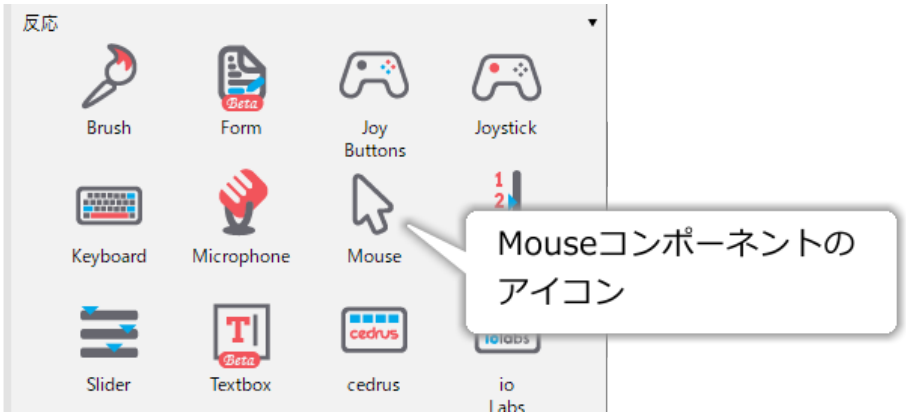


図 9.6　Mouse コンポーネントのアイコン。

続いて「データ」タブです。**[マウスの状態を保存]** はマウスのどのような情報が保存されるかを指定します。選択可能な項目については 表 9.1 をご覧ください。 **[時刻の基準]** は Mouse コンポーネントで使用される時刻をルーチン開始時 (routine)、実験開始時、Mouse コンポーネント開始時のいずれのタイミングで 0 にするかを指定します。Mouse コンポーネント開始時というのは、その Mouse コンポーネントの **[開始]** で指定されたタイミングで 0 にするということです。ルーチンの途中から Mouse コンポーネントが有効になって、その時点を 0 秒としたい場合に便利です。

表 9.1　Mouse コンポーネントの **[マウスの状態を保存]** で選択可能な値

値	概要
最終	ルーチン終了時のマウスの状態が保存されます。 **[名前]** が foo の場合、実験記録ファイルの foo.x と foo.y にマウスカーソルの X 座標と Y 座標、foo.leftButton、foo.midButton、foo.rightButton にマウスの左ボタン、中央ボタン、右ボタンの状態が保存されます (0=押されていない、1=押されている)。座標の単位は実験設定ダイアログの **[単位]** に従います。
クリック時	マウスのいずれかのボタンが押されるたびに、その時のマウスの状態が保存されます。「最終」の時に出力される項目に加えて、押された時刻を保存する foo.time(foo は **[名前]** に入力した文字列) という項目が実験記録ファイルに出力されます。ボタンを押しっぱなしにしていると、その間 1 フレーム毎に状態が保存されていきます。
フレーム毎	マウスが有効な間 (**[開始]** から **[終了]** の間)、フレーム毎にマウスの状態が保存されます。保存形式は「クリック時」と同じです。60Hz のモニターを使っている場合 1 秒間に 60 件もデータが記録されますのでご注意ください。
なし	マウスの状態を保存しません。

注意する必要があるのは **[マウスの状態を保存]** の値によってどのようなデータが保存されるかです。まず、「最終」ではボタン押しの有無にかかわらず、とにかくルーチン終了時のマウスの状態が保存されます。この点は Keyboard コンポーネントと異なっています。マウスカーソルの座標の単位は実験設定ダイアログの **[単位]** に従います。

「クリック時」では、 **[開始]** と **[終了]** で指定された期間内にマウスのボタンを押すとマウスの状態が記録されます。このように書くと「クリックした回数だけ記録される」と思われてしまうかも知れませんが、そうで

はなくて「ボタンが押されている間ずっとフレーム毎に」記録され続けます。つまり、リフレッシュレートが 60Hz のモニターを使用していてマウスのボタンを 1 回、1 秒間押し続けると、1 件ではなく 60 件のデータが保存されます。実験記録ファイルには、カーソルの X 座標の値が [0.48, 0.49, 0.51]、Y 座標の値が [0.09,-0.03, -0.11] といった具合に Python のリストの形で保存されたデータが出力されています。この例の場合、ボタンが押されたときのマウスカーソルの座標は記録された順番に [0.48, 0.09]、[0.49, -0.03]、[0.51, -0.11] だったということです。このように 3 件のデータある場合、「素早く 3 回クリックされた」のか「1 回、3 フレーム分の時間ボタンを押した」のか「1 回素早くクリックし、1 回 2 フレーム分の時間ボタンを押した」のかを区別するには、ボタンが押された時刻の出力を確認する必要があります。 表 9.1 に記したとおり、時刻は実験記録ファイルの foo.time という列に同じ書式で出力されています (foo は **[名前]** に指定した文字列)。

また、当然ですが **[ボタン押しで Routine を終了]** がチェックされている状態では、「クリック時」が設定されていていても、ボタン押しが検出された時点でルーチンが終了してしまうので 1 件しかデータが保存されません。

続いて「フレーム毎」ですが、これを選択すると「クリック時」と同様のデータが Mouse コンポーネントが有効な期間中ずっと記録され続けます。最後の「なし」を選択すると、Mouse コンポーネントは実験記録ファイルに何も出力しません。

これから作成する実験では、「最終」でルーチン終了時の状態を記録しても意味がありません。「クリック時」にしたら参加者が操作した軌跡が保存されるので役に立ちそうな気がしますが、実際に保存させるには実験参加者にずっとマウスのボタンを押さえつづけてもらわないといけません。参加者への負担になりますし、何よりちゃんと押してくれなかった場合にデータが取得できないのでは話になりません。「フレーム毎」にすると軌跡は保存されますが、同時にマウスのボタンの状態も保存されて記録ファイルが非常に大きくなってしまいます。操作記録には Code コンポーネントを使うことにして、とりあえず「なし」を使う事にします。

Mouse コンポーネントにはまだ設定項目がありますが、解説はここでいったん区切りとして実験の作成に入りましょう。

チェックリスト

- ルーチン終了時のマウスカーソルの位置およびマウスのボタンの状態を記録することができる。
- ボタンが押された時点のマウスカーソルの位置およびマウスのボタンの状態を記録することができる。
- 実験記録ファイルにマウスカーソルの座標やボタンの状態がリストとして複数件出力されている時に、個々のデータの座標値やその取得時刻を判断できる。

9.3 実験の作成

まずは前章までに解説済みのテクニックで作成できる部分を作成します。

準備作業

- この章の実験のためのフォルダを作成して、その中に exp09.psyexp という名前で新しい実験を保存する。
- 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。

  - PsychoPy の設定で height 以外の単位を標準に設定している場合は、実験設定ダイアログの「スクリーン」タブの **[単位]** を height にしておく。

- trial ルーチン
  - Mouse コンポーネントをひとつ配置して、 **[名前]** を `mouseTrial` にする。
    - 「基本」タブの **[終了]** を空白にする。
    - 「基本」タブの **[ボタン押しで Routine を終了]** を「なし」にする。
    - 「データ」タブの **[マウスの状態を保存]** を「なし」にする。
  - Code コンポーネントをひとつ配置し、 **[名前]** を `codeTrial` にする。
  - Polygon コンポーネントを 1 つ配置して以下の作業を行う。 これらの作業によって path1、path2、path3、path4、path5 という 5 つの Polygon コンポーネントが作成される。
    - **[名前]** を `path1` にする。
    - **[終了]** を空白にする。
    - **[形状]** を長方形にする。
    - **[塗りつぶしの色]** を `white` にする。
    - **[回転角度 $]** を `path1ori` にして、「繰り返し毎に更新」に設定する。
    - **[位置 [x, y] $]** を `path1pos` にして、「繰り返し毎に更新」に設定する。
    - **[サイズ [w, h] $]** を `(0.24, 0.02)` に設定する。
    - path1 をコピーして「Component の貼り付け」で貼り付け、**[名前]** を `path2` にする。 **[回転角度 $]** を `path2ori` 、 **[位置 [x, y] $]** を `path2pos` にする。以後、同様の手順で `path3`, `path4`, `path5` を作成する。 **[回転角度 $]** と **[位置 [x, y] $]** も **[名前]** に合わせて `path2ori`, `path2pos`, `path3ori`, `path3pos`,... と変更する。
  - Polygon コンポーネントを 2 つ配置して **[終了]** を空白にし、**[名前]** をそれぞれ `goalDisc` 、`probe` にする。必ず goalDisc は path1〜path5 より上に描画されるようにし、probe は goalDisc よりも上に描画されるようにする。
  - goalDisc について以下の作業を行う。
    - **[終了]** を空白にする。
    - **[形状]** を円にする。
    - **[塗りつぶしの色]** を `green` にする。
    - **[位置 [x, y] $]** を `goalPos` にし、「繰り返し毎に更新」に設定する。
    - **[サイズ [w, h] $]** を `(0.03, 0.03)` にする。
  - probe について以下の作業を行う。
    - **[終了]** を空白にする。

        * **[形状]** を円にする。
        * **[塗りつぶしの色]** を `red` にする。
        * **[位置 [x, y] $]** を `[px, py]` にし、「フレーム毎に更新する」に設定する。
        * **[サイズ [w, h] $]** を `(0.01, 0.01)` にする。
- ready ルーチン (作成する)
    - フローの trial ルーチンの前に挿入する。
    - Mouse コンポーネントをひとつ配置して、 **[名前]** を `mouseReady` にする。
        * **[終了]** を空白にする。
        * **[ボタン押しで Routine を終了]** が「すべてのクリック」になっていることを確認する。
        * **[マウスの状態を保存]** を「なし」にする。
    - Polygon コンポーネントを 2 つ配置して **[終了]** を空白にし、 **[名前]** をそれぞれ `startDisc` 、`probeReady` にする。probeReady は startDisc よりも上に描画されるようにする。
    - startDisc について以下の作業を行う。
        * **[形状]** を円にする。
        * **[塗りつぶしの色]** を `green` にする。
        * **[位置 [x, y] $]** を `startPos` にし、「繰り返し毎に更新」に設定する。
        * **[サイズ [w, h] $]** を `(0.03, 0.03)` にする。
    - probeReady について以下の作業を行う。
        * **[形状]** を円にする。
        * **[塗りつぶしの色]** を `red` にする。
        * **[位置 [x, y] $]** を `startPos` にし、「繰り返し毎に更新」に設定する。
        * **[サイズ [w, h] $]** を `(0.01, 0.01)` にする。
    - Text コンポーネントをひとつ配置する。
        * **[文字の高さ $]** を `0.03` にする。
        * **[位置 [x, y] $]** `(0, -0.32)` にする。
        * **[文字列]** に **準備ができたらマウスの左ボタンをクリックしてください** と入力する。
- trials ループ (作成する)
    - ready ルーチンと trials ルーチンを繰り返すように挿入する。
    - **[繰り返し回数 $]** を 1 にする。
    - **[条件]** に exp09cnd.xlsx を指定する。

ここまで準備できれば、あとは ready ルーチンと trials ルーチンの Code コンポーネントに必要なコードを記入するだけです。まず、trials ルーチンの Code コンポーネントで実現すべき処理を整理して、何が今まで学んできたことではできないのかをはっきりさせましょう。まだ exp09.psyexp で実現できていない処理は以下の 3 点です。

1. 現在のマウスカーソルの座標を得る。
2. マウスカーソルの動きを上下反転させた場合の座標を計算する。
3. 反転させた座標が goalDisc に含まれているか判定し、含まれていたらルーチンを終了する。
4. 試行開始時にマウスカーソルをスタート地点に移動させる。

1 は未解説の処理です。先ほど解説した Mouse コンポーネントの機能には、ルーチン実行中に座標を得るという機能は含まれていませんでした。2 もちょっと考える必要がありそうです。3 は 第 6 章 、第 7 章 で扱ってきた if 文が利用できそうです。「指定した条件に当てはまればルーチンを終了する」という処理は 第 7 章 ですでに解説しました。ですから、残っているのは「座標が goalDisc に含まれているか否かをどうやって判定すればよいか」という問題だけです。4 も未解説の処理ですね。

1 と 4 の問題を解決するには、PsychoPy のマウス制御用クラスである psychopy.event.Mouse クラス (以下 Mouse クラス) を利用する必要があります。Buildern の Mouse コンポーネントもこのクラスを利用しています。まずは Mouse クラスの解説から始めることにしましょう。

9.4 psychopy.event.Mouse クラスのメソッドを利用してマウスの状態を取得しよう

第 6 章 で解説したように、Builder 上で Grating コンポーネントを配置すると、その **[名前]** に設定した変数に Grating クラスのインスタンスが格納されます。それと同様に、Builder 上で Mouse コンポーネントを配置すると **[名前]** に設定した変数に Mouse クラスのインスタンスが格納されます。現在作成中の exp09.psyexp では、trial ルーチンに mouseTrial という **[名前]** の Mouse コンポーネントを配置したのですから、Code コンポーネントからは mouseTrial という変数を通して trial ルーチン上の Mouse クラスを利用することができます。

Mouse クラスは、データ属性に直接アクセスするのではなく、メソッドを通してすべての処理を行うように設計されています。ですから、ここではメソッドのみを紹介しましょう 表 9.2 に代表的な Mouse クラスの主なメソッドを示します。

表 9.2　psychopy.event.Mouse クラスの主なメソッド

メソッド, 概要	
clickReset(buttons=(0,1,2))	マウスの反応時間計測用タイマーをリセットします。buttons にはリセットしたいボタンを並べたリストを指定します。0=左ボタン、1 ＝中央のボタン、2=右ボタンです。
getPos()	現在のマウスカーソルの座標をリストとして取得します。リストの最初の要素は X 座標、次の要素は Y 座標を示しています。
getPressed(getTime=False)	現在のボタンの状態をリストとして取得します。リストの要素は順番に左ボタン、中央のボタン、右ボタンに対応していて、0=押されていない、1=押されている、です。getTime=True とすると、最後に clickReset を呼んでからの時間も返されます。
setPos(newPos=(0,0))	マウスカーソルの位置を newPos に移動させます。
setVisible(visible)	visible に True を指定すると、マウスカーソルが描画されます。False を指定すると、マウスカーソルは描画されません。
isPressedIn(shape, buttons=(0,1,2))	引数 shape で指定したオブジェクトの内部にマウスカーソルがある状態で buttons に指定したボタンが押されていれば True、押されていなければ False を返します。
getRel()	最後に getPos() または getRel() を実行したときのマウスカーソル位置を基準として、現在のマウスカーソルの相対位置を表すリストを返します。リストの最初の要素は X 座標、次の要素は Y 座標を示しています。
getWheelRel()	最後に getWheelRel() を実行したときのマウスホイール位置を基準として、マウスホイールの移動量を表すリストを返します。多くの場合、リストは 2 次元で 2 番目の要素に移動量が格納されていますが、使用しているマウスのデバイスドライバに依存します。

getPos( ) と getPressed( ) を利用すれば、Mouse コンポーネントで出力される程度の情報を取得することができます。注目してほしいのは、clickReset( ) や getPressed( ) におけるボタンの番号です。clickReset( ) では、マウスの左ボタンが 0、中央のボタンが 1、右ボタンが 2 で表されます。一方、getPressed( ) の戻り値として得られるマウスボタンの状態は [0, 0, 1] といった具合に 0 または 1 が 3 つ並んだリストで、1 番目が左ボタン、2 番目が中央のボタン、3 番目が右ボタンに対応しています。clickReset( ) のボタン番号と getPressed( ) の戻り値のリストにおけるボタンの順番が一致しているのは一目瞭然ですが、なぜボタン番号が 1、2、3 ではなく 0、1、2 なのでしょうか。それは、Python の文法ではリストの要素の順番を数える時には 1 からではなく 0 から始めるからです。ですから、Python にとっては 0、1、2 と番号が割り当てられている方が自然なのです。「0 から数え始める」ということは次節以降の内容を理解する上で非常に重要なので必ず覚えておいてください。

以上を踏まえて Builder の作業に戻りましょう。exp09.psyexp の trial ルーチンを開いて、codeTrial の **[フレーム毎]** に以下のコードを入力してください。

```
mousePos = mouseTrial.getPos()
```

これで、フレーム毎に mousePos という変数にマウスカーソルの位置が代入されるようになりました。続いて probe の動きをマウスカーソルの動きと上下方向だけ反対にする方法を考えます。

なお、ひとつ補足しておきますと、今回とは違ってマウスカーソルの位置に合わせて刺激の位置を変更するのであれば、Code コンポーネントを使う必要はありません。動かしたい刺激の **[位置 [x, y] $]** に以下のように入力して「フレーム毎に更新する」に設定するだけで目的を達成できます (Mouse コンポーネントの **[名前]** は mouseTrial とする)。

```
mouseTrial.getPos( )
```

なぜこれだけで済むか、おわかりでしょうか。 getPos( ) メソッドの戻り値は X 座標、Y 座標の値を並べたリストであり、 **[位置 [x, y] $]** に記述すべき値とデータの型が一致しているから、というのが理由です。関数やメソッドを見た時に、その戻り値の型を思い浮かべる習慣をつけるようにしてください。

チェックリスト

- Code コンポーネントでマウスカーソルの座標を取得するコードを記述できる。
- Code コンポーネントでマウスのボタンの状態を取得するコードを記述できる。
- getPressed() メソッドを用いて取得したマウスのボタンの状態を示すリストの各要素がどのボタンに対応しているか答えられる。また、リストの各要素の値とボタンの状態の関係を答えられる。
- Code コンポーネントを用いずにマウスカーソルの位置に刺激を描画することができる。

9.5 リストの要素にアクセスしてマウスカーソルと上下反対にプローブを移動させよう

変数 mousePos に現在のマウスカーソルの座標を代入することができたので、次はマウスカーソルと上下反対にプローブを移動させる方法を考えましょう。 図 9.7 の左図をご覧ください。マウスカーソルと上下方向だけ反対にプローブが移動するということは、マウスカーソルが (dx, dy) 移動したときにプローブは (dx, -dy) 移動するということです。

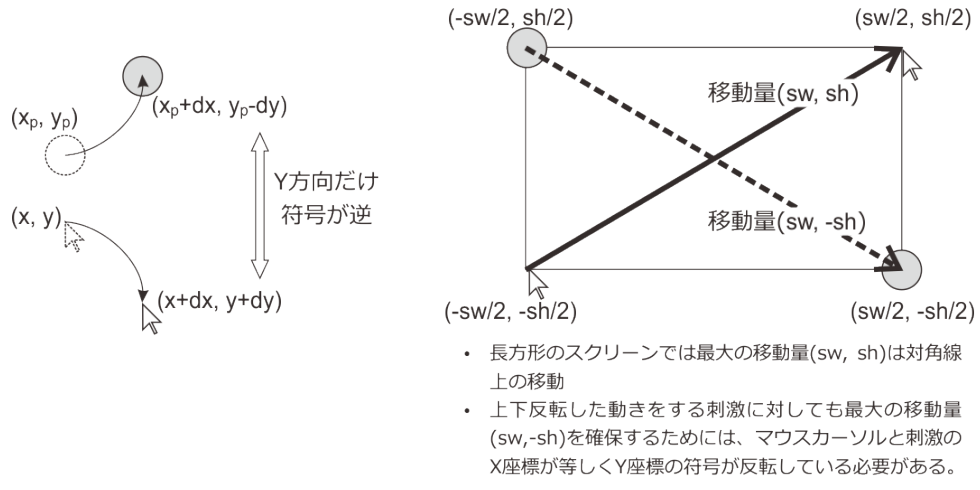


図 9.7　マウスカーソルと上下方向だけ反転して移動するプローブ。マウスカーソルの Y 座標の符号を反転した座標をプローブの座標とすれば、最大の移動量 (対角線上の運動) に対応することができます。ただし sw、sh はスクリーン幅、スクリーン高を表しているとします。

この要件さえ満たせばどのようにプローブの座標を決定しても構わないのですが、ここではマウスカーソルを

最大限移動させてもプローブが追従できるように座標を決定する方法を考えます。PsychoPy でサポートしているスクリーンは長方形ですから、マウスカーソルの移動量が最大となるのは対角線上を頂点から頂点まで移動させた時です。マウスカーソルを幅 sw、高さ sh のスクリーンの左下から右上の頂点へ移動させるとすると、マウスカーソルの座標は (-sw/2, -sh/2) から (sw/2, sh/2) まで移動し、移動量は (sw, sh) です。この時、プローブは (sw, -sh) だけ移動しないといけませんが、(sw, -sh) の移動量を確保するためにはプローブは (-sw/2, sh/2) から (sw/2, -sh/2) へ移動する以外あり得ません。これらの座標を比較すると、マウスカーソルの Y 座標の符号を反転した座標をプローブの座標とすれば、最大の移動量に対応できることがわかります (図 9.7 右)。

以上を踏まえたうえで、Code コンポーネントに入力するコードを考えましょう。マウスカーソルの X 座標と Y 座標がそれぞれ mx、my という変数に格納されているのであれば、[mx, -my] と書けば目的を達成できます。しかし、今回は mousePos という変数に X 座標と Y 座標をまとめてひとつのリストとして代入されているので、このような書き方ができません。mousePos に代入されたリストから、X 座標と Y 座標を個別に抜き出す必要があります。

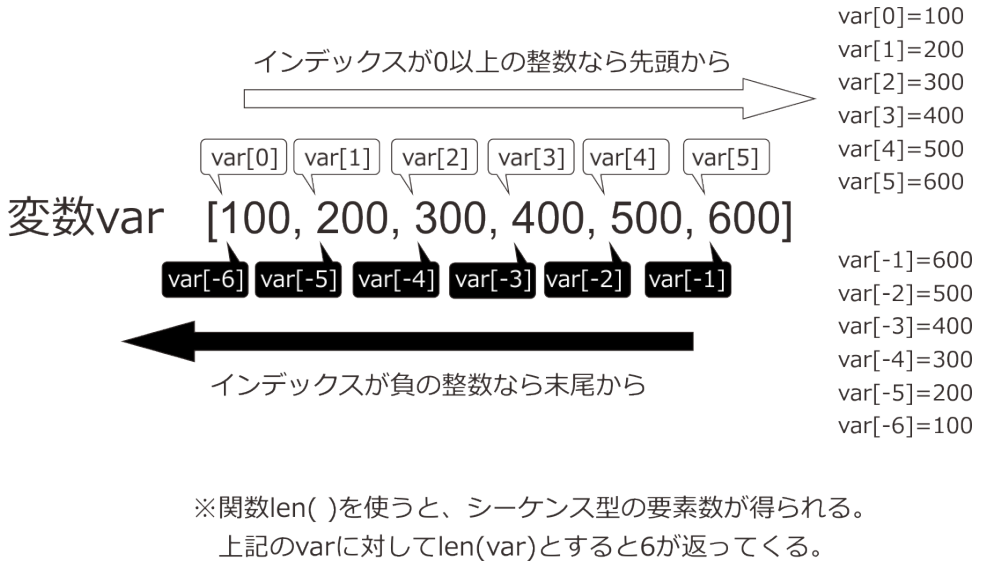


図 9.8　シーケンス型のデータからの要素の取出し。インデックスが 0 以上の整数なら先頭から、負の整数なら末尾から数えます。

Python の文法において、リストをはじめとするシーケンス型のデータから一部の要素を取り出すには、[ ] 演算子を使用します。[ ] 演算子は、mousePos[1] という具合にシーケンス型のデータの後に添えて、[ ] の中に何番目の要素を取り出すかを指定します。この指定のことをインデックスと呼びます。 図 9.8 にインデックスとして整数を指定した時の動作を示します。インデックスが 0 以上の整数の場合、最初の要素を 0 として順番に先頭 (左) から数えた位置にある要素を取り出します。現在変数に格納されているシーケンス型データに何個の要素が含まれているかわからない場合は len( ) という関数を用います。len( ) は、引数に与えられたデータに含まれる要素数を返します。インデックスとして負の整数が指定された場合 (負のインデックス) は、最後の要素を-1 として末尾 (右) から数えた位置にある要素を取り出します。0 から数えるというのは先ほど getPressed( ) メソッドについて解説した時に述べたとおりですね。[ ] 演算子を使用して、以下のようなコードを書くと変数 mousePos から X 座標、Y 座標を取り出してそれぞれ変数 px、py に代入することができます。

```
px = mousePos[0]
py = mousePos[1]
```

今回は Y 座標の符号を反転してプローブの座標として用いたいのですから、以下のように-演算子を組み合わ

せて代入と符号反転を一度に済ませると便利です。

```
px = mousePos[0]
py = -mousePos[1]
```

これらの文を exp08.py の trial ルーチンの codeTrial の **[フレーム毎]** に追加してください。追加後の **[フレーム毎]** は以下のようになります。

```
mousePos = mouseTrial.getPos( )
px = mousePos[0]
py = -mousePos[1]
```

trial ルーチンの probe の **[位置 [x, y] $]** に [px, py] と設定されておりますので、これでマウスカーソルと上下反対方向に移動するプローブが完成しました。この節の目標は達成されましたが、せっかく [ ] 演算子が出てきましたので、シーケンス型の扱いについてもう少し学んでおきましょう。まず、今回のように符号の反転などの操作が必要ないのであれば、以下の書式で、関数やメソッドの戻り値として渡されたシーケンス型データを要素に分解して別々の変数に代入することができます。

```
px, py = mouseTrial.getPos( )
```

この書式を使用する場合、=演算子の左辺にカンマ区切りで並べられた変数の個数と、戻り値として渡されるシーケンス型データの要素数が一致している必要があります。一致していない場合、エラーでスクリプトの処理が停止してしまいます。

続いて補足しておきたいのは、多重シーケンスからの要素の取出しです。シーケンス型は非常に柔軟なデータ型で、入れ子のようにシーケンスの要素としてシーケンスを含むことができます。このような多重のシーケンスに対して [ ] 演算子を適用した例を 図 9.9 に示します。この例では ['A', 'B', 'C'] というリストと [1, 2, 3, 4] というリストを要素とするリストが変数 var に格納されています。var[0] は ['A', 'B', 'C']、var[1] は [1, 2, 3, 4] です。var[0] はリストなのですから、これに対してさらに [ ] を適用することができて、var[0][0] は'A'、var[0][1] は'B' です。同様に、var[1][3] とすると 4 が得られます。もちろん負のインデックスも組み合わせて使用できますので、var[1][3] は var[1][-1] と書くこともできます。

少しややこしいという注意が必要な点として、リストを作る [ ] とシーケンス型の要素を取り出す [ ] 演算子の違いがあります (図 9.10)。リストを作る [ ] の前には変数も値もありません。一方、要素を取り出す [ ] 演算子は必ずその前にシーケンス型のデータがあります。「シーケンス型のデータがあればよい」ということは、[0.24, 0.02][0] という具合に変数ではなくリスト等のデータが前に直接置かれていても構わないということです。 [0.24, 0.02][0] は「[0.24, 0.02] という要素数 2 のリストの最初の要素」ということですから、その値は 0.24 です。よろしいでしょうか？

シーケンス型に対する [ ] 演算子の使い方には解説したいテクニックがたくさんあるのですが、どんどんこの章の実験から離れていってしまいますので、ひとまずこのくらいにしておきましょう。ここで述べただけでも Builder を便利に拡張する様々なコードを書くことができます。終わりと言いつつ最後にひとつだけ付け足しておきますと、[ ] 演算子は文字列に対しても同様に使用することができます (というか文字列はシーケンス型の仲間です)。'Psychology' という文字列が格納されている変数 s に s[3] とすると'c' が得られますし、s[-2] とすると'g' が得られます。覚えておいてください。

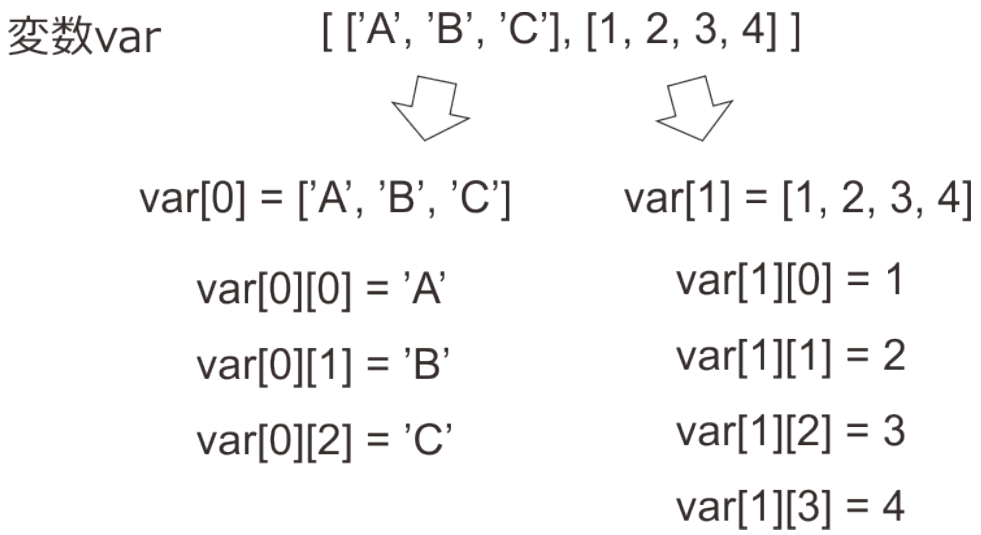


図 9.9　多重シーケンスからの要素の取出し。var[0] とするとリストが得られるので、さらに [ ] 演算子を適用して var[0][0] とすると var[0] の最初の要素を取り出せます。

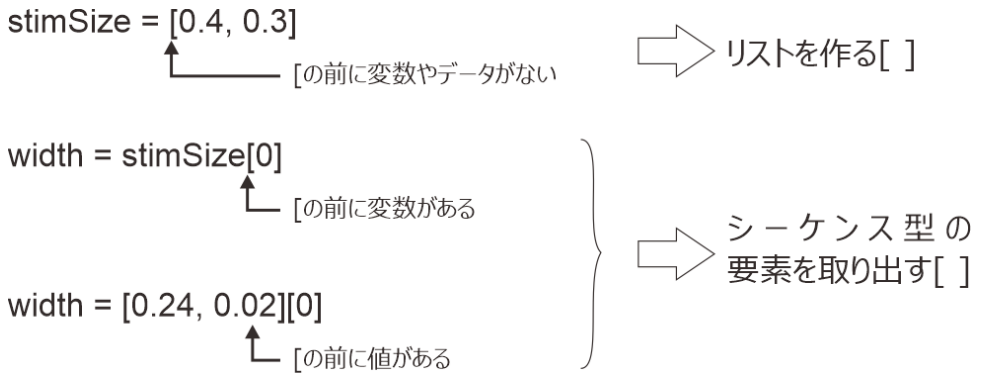


図 9.10　リストを作る [ ] とシーケンス型の要素を取り出す [ ]。

チェックリスト

- シーケンス型のデータから要素をひとつ取り出して他の変数に代入したり関数の引数に使ったりすることができる。
- シーケンス型データの前から N 番目の要素を取り出す時の式を答えられる。
- シーケンス型データの後ろから N 番目の要素を取り出す時の式を答えられる。
- 関数を用いてシーケンス型データに含まれる要素数を調べることができる。
- シーケンス型のデータが入れ子構造になっている時に、要素であるシーケンス型データや、さらにその要素を取り出すことができる。
- 文字列の中から N 番目の文字を取り出して変数に代入したり関数の引数に使ったりすることができる。
- [1,2,3][0] といった具合に [から始まって、] の後に [ ] 演算子が続く式を評価した時の値を答えられる。

9.6 刺激の重なりを判定しよう

プローブの動きが実現できたので、続いてプローブの中心がゴール地点の円内に入ったことを判定する方法を考えましょう。これは「プローブの座標がゴール地点の座標を中心とした半径 0.015 の円内に入る」ことと等しいので、以下の式で判定することができます。さっそく [ ] 演算子の復習になっているので、よくわからなければ前節を復習してください。

```
(goalPos[0]-px)**2 + (goalPos[1]-py)**2  < 0.015**2
```

念のため補足しておきますが、ゴール地点の座標は変数 goalPos に格納されていて、ゴール地点に置かれている Polygon コンポーネント goalDisc の **[サイズ [w, h] $]** は (0.03, 0.03) です。 **[サイズ [w, h] $]** は刺激の幅と高さ、すなわち直径に相当しますので、半径は 0.015 です。この式を if 文の条件式に用いれば目的は達成できるのですが、この節ではもっと便利な方法を学びましょう。

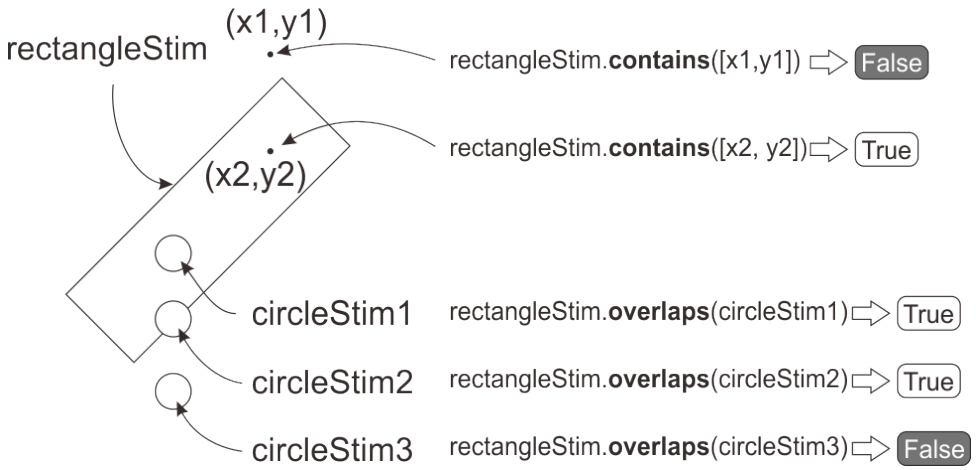


図 9.11　Contains( ) と Overlaps( ) メソッドの例。

Polygon コンポーネントに対応するクラスが複数あるのは 第 6 章 で述べたとおりですが、 **[頂点数]** が 3 以上の時に用いられるクラスでは、contains( ) と overlaps( ) というメソッドが用意されています。これは今回の実験にはうってつけのメソッドで、contains( ) は引数に指定された座標が内側に含まれていれば True、いなければ False を返します。overlaps( ) は、引数に指定された Polygon コンポーネントと重なっていれば True、いなければ False を返します。文章で説明するより図の方がわかりやすいでしょう。 図 9.11 は rectangle という名前の Polygon コンポーネントから contains( ) と overlaps( ) を呼び出した例を示しています。判定の対象となる図形が 図 9.11 のように傾いている場合、先ほどのような簡単な条件式では図形の中に点が含まれているかを判定することはできませんので、contains( ) と overlaps( ) は非常にありがたいメソッドです。もちろん図形が三角形や五角形など、Polygon コンポーネントで描画できる多角形であれば適切に判定が行われます。

ここまで解説が進めば、もう「プローブの中心がゴール地点の円内に入ったらルーチンを終了する」という動作を実現するのは簡単です。円内に入った判定には contains( ) を使用し、ルーチンを終了するには continueRoutine に False を代入すればいいのですから

```
if goalDisc.contains([px, py]):
    continueRoutine = False
```

とすればよいはずです。このコードを、trial ルーチンの codeTrial の **[フレーム毎]** に追加しましょう。念のた

め、追加後のコード全体を示しておきます。これで trial ルーチンは完成しました。

```
mousePos = mouseTrial.getPos( )
px = mousePos[0]
py = -mousePos[1]
if goalDisc.contains([px, py]):
    continueRoutine = False
```

これでマウスカーソルの現在位置の取得、マウスカーソルと上下反転した移動をするプローブ、プローブとゴール地点との重なりを判定してルーチンの終了、の課題をクリアしました。ここで一度、exp09.psyexp を保存して実行してみてください。狙い通りの動作が実現できているはずです。残るは試行開始時のマウスカーソルの位置を設定するだけです。

チェックリスト

- ある座標が Polygon コンポーネントで描画した多角形の内側に含まれているか判定するコードを書くことができる。
- Polygon コンポーネントで描画した二つの多角形に重なっている部分があるが判定するコードを書くことができる。

9.7 カーソルの位置を設定し、カーソルの表示 ON/OFF を制御しよう

Mouse クラスのメソッドを使用すれば、カーソル位置の設定は何も難しいことはありません。 表 9.2 で紹介済みの setPos( ) メソッドを使用します。

注意しないといけないのは、今回の実験では「スタート位置にプローブを置くためには、マウスカーソルの位置はスタート位置の座標と上下反対の位置になければいけないという点です。前節ではマウスカーソルの位置をプローブ位置に変換しましたが、逆にプローブ位置をマウスカーソルの位置に変換しないといけないわけです。

変換といっても単に Y 座標の符号を反転すればよいだけですから、処理は非常に単純です。startPos にスタート位置の座標が格納されているのですから、[startPos[0], -startPos[1]] とすればよいでしょう。これを setPos( ) の引数にすればよいのですから、以下のコードを実行すればプローブのスタート位置に対応する位置へマウスカーソルを設定することができます。

```
mouseTrial.setPos([startPos[0], -startPos[1]])
```

試行開始時にカーソルがスタート位置にないといけないのですから、このコードは trial ルーチンの開始時に実行するべきです。trial ルーチンに配置してある codeTrial の **[Routine 開始時]** に追加しましょう。

これで実験の基本的な部分は完成ですが、実験の間ずっとマウスカーソルが表示されたままになっているのが気になります。実験設定ダイアログに Show mouse というマウスカーソルを表示させるための項目はあるのですが、このチェックを外してもマウスカーソルは消えません。 Show mouse は Mouse コンポーネントを使っていない時にマウスカーソルを表示したい (ウィンドウモードで他アプリケーションと一緒に使用したいときなど) 場合に使用するためのもので、Mouse コンポーネントを使用するとマウスカーソルは強制的に表示されます。まあ普通はマウスを使用する時にはカーソルが表示されているのが当然なのでこのような仕様に

なっているのでしょうが、今回の実験の場合では余計です。マウスカーソルを非表示にするには、 表 9.2 の setVisible( ) メソッドを使用します。以下のコードを trial ルーチンの Code コンポーネントの **[実験開始時]** に入力してください。

```
mouseTrial.setVisible(False)
mouseReady.setVisible(False)
```

入力を終えたら、exp09.psyexp を保存して実行してください。今度はマウスカーソルが表示されないはずです。このコードを追加してもなおカーソルが表示される場合は、実験設定ダイアログの **[キーボードバックエンド]** の設定を確認してください (筆者の環境では **[キーボードバックエンド]** が ioHub に設定されているとsetVisible() が無効になる場合がありました)。

これで 図 9.5 に示した実験の流れは完成しましたが、現在の状態では参加者の反応が一切記録されません。心理学実験の実習であれば、ストップウォッチなどを利用しながら手作業で記録するのも良いのですが、せっかく PC を使っているのですからやはり記録も PC で行いたいところです。次節では、反応の記録方法について考えてみましょう。

チェックリスト

- マウスカーソルの位置を設定するコードを書くことができる
- マウスカーソルの表示 ON/OFF を切り替えるコードを書くことができる。

9.8 for 文を用いて複数の対象に作業を繰り返そう

今回の実験では、実験参加者のどのような反応を記録すればよいでしょうか。真っ先に思い浮かぶのは、課題遂行中の (つまり trial ルーチン実行中の) プローブの軌跡をすべて記録することです。しかし、ただプローブの軌跡を記録しただけでは、プローブがパスの上にきちんと乗っているのか後から計算しないといけません。パスに用いる長方形の中心座標値の計算 (図 9.3) よりもさらに面倒な計算なので、できることならしたくありません。せっかく contains( )、overlaps( ) というメソッドを覚えたのですから、これを利用してプローブがパス上にあるか否かを判定しましょう。

今回の課題では、パスを構成する Polygon コンポーネントが 5 個あり、このいずれかの上にプローブがあれば、パス上にプローブが乗っています。すでに今まで学んだ Python の文法の範囲でも何通りかの方法でこの処理を記述することができますが、ここでは以下のようなコードを考えてみましょう。

```
onPath = False
if path1.contains([px, py]):
    onPath = True
if path2.contains([px, py]):
    onPath = True
if path3.contains([px, py]):
    onPath = True
if path4.contains([px, py]):
    onPath = True
if path5.contains([px, py]):
```

(次のページに続く)

(前のページからの続き)

```
    onPath = True
```

最初に onPath という変数に False を代入しておいて、path1 から path5 まで順番に contains( ) を実行して戻り値が True であれば onPath を True にします。最後の行まで進んだ時点で onPath が True であればいずれかの上にプローブが乗っています。2 つ目の if からは elif のほうが効率がいいじゃないの？と思われた方は良い点に注目しておられますが、以下の解説と対応づけるためにわざとこうしていますのでご理解ください。

さて、このコードは確かに目的とする処理は達成できるのですが、、同じような文がだらだらと続いて読みづらいですし、「やっぱり contains( ) じゃなくて overlaps( ) を使うことにしよう」などと思ったときに書き換えが面倒です。うっかり 1 行だけ書き換え忘れたりすると、間違えた場所を見つけるのは非常に厄介です。この例のように「変数が異なるだけで同じ処理を繰り返す」時に非常に有効な for という文が Python には用意されています。

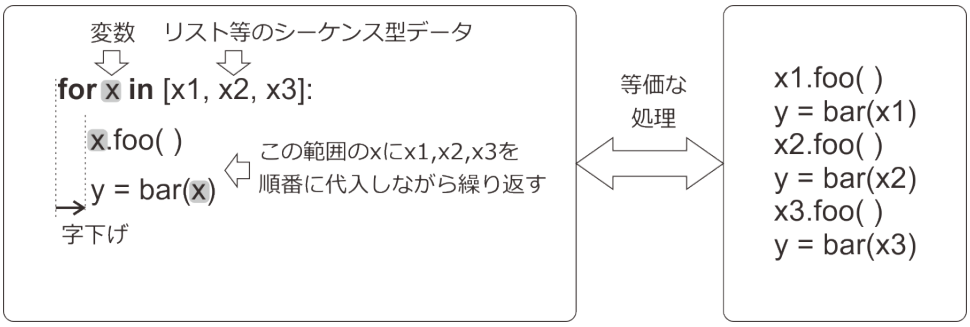


図 9.12　for 文による繰り返し処理。

図 9.12 に for 文の例を示します。for 文は「for 変数 in シーケンス型データ:」という形で使用し、シーケンス型データの先頭から要素をひとつずつ取り出して変数に代入しながら処理を繰り返します。繰り返しの範囲は if 文と同様に、字下げ量が for の出現する行と同じかそれより少ない行の手前まで及びます。また、for が出て来る行の最後にコロンが必要な点も if 文と同じなので忘れないようにしてください。 図 9.12 左の例では、for に続く変数として x を使用していますので、x に x1 を代入して繰り返し処理を行い、続いて x に x2 を代入して繰り返し処理を…という具合に実行されます。結果的に、 図 9.12 左の for 文は、 図 9.12 右に示したコードと等価な処理を行います。for 文を使うと、path1 から path5 まで contains( ) メソッドを実行するコードは以下のように短縮できます。

```
onPath = False
for path in [path1, path2, path3, path4, path5]:
    if path.contains([px, py]):
        onPath = True
```

とりあえずこれで「プローブがパス上にあるか判定する」という目標は達成されましたが、for 文についてもう少し勉強しておきましょう。今回は 5 個ある長方形のどれにプローブが乗っていても構わないので、もし path1 の上にプローブが乗っていれば (すなわち path1.contains([px, py]) の戻り値が True であれば)、path2 以降について contains( ) を実行する必要がありません。今時の PC の性能であれば残り 4 回余分に contains( ) を実行しても大した負担になりませんが、PC の処理遅延による刺激の描画漏れや反応時間計測の遅延が起こる可能性を少しでも下げたいと思うのであれば PC に余分な処理をさせたくありません。このように、for 文を最後まで繰り返す前に「これ以上 for 文を繰り返す必要はない」という状況になった場合は、break という

文を使います。 図 9.13 に break の使用例を示します。for 文による繰り返し処理を遂行中に break が実行されると、まだ繰り返すべき処理が残っていても直ちに for 文が終了します。 図 9.13 の場合、プローブの中心が path3 の上に乗っていますので、path に path1、path2 を代入して実行している時は if 文の式が True になりません。break は if 文の中にあるので実行されず、次の繰り返しへ進みます。そして path に path3 が代入された時、if 文の式が True になるので break が実行され、直ちに for 文による繰り返しが終了します。したがって、path4、path5 に対する処理は実行されません。

この節最初で if 文をずらずらと並べたコードを示したときに「2 番目以降は elif 文のほうが効率がいいんじゃないのか？」と思った方は、この break による for 文の中断が elif による効率化と対応することがおわかり頂けると思います。

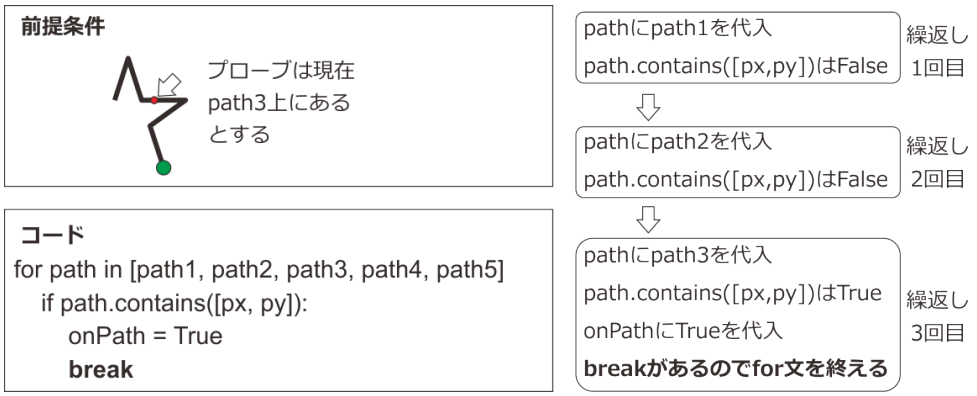


図 9.13　break による for 文の中断。if 文などと組み合わせて使用するのが一般的です。

この章の実験を完成させるためにはここまで学んでいれば十分なのですが、今後皆さんが自分の実験を作成する時のための知識を高める、という観点でもうひとつ for 文について紹介しておきたい点があります。少々高度な内容を例題として取り上げますので、その前にいったん節を区切りましょう。

チェックリスト

- ある変数に格納されている値だけが異なる処理を繰り返し実行しなければいけない時に、for 文を用いて記述することができる。
- for 文を継続する必要がなくなった時に直ちに終了させることができる。

9.9 ルーチンに含まれる全コンポーネントのリストから必要なものを判別して処理しよう

この節で紹介するコードは、この章の実験の完成版では使用しませんので、ここまで作業してきた exp09.psyexp に手を加えずに読み進めてください。コードを入力して実行させたい場合は、exp09.psyexp を別名で保存して作業してください。

それでは for 文の勉強を再開しましょう。for 文の実行を制御す文として、break と併せて覚えておきたいのが continue です。continue は、現在実行中の繰り返しを終了させて、次の繰り返しに移行させる時に使用します。 図 9.14 は break と continue の働きを示した図です。break はもう for 文をこれ以上続ける必要がないときに、continue は現在の要素に対してはもう処理する必要がないけれども残りの要素に対して処理を続ける必要あるときに使用します。この節では、continue を使用例として、「ルーチンに含まれるコンポーネントから特定のコンポーネントを探して処理をする」という処理をするコードを作成します。この「特定のコンポーネ

ントを探す」という処理が少々難解なので、「とりあえず Builder の基本的な使い方をマスターして自分の実験を作ってみたい」という方はよくわからなくても先に進んでいただいて構いません。

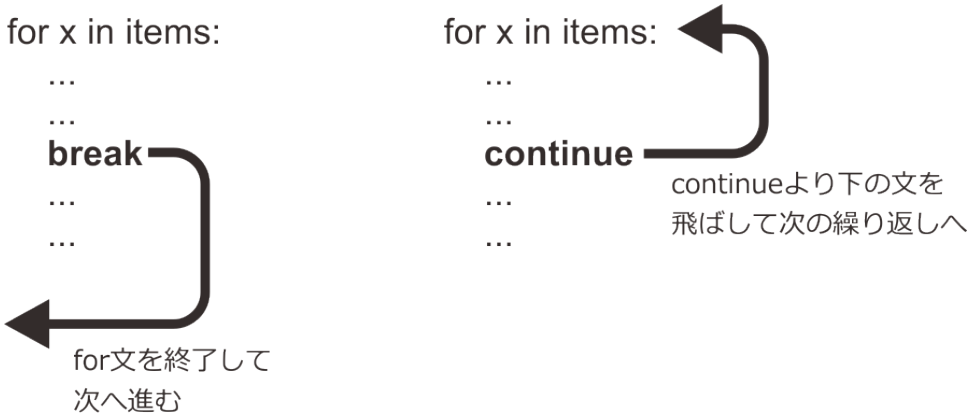


図 9.14　break と continue の違い。残りの要素に対して処理をする必要がない場合は break、必要がある場合は continue を使用します。

例題の題材は前節に引き続き「プローブがパス上にあるか判定する」処理です。前節ではパスを構成する Polygon コンポーネントのオブジェクトを並べたリストを手で入力して for 文へ渡しましたが、オブジェクトの個数が多くなるとリストが長くなって後から読み返す時に非常に読みにくいです。また、コンポーネントの名前を変更したら入力済みのコードを全部修正必要があり、間違いが生じやすいです。

Builder は、この問題を解消するのにうってつけの内部変数を持っています。Builder は各ルーチンの処理を開始する前に、「ルーチン名 +Components」という名前の変数に、ルーチン内に配置されている全てのコンポーネント (正確に言うとコンポーネントに対応するクラスのインスタンス：「 9.14.1:Builder の内部変数 trialComponents に含まれるオブジェクトについて 」参照) を列挙したリストを格納します。trial ルーチンでしたら変数名は trialComponents、ready ルーチンでしたら readyComponents です。この変数を for c in trialComponents: という具合に for 文に渡せば、ルーチンに含まれるインスタンスを持つすべてのコンポーネントに対して処理を行うことができるのです。そうすればコンポーネントの数が増えてもコードの長さは変わりませんし、 **[名前]** を変更した時にいちいちコードを修正する必要もありません。

ただ、今回の「プローブがパス上にあるか判定する」という処理に変数 trialComponents を利用するには、少々面倒な問題を解決する必要があります。trialComponents には trial ルーチンに置かれたすべてのコンポーネントが列挙されているので、Mouse コンポーネントやプローブ自身、ゴール地点の円も含まれています。これらに対して contains( ) を実行しようとすると問題が生じるので、パスを構成している Polygon コンポーネントに対してだけ処理をしなければいけません。このような時に、continue は非常に有効です。具体的には以下のように contains( ) を実行する前に、現在処理しようとしているコンポーネントが path1 から path5 ではない時には continue を実行します。このようにすれば、continue 以下の文が実行されずに for 文は次の繰り返しへ進みます。

```
onPath = False
for c in trialComponents:
    if (path1 から path5 ではないことを判別する式)
        continue
    if c.contains([px, py]):
        onPath = True
```

(次のページに続く)

(前のページからの続き)

```
        break
```

残る問題は、現在 c に代入されている要素が path1 から path5 でないかをどうやって判別するかです。これには PsychoPy の刺激描画用クラスが持っている name というデータ属性を使用します。データ属性 name には対応する Builder のコンポーネントの **[名前]** に入力した文字列が格納されているので、name に'path' という文字列が含まれているかを判別すればよいでしょう。この判別にはシーケンス中にある要素が含まれるか否かを判定するときに用いた in 演算子が使用できます (第 7 章 参照)。

```
'path' in c.name
```

c.name に'path' という文字列が含まれていれば、この式は True になります。これで万全、と言いたいところなのですが、残念ながらもう一工夫必要です。データ属性 name は PsychoPy の刺激描画用クラスが持っているものであり、trial ルーチンで使われている Mouse クラスには name がないのです。ですから、c に Mouse クラスのインスタンスが代入されている状態で'path' in c.name を実行すると「name というデータ属性はありません」というエラーが出て実験が停止してしまいます。この問題に対応するためには、まず c に格納されているオブジェクトが name というデータ属性を持っていることを確認する必要があります。確認のためには hasattr( ) という関数を使います。hasattr( ) は 2 個の引数を持ち、第 1 引数に指定されたオブジェクトが第 2 引数に指定された名前のデータ属性を持っていれば True が返されます。以下のように使用すると、c に格納されているオブジェクトが name というデータ属性を持っている時に True の値を得ることができます。

```
hasattr(c, 'name')
```

この二つの条件式を組み合わせれば、c が'path' という文字列を **[名前]** に含むコンポーネント (に対応するインスタンス) であるか判別できます。どちらか一方の条件式を満たさないオブジェクトはパスを構成している Polygon コンポーネントではありませんので、

```
not hasattr(c, 'name') or not 'path' in c.name
```

が True となった時には continue を使って次のオブジェクトへ処理を進めればよいということになります (or の前後はこの順序である必要があります：「 9.14.2: 論理式の評価順序について 」参照)。

```
if not hasattr(c, 'name') or not 'path' in c.name:
    continue
```

このコードを for 文に追加したものを 図 9.15 左に示します。コードがどのように動作するのかを追った流れをその右側に示してあります。continue の働きを確認してください。

for 文にはまだまだ解説したい機能があるのですが、いくらなんでも脱線しすぎですのでそろそろ実験の作成に戻りましょう。次節では、プローブがパス上にあるかを判定した結果とプローブの座標を実験記録ファイルに出力することに取り組みます。

チェックリスト

- for 文で現在処理中の要素に対してこれ以上処理を続ける必要がなくなった時に、次の要素の処理へ直ちに移行するコードを書くことができる。

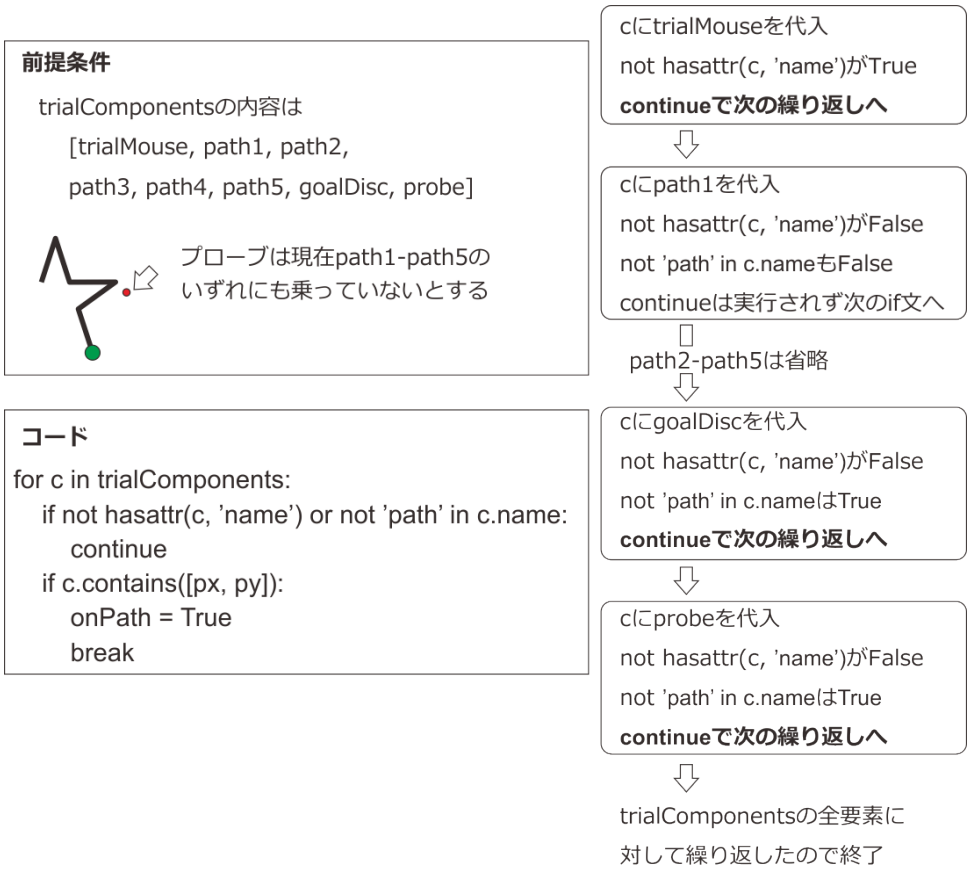


図 9.15　continue による繰り返し処理のスキップ。

- ルーチン内に含まれるコンポーネントの一覧を格納した Builder の内部変数の名前を答えられる。
- オブジェクトがある名前のデータ属性 (たとえば'foo') を持っているか判別するコードを書くことができる。
- ある文字列が別の文字列の中に含まれているか (例えば'psych' が'psychophysics' に含まれるか) 判別するコードを書くことができる。

9.10 リストにデータを追加してマウスの軌跡を保存しよう

この章の実験作成も最終段階です。プローブの軌跡と、プローブがパス上にあるかを判定した結果を実験記録ファイルに出力しましょう。具体的には、Mouse コンポーネントの出力のように、実験記録ファイルに probe_x、probe_y、on_path という列を設けて、そこへそれぞれ X 座標、Y 座標、判定結果の値を並べたリストを出力したいと思います。

この章までにリストは何度も扱ってきましたが、以前の章に出てきたリストはすべて事前に長さが決まっていました。**[位置 [x, y] $]** に入力する座標でしたら長さは 2 ですし、**[前景色]** に RGB で色を指定するのでしたら長さは 3 です。それに対して、今回実験記録ファイルに出力しようとしているリストは事前に長さがわかりません。実験参加者が課題遂行に時間がかかればかかるほど、プローブの座標を記録したリストは長くなります。課題遂行に要する時間が予測できないので、記録に必要なリストの長さも予測できないのです。このような場合に有効なのが、リストの要素を追加したり削除したりする機能です。実は今までリストと呼んできたデータは Python に標準で備わっている List というクラスのインスタンスであり、List は PsychoPy の Mouse

クラスなどと同様にメソッドを持っています。 表 9.3 に List クラスの主なメソッドを示します。これらのメソッドを活用すれば問題を解決できそうです。

表 9.3　List クラスの主なメソッド

メソッド, 概要	
append(x)	リストの末尾に新たな要素として x を追加します。リストの長さは 1 増加します。
extend(seq)	リストの末尾に新たな要素として seq の要素を追加します。seq はシーケンス型や文字列型など、長さが定義されるデータ型である必要があります。リストの長さは seq の長さ分増加します。
insert(i, x)	リストの i 番目の要素として x を追加します。i は整数でなければいけません。
remove(x)	リストの先頭からチェックして、最初に現れた x を削除します。x がリストに含まれていない場合はエラーになります。
index(x)	リストの先頭からチェックして、最初に現れた x のインデックスを返します。x がリストに含まれていない場合はエラーになります。
count(x)	リストに x が何個含まれているかを返します。
sort( )	リストの要素を昇順に並び替えます。
reverse( )	リストの要素を現在の順番と逆順に並び替えます。

さて、じっくりと 表 9.3 を眺めると、要素を追加するメソッドとして append( ) と extend( ) の 2 種類があることがわかります。両者の違いをまとめたものが 図 9.16 です。ポイントは、append( ) は引数をそのままひとつの要素として付け加えるのに対して、extend( ) は引数の要素を付け加えます。ですから、append( ) を実行すると必ずリストの要素が 1 個増えます。それに対して extend( ) の場合は引数に含まれる要素数だけリストの要素が増えます。また、extend( ) は長さがないデータ、すなわち [ ] 演算子を付けて要素を取り出すことができないデータを引数にすることはできません。具体的には数値や真偽値 (True/False) などは extend( ) の引数にできません。今回の目的では、保存しようとしているプローブの X 座標、Y 座標は数値、判定結果は真偽値ですから、いずれも extend( ) で追加することができません。append( ) を使用しましょう。

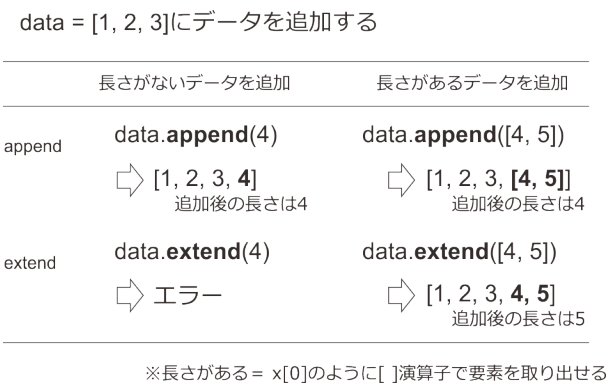


図 9.16　append( ) と extend( ) によるリストへのデータの追加。

それではコードを入力していきます。まず、exp09.psyexp を開いて trial ルーチンの Code コンポーネントのプロパティ設定ダイアログを開いてください。 **[Routine 開始時]** に以下のコードを追加してください。

```
probeX_list = [ ]
probeY_list = [ ]
onPath_list = [ ]
```

ここでは、trial ルーチン開始時にこれから実行する試行のデータを追加していくためのリストを用意しています。=演算子の右辺に奇妙な式が出てきましたが、 図 9.10 をよく思い出してください。式の中に [ ] が出てきた場合、[の直前が変数やデータであれば、要素を取り出す [ ] です。そうでなければ、リストを作成する [ ] です。上記の式の場合、[の前は=演算子であって変数やデータではありませんので、これはリストを作成する [ ] です。[ ] の中身に何も書いてありませんから、中身が空っぽ (長さは 0) のリストが作成されます。

続いて、 **[フレーム毎]** に入力済みのコードの最後に以下のコードを追加します。

```
probeX_list.append(px)
probeY_list.append(py)
onPath_list.append(onPath)
```

ここでは新しいプローブ座標 (px, py) が得られるたびに、先ほど作成したリストにプローブの X 座標、Y 座標、判定結果を追加しています。そして最後に、 **[Routine 終了時]** でこれらのリストを実験記録ファイルに出力します。この方法については 第 7 章 ですでに解説済みですので、わからない人は 第 7 章 をしっかり復習してください。

```
trials.addData('probe_x', probeX_list)
trials.addData('probe_y', probeY_list)
trials.addData('on_path', onPath_list)
```

probeX_list、probeY_list、onPath_list の内容は、実験記録ファイルに出力さえしてしまえばもう用済みです。ですから、次の試行を開始する時には内容を初期化 (=空っぽに) してしまっても構いません。このように、現在実行中のルーチン内でだけ必要なリストは、 **[Routine 開始時]** で作成するのが定番です。一方、実験を通じてデータを蓄積して、実験終了時に何かの処理をするためのリストを作成するのであれば、 **[実験開始時]** で作成するべきです。

これで作業は完了しました。exp09.psyexp を保存して実行し、数試行実行してみてください。適当なところで Escape キーを押して実験を終了し、Excel で trial-by-trial 記録ファイルを開いてみましょう。 図 9.17 のように probe_x、probe_y、on_Path という列が追加されていて、Python のリストのように [ ] で囲まれたカンマ区切りの値が並んでいるはずです。Excel にはこれらの値がまとめて文字列として認識されてひとつのセルに収められています。ちょっと面倒ですが、これらのセルの文字列を別の場所にコピーして「データの区切り位置」でカンマを区切り文字に指定すれば、ひとつひとつの値が 1 個のセルを占めるように変換できます。この状態まで変換すれば、後は 図 9.18 のように Excel の機能でグラフをプロットしたり、さまざまな関数を利用して分析したりできます。

これで完成！と言いたいところですが、この節で解説した方式だと一般的な 60fps のモニターを使用している場合 1 秒ごとに 60 件ものデータが追加されてしまうので、参加者がじっくりと課題に取り組む人の場合、膨大な量のデータが出力されてしまいます。軌跡をどの程度詳細に分析するかに寄って適切なサンプリング頻度は異なりますが、おおよその軌跡が把握できればよいだけでしたら、1 秒間に 10 件程度でも充分でしょう。次節では、このようなデータの間引きを行うコードを考えます。

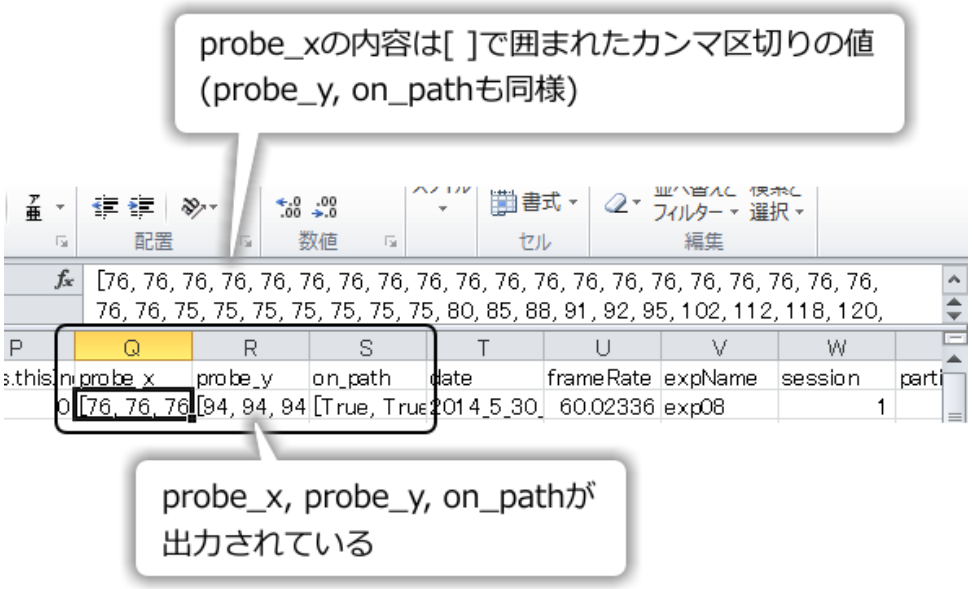


図 9.17　trial-by-trial 記録ファイルにおけるプローブ座標と判定結果の出力。Python のリストと同様に、[ ] で囲まれたカンマ区切りの値として出力されています。

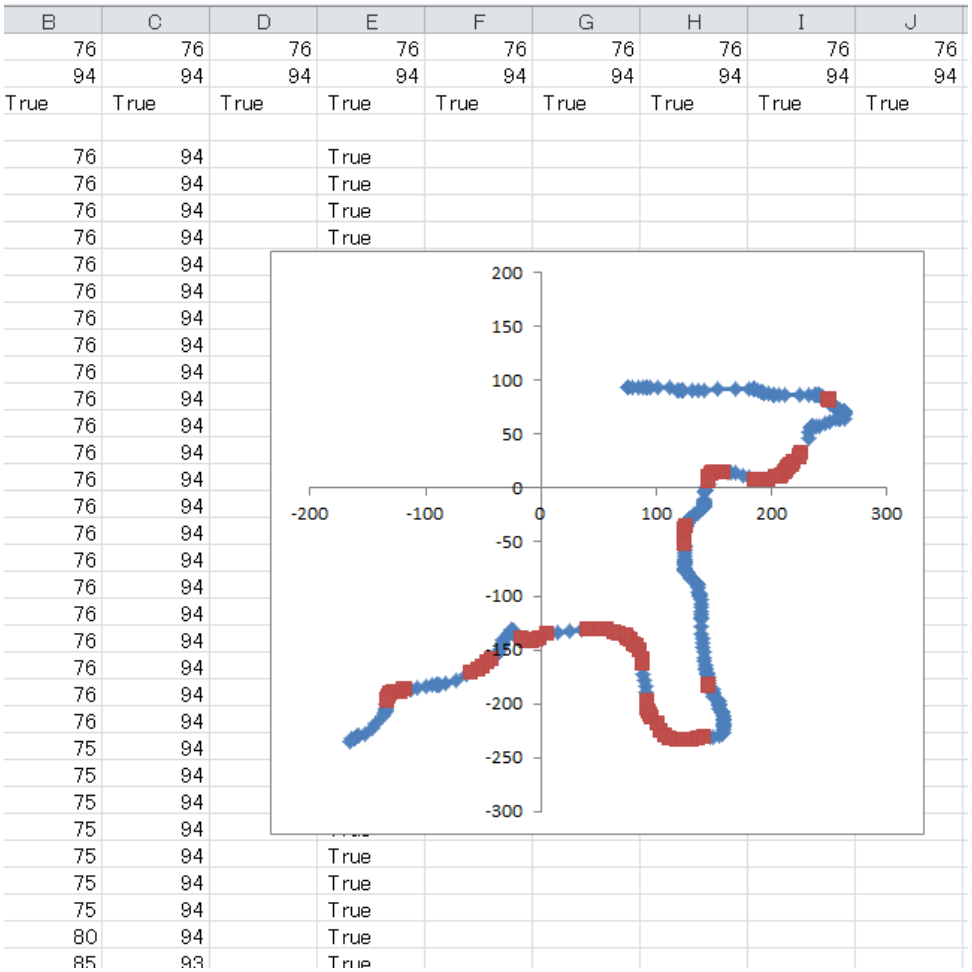


図 9.18　probe_x、probe_y、on_Path を Excel の「データの区切り位置」機能を用いて復元し、軌跡をグラフとしてプロットしたもの。軌跡の青い部分はパス上にプローブの中心が乗っていたことを、赤い部分は乗っていなかったことを示しています。

チェックリスト

- リストにデータを追加するメソッドである append( ) と extend( ) の違いを説明できる。
- 中身が空のリストを作成することができる。
- ルーチン内でのみ必要なデータを蓄積するリストを作成するコードはどの時点で初期化すべきか判断できる。

9.11 軌跡データを間引きしよう

軌跡データの間引きといってもいろいろな方法があるのでしょうが、もっとも単純な方法はサンプルを 1 つおきに保存すると言った具合に、一定間隔でサンプルを保存してその間のサンプルは捨ててしまうといったものでしょう。このように「順番に並んでいる (取得される) データを一定間隔で取り出して処理する」という作業には定番のコードがありますので、ぜひ覚えておきましょう。exp09.psyexp を開いて、trial ルーチンに配置してある Code コンポーネントの **[フレーム毎]** の最後を見てください。プローブの座標とプローブがパス上にあるかの判定結果のデータを append している 3 行のコードがあります。frameN % 6==0 という式が True になるときだけこの 3 行のコードを実行するように、以下のように if 文を書き足します。これだけの書き換えで、で 6 件につき 1 件のペースでデータを間引いて記録するようになりました。

```
if frameN % 6 == 0:
    probeX_list.append(px)
    probeY_list.append(py)
    onPath_list.append(onPath)
```

さて、なぜこれだけでデータを 6 件に 1 件に間引くことができるのでしょうか。ポイントは % 演算子にあります。第 5 章 で Python の算術演算子を紹介しましたが、その中に % 演算子が含まれていたのを覚えていますでしょうか。% 演算子は、x % y の形で使用して、x を y で割った時の余りが得られます。余りの事を「剰余」と呼ぶことから、% 演算子のことを「剰余演算子」と呼びます。frameN は 第 5 章 で紹介した Builder の内部変数で、ルーチンの実行が開始してから描画したフレーム数が格納されています。ということは、frameN % 6 という値は 6 フレーム毎に 0 になります (図 9.19)。この性質を利用して、frameN % 6==0 が True となる時だけに append を行えば、6 件につき 1 件のペースでデータを間引いて記録できるのです。

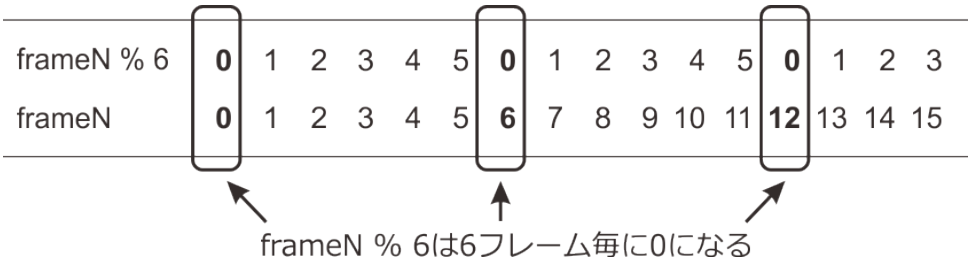


図 9.19　剰余演算子を利用したデータの間引き。

% 演算子のもう一つの重要な応用例を 図 9.20 に示します。この例では、数値がカレンダーのように左上から右下に向かって並べられています。このように、升目状に数値が並んでいるデータ構造を 2 次元配列と呼びます。2 次元配列のデータはさまざまなプログラミングで使用されますが、その際に「m 行 n 列の位置にあるデータは先頭から数えて何番目か？」とか、逆に「先頭から数えて i 番目のデータは何行目、何列目にあるか？」といった計算が求められることがよくあります。Python のインデックスが 0 から数え始めることに注

意すると、m 行 n 列の位置にあるデータは m ×列数 +n であることがわかります。インデックス i が与えられた時の行数は「i を列数で割って小数点以下を切り捨てたもの」、すなわち int(i / 列数) で計算できることがわかります。そして、列数を計算する時が % 演算子の出番です。i % 列数を計算すれば、インデックス i の要素が何列目にあるかわかります。この章の実験では使用しないテクニックですが、非常に有効ですので覚えておいてください。

図 9.20　2 次元配列を 1 次元に展開した時の要素のインデックスと行、列の相互変換。インデックスは左上から右方向へ順番に割り当てて、右端に達したら次の行の左端へ移るものとします。

% 演算子の話はこのくらいにしておいて、exp09.psyexp の作業に戻りましょう。もう先ほどの if 文を入力しただけでこの節の目的はもう達成できているのですが、このまま実行しても間引きの効果が大変わかりづらいです。そこで、プローブ座標などを保存したフレーム番号を frameN という列名で実験記録ファイルに出力するように改造し、ついでに実験情報ダイアログに Interval という項目を設けて何フレーム毎に保存するかを指定できるようにしましょう。これはすでに解説済みのテクニックの復習ですので、自信がある人はノーヒントで取り組んでみてください。うまく動作したら「チェックリスト」まで飛ばしていただいて構いません。

では、作業内容を順番に説明します。まず、フレーム番号を保存するためのリストが必要なので、trial ルーチンの Code コンポーネントの **[Routine 開始時]** に以下のコードを追加しましょう。

```
frameN_list = [ ]
```

続いて、**[フレーム毎]** の最後でリストにデータを append しているところの最後に frameN_list へ frameN を append するコードを追加しましょう。また、剰余演算の部分で実験情報ダイアログの Interval という項目から値を取得するようにしておきましょう。実験情報ダイアログから取得した値は文字列なので、数値に変換するために int( ) を使用しないといけない点に注意してください。

```
if frameN % int(expInfo['Interval']) == 0:
    probeX_list.append(px)
    probeY_list.append(py)
```

(次のページに続く)

(前のページからの続き)

```
    onPath_list.append(onPath)
    frameN_list.append(frameN)
```

以上のコードを入力したら、実験設定ダイアログを開いて、実験情報ダイアログに Interval という項目を追加しておきましょう。そして最後に、trial ルーチンの Code コンポーネントに戻って **[Routine 終了時]** の最後に frameN_list を実験記録ファイルに出力するコードを付け足しておきましょう。

```
trials.addData('frameN', frameN_list)
```

以上で実験は完成です。exp09.psyexp を保存して実行してみましょう。Interval の値をいろいろと変更して、実験記録ファイルの frameN の値の間隔が変化することを確認してください。

チェックリスト

- % 演算子を用いて N フレーム (N=2,3,4,…) に 1 回の頻度で処理を実行させることができる。
- 2 次元配列の要素のインデックスから、その要素が何行目、何列目にあるかを計算することができる。

9.12 ゴール地点でクリックして終了するようにしてみよう

この章も長くなりましたし、そろそろ練習問題にしたいのですが、その前にもう一つだけ作業をします。本当はこの作業を練習問題にしたいのですが、今後皆さんが他人が書いたコードを読まないといけなくなった時に知っておくと役立つポイントがありますので解説しておきます。

この節で行う作業は、「プローブがゴールに到着したときに自動的に試行を終了するのではなく、ゴールに到着したうえでマウスをの左ボタンをクリックしないといけないようにする」というものです。ここまでの作業では Mouse クラスの getPressed( ) の出番がなかったので、このメソッドを使う練習をしておくのが狙いです。

ここまでの作業で作成した exp09.psyexp の trial ルーチンの終了判定は、以下のようなコードで行われています。

```
if goalDisc.contains([px, py]):
    continueRoutine = False
```

この if 文の条件に「マウスの左ボタンが押されている」という条件を付けくわえれば、目的は達成できます。両方の条件を満たさないといけないので、and 演算子を使って以下のように書けるはずです。

```
if startDisc.contains([px, py]) and マウスの左ボタンが押されている:
    continueRoutine = False
```

表 9.2 の getPressed( ) メソッドを利用すれば、ボタンの状態を保持したリストが得られます。これを buttonStatus という変数に格納しておきましょう。このリストの最初の要素が左ボタンの状態に対応しているのですから、以下のように書けば左ボタンが押されていることを判定できます。

```
buttonStatus = mouseReady.getPressed( )
if startDisc.contains([px, py]) and buttonStatus[0]==1:
    continueRoutine = False
```

これで全く問題ないのですが、Python に慣れるためにちょっと頭の体操をしましょう。getPressed( ) の戻り値はリストです。リストから要素を取り出すには、リストの後ろに [ ] 演算子を添えればよいのでした。ですから、上記のコードにおける buttonStatus という変数は必要ではなくて、以下のように書くことができます。

```
if startDisc.contains([px, py]) and mouseReady.getPressed( )[0] ==1:
    continueRoutine = False
```

Python 以外のプログラミング言語に慣れている方の中には、関数を呼び出す ( ) の後ろに [ ] 演算子があるのを見て奇妙に思う方もおられるかも知れません。しかし、これは var[0] がリストの時に var[0][1] という具合に [ ] 演算子を連続して書くことができたのと同じことです。このような書き方は web 上のさまざまな Python のサンプルコードでしばしば見かけますので、しっかりと理解しておいてください。

この節で取り上げた「ある座標が刺激に含まれていて、かつマウスがクリックされている」という状態を判定するテクニックは、Builder で作った実験のスクリーン上にボタンを表示してマウスでクリックして選択させるユーザーインターフェースを実装したいときなどに有効なので、ぜひ覚えておきたいところです。この用途には 表 9.2 にも挙げた isPressedIn() が便利ですが、「カーソルが重なっただけで色を変化させる」といった凝った動作をさせる必要がある場合などには、contains( ) を使う方が柔軟に対応できます。isPressedIn(shape, buttons) の例については「 10.5: マウスで一時停止やスキップを行えるようにしよう (上級) 」をご覧ください。

チェックリスト

- 関数やメソッドの戻り値に直接 [ ] 演算子を適用した式を理解できる。
- Polygon コンポーネントで描画した多角形にマウスカーソルを重ねてクリックするとルーチンの終了や反応の記録などの処理を行うコードを書くことができる。

9.13 練習問題：反転方向切り替え機能とフィードバック機能を追加しよう

以上でこの章の解説は終わりです。この章の内容まで理解できていれば、Builder で相当複雑な実験を作成することができるはずです。この章の仕上げとして、以下の練習問題に取り組んでください。ここまで理解できた人であればノーヒントでできるものと期待します。

- 実験情報ダイアログに Direction という項目を追加してください。Direction の値に応じて以下のようにプローブの動き方を変更してください。
  - Direction が UD という文字列であれば exp09.psyexp と全く同一の動作。
  - Direction が LR という文字列であれば、プローブがマウスカーソルの動きと左右反対に動く。
  - Direction が UD でも LR でもなければ、プローブとマウスの動きが一致。
- プローブの中心座標がパス上にある時にはプローブの **[塗りつぶしの色]** が赤色、パス上にない時には **[塗りつぶしの色]** が黒色になるようにしてください。

9.14 この章のトピックス

9.14.1 Builder の内部変数 trialComponents に含まれるオブジェクトについて

本文中で述べたように、Builder では各ルーチンの処理を開始する前に、「ルーチン名 +Components」という名前の変数 (以下 trialComponents) が作成されます。ここには、ルーチン実行中に毎フレーム処理しなければいけないオブジェクトがリスト型のデータとして列挙されています。オブジェクトはそれぞれ対応するコンポーネントの機能を実現するためのクラスインスタンスです。「6.9.5:Builder のコンポーネントと PsychoPy のクラスの対応 」で述べたとおり、コンポーネントによって Grating のように一対一にクラスが対応している物と、Polygon のように対応するクラスが変化するものがあります。また、Code コンポーネントは直接 Python のコードを Builder に埋め込むという性質上、他のコンポーネントとは異なる方法で管理されており、trialComponents に列挙されません。

9.14.2 論理式の評価順序について

本文中に出てきた「c が'name' というデータ属性を持っていないか、path という文字列をデータ属性 name に含んでいない」という条件を検査する式について補足しておきます。この式は以下の通りで、「or の前後はこの順序である必要がある」と本文中で述べました。

```
not hasattr(c, 'name') or not 'path' in c.name
```

なぜこの順序でなければいけないのでしょうか。その理由は、Python が論理式を評価する順番にあります。Python では、論理演算子は算術演算子や比較演算子より優先順位が低く、論理演算子の中では not、and、or の順に評価されます。優先順序に差がない部分は左側から評価されます。上記の順序であれば、左から評価されるので not hasattr(c, 'name') が評価されます。もし not 'path' in c.name が or の左に書いてあれば、いきなり c のデータ属性 name の値を参照するので c がデータ属性 name を持っていなければエラーで停止してしまいます。

not hasattr(c, 'name') を左に書いても結局データ属性 name が無かったら not 'path' in c.name でエラーになるんじゃないの？と思われるかもしれませんが、Python には「論理式の評価途中で値が確定してしまったら、残りの部分の評価を行わない」という規則があります。どういう事かと言うと、A or B という式で A が True であれば、B が True であろうと False であろうと A or B の結果は True です。このような時に、Python はわざわざ時間をかけて B の評価を行わず、A or B の結果は True と判断します。

今回の例に当てはめて考えましょう。not hasattr(c, 'name') が True だったら、この時点で条件式は True になるので、not 'path' in c.name は評価しません。not hasattr(c, 'name') が False の場合は、式の真偽が確定しませんので not 'path' in c.name を評価します。この時、not hasattr(c, 'name') が False だったのですから、c は必ずデータ属性 name を持っています。従って、not 'path' in c.name で「データ属性 name が無い」というエラーが生じることはありません。

同様の理由で、A and B という式を評価する時に、A が False であれば B の評価は行われません。

なお、この if 文がわかりにくいという方は、以下のように二つの if 文に分けて書くこともできます。覚えやすい方を覚えていただければと思います。

```
if not hasattr(c, 'name')
    continue
```

(次のページに続く)

(前のページからの続き)

```
if not 'path' in c.name
    continue
```

9.14.3 リストとタプル

タプルについては、第 5 章 で「シーケンス型」というデータ型が初登場したときに名前だけは紹介していたのですが、その性質については全く解説しませんでした。それは 第 5 章 の時点ではリストとタプルの違いを説明することが難しかったからなのですが、本章で [ ] 演算子による要素の指定を学んだおかげでようやく準備が整いました。

タプルは data = (1, -7, 'psychoplogy') といった具合に、リストと同様に要素をカンマで並べて作成します。リストとの違いは、リストでは [ ] で要素を囲んだのに対してタプルでは ( ) で要素を囲む点です。作成したタプルは、リストと同様に [ ] 演算子を適用することによって要素にアクセスすることができます。先の例で data[2] とすれば'psychology' が得られますし、data[-3] とすれば 1 が得られます。機能的な意味でのリストとタプルの最大の違いは、リストは要素を追加したり変更したりできるのに対して、タプルはそのような変更ができないという点にあります。例えば、リストであれば

```
data = [1, -7, 'psychology']
data[1] = 5
```

とすれば、data は [1, 5, 'psychology'] となります。一方、タプルに対して同様の処理をしようとするとエラーとなってプログラムの実行が停止します。

```
data = (1, -7, 'psychology')
data[1] = 5    #エラーとなる
```

また、リストにおける append( ) や extend( ) といったメソッドもタプルには存在しません。

どう考えてもタプルは不便なだけのような気がするのですが、なぜタプルなどというデータ型が用意されているのでしょうか。それは、タプルの方がリストよりも効率的かつ高速に処理できるからです。なぜそうなるのかを説明するのは難しいのですが、製本されたノートとルーズリーフの違いのようなものを思うと少しイメージしやすいかもしれません。ルーズリーフは途中に新しいページを挿入したり、順番を入れ替えたりすることが容易にできますが、本当に将来挿入や入れ替えをする必要があるのなら、複数件のメモを一枚のルーズリーフに書くことはできません。ほんの数行だけのメモだけで一枚のルーズリーフを使ってしまい、大変効率が悪いです。後で挿入や入れ替えをする必要がないのなら、ルーズリーフを使わなくても通常のノートに隙間なくメモを書き込む方がいいでしょう。恐らく使用するページ数も少なく済み、ルーズリーフを用意するより安価でかさばらないはずです。これはあくまで例え話に過ぎませんが、変更できないようにすることで得られるメリットがあるから Python にはタプルというデータ型が用意されていると理解しておいてください。

9.14.4 [クリック可能な視覚刺激 $] の活用

Button コンポーネントは比較的新しく追加されたコンポーネントなのですが、Button コンポーネントが使えるようになる以前にクリックできるボタンを Builder の画面上に配置する目的で役に立った機能が Mouse コンポーネントの **[クリック可能な視覚刺激 $]** です。本文にも書いた通り、 **[ボタン押しで Routine を終了]** で「有効なクリック」を選択した際に、 **[クリック可能な視覚刺激 $]** に記述した視覚刺激上にマウスカーソルを置いてクリックしたときだけルーチンが終了します。例えば buttonYes, buttonNo という名前の Polygon コンポーネントを画面上に配置して、**[クリック可能な視覚刺激 $]** に buttonYes, buttonNo と名前をカンマ区切りで列挙すれば、これらの Polygon コンポーネント上にマウスカーソルがある状態でマウスをクリックした時にルーチンを終了させることができます。

Button コンポーネントでは **[Routine を終了]** 、 **[コールバック関数]** 、 **[クリック毎に 1 回実行]** を組み合わせることによって、「 8.3: ミューラー・リヤー錯視の実験をマウスで操作できるようにしよう 」のように刺激の調整のために何度も押せるボタンや、一度クリックしたらルーチンを終了するボタンなど、さまざまなボタンを実現できます。これに比べると、Mouse コンポーネントの **[クリック可能な視覚刺激 $]** は柔軟性に欠けると言わざるを得ません。とはいえ、 **[クリック可能な視覚刺激 $]** を使う方が楽なパターンがいくつかあります。

まず、Mouse コンポーネントの「データ」タブにある **[クリック時に保存するパラメータ $]** を使うと、クリックした刺激刺激の度のパラメータを実験記録ファイルに出力するか指定できます。初期値では name, とだけ書かれていますが、これはクリックされたオブジェクトの **[名前]** を出力することを意味しています。name は **[名前]** に対応するデータ属性の名称で、他に 表 9.4 に挙げたようなデータ属性を指定することができます。複数の属性を指摘するときは name, color といった具合にカンマ区切りで列挙してください。mouse_resp という名前の Mouse コンポーネントで stim という名前の視覚刺激を **[クリック可能な視覚刺激 $]** に指定し、name, pos を **[クリック時に保存するパラメータ $]** に指定した場合、実験記録ファイルに mouse_resp.clicked_name, mouse_resp.clicked_pos という見出しの列に値が出力されます。

表 9.4　**[クリック時に保存するパラメータ $]** で使える主なデータ属性

name	**[名前]** に対応
opacity	**[不透明度 $]** に対応
ori	**[回転角度 $]** に対応
pos	**[位置 [x, y] $]** に対応
size	**[サイズ [w, h] $]** に対応
color	**[前景色]** に対応
fillColor	**[塗りつぶしの色]** に対応 (Polygon コンポーネント)
lineColor	**[枠線の色]** に対応 (Polygon コンポーネント)

[クリック時に保存するパラメータ $] が便利な場面は、「位置や色、回転角度が変化している刺激が、ある状態に達したと判断した時に反応させる」といった実験を行う時です。Button コンポーネントでも同様のことはできますが、目的のパラメータを保存するコードを書く必要があります。また、タブレット PC などでタッチパネルを用いて実験する場合、小さな反応ボタンを押すよりも刺激そのものを押させた方が操作しやすいといったケースも考えられるでしょう。

他に Mouse コンポーネントの **[クリック可能な視覚刺激 $]** を使う方が有利な例としては、ボタンのデザインを細かく調整したい場合や、長方形ではない形状の領域を選択させたい場合などが挙げられます。Button コンポーネントのデザインはかなりシンプルですし、テキストの配置もそれほど調整ができません。自分で

Polygon コンポーネントと Text コンポーネントを組み合わせれば細かな調整ができますし、ボタンを画像ファイルとして用意して Image コンポーネントで描画すれば、非常に凝ったデザインのボタンにすることもできます。長方形ではない形状の領域というのは、例えば写真の中で特定のターゲットを探し出してクリックさせるといった課題が考えられます。図 9.21 は写真に写っている動物を探す課題を想定した例で、Polygon コンポーネントの **[形状]** を「カスタムポリゴン...」にしてターゲットの形に合わせた領域を設定しています。図 9.21 上では、動作確認用に Polygon コンポーネントを写真の上に赤い輪郭線で重ねて描画しています。実験本番では、 図 9.21 下のようにルーチンにおけるコンポーネントの順序を変更して写真が上に描画されるようにすれば、Polygon コンポーネントは参加者には見えません。しかし Mouse コンポーネントによるクリックは有効なので、ターゲットの領域がクリックされたことを検出できます。なお、コンポーネントの順番を入れ替える以外にも、本番時には Polygon コンポーネントの **[不透明度 $]** を 0 にして見えなくしてしまうという方法も使えますが、領域が複数個ある時には順番を入れ替える方が早く作業できるはずです。また、ルーチンペインをぱっと見ただけでターゲット領域が見える状態 (写真の上に描かれている) か隠れた状態 (写真の下に描かれている) か判別できる点でも優れていると思います。

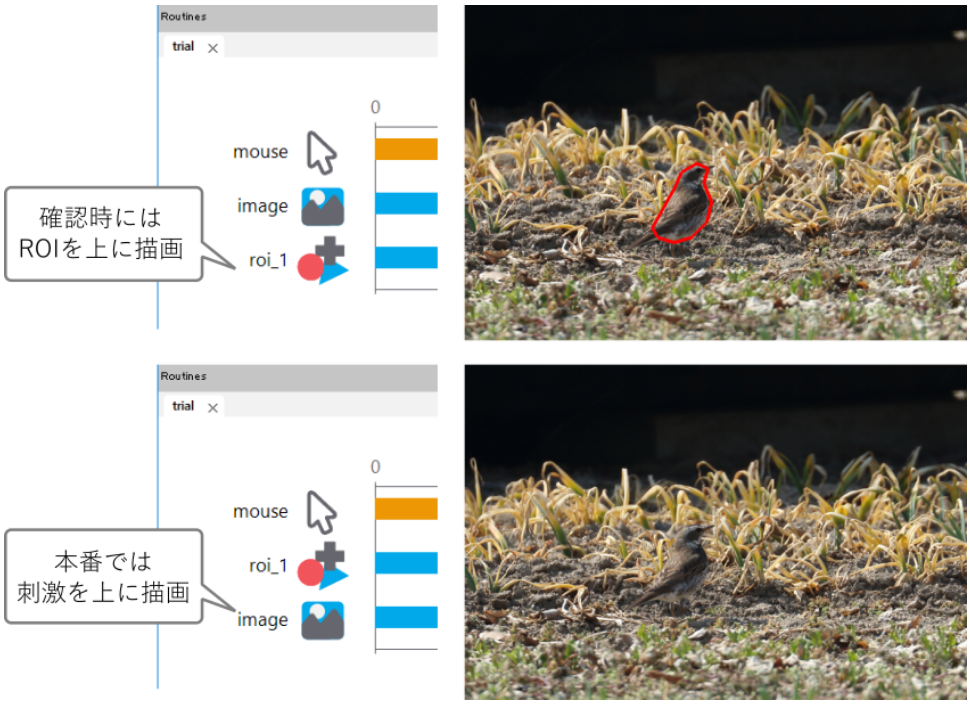


図 9.21　矩形ではない領域をクリックさせる課題を作成するときには、Polygon コンポーネントを **[クリック可能な視覚刺激 $]** に指定するとよい

図 9.21 のような写真を刺激に使う場合、ターゲットの正確な座標が

最後に、Mouse コンポーネントには **[新たなクリックのみ検出]** がある点も Button コンポーネントより優れた点です。「 8.5: 概念識別の実験をマウスで反応できるようにしよう 」では Slider コンポーネントに対するクリックが、次のルーチンに配置した Button コンポーネントに対するクリックと判定されてしまわないように、Slider コンポーネントと Button コンポーネントの位置を少し上下にずらしました。しかし Mouse コンポーネントと **[クリック可能な視覚刺激 $]** でボタン機能を実現する場合、 **[新たなクリックのみ検出]** をチェックしておけば新しいルーチンに入って一旦「ボタンを押していない」という状態になったあと、改めて「ボタンを押す」という操作をしないとクリックとして検出されないので、位置をずらすなどの対策をする必要がなくなります。

9.14.5 視覚刺激コンポーネントの [ドラッグ可能] について

「9.14.4:[クリック可能な視覚刺激 $] の活用」で Polygon コンポーネントを写真上のターゲットに合わせて配置していましたが、このような調整をしていると「もう少し下にずらしたいんだけど具体的に **[位置 [x, y] $]** をどのくらい変更したらいいのかな」というのがわからないことがよくあります。仕方がないので **[位置 [x, y] $]** を少しだけ変更して実験を実行してみて、その結果を見て改めて **[位置 [x, y] $]** を変更して…という作業を繰り返す人が多いと思うのですが、Polygon や Text などの視覚刺激コンポーネントが持つ **[ドラッグ可能]** を使うと少し作業が楽になるかもしれません。

[ドラッグ可能] は視覚刺激コンポーネントのプロパティ設定ダイアログの「レイアウト」タブにあり、この項目をチェックしていると、実験実行中にその刺激をマウスでドラッグして位置を変更できるようになります。Mouse コンポーネントを追加しなくても、実験設定ダイアログで **[マウスカーソルを表示]** をチェックしておけばこの機能を利用することができます。位置を調整したい刺激の **[名前]** が stim だとして、以下のように作業します。

- stim の **[ドラッグ可能]** をチェックする。
- Text コンポーネントをひとつ配置して、**[文字列]** に $str(stim.pos) と入力して「フレーム毎に更新」に設定する。位置やフォントの大きさ、色などは邪魔にならない＆見えなくならないよう適切に設定する。

この状態で実行すると、stim の位置をマウスでドラッグして変更することができ、しかも変更後の座標が画面上でリアルタイムに確認できます。複数の刺激の位置を同時に調整したい場合は、それらの刺激の **[ドラッグ可能]** をすべてチェックして、Text コンポーネントの **[文字列]** には

```
$str(stim1.pos)+'\n'+str(stim2.pos)+'\n'+str(stim3.pos)
```

のように'n' を間にはさみながら + 演算子で連結すればよいでしょう。'n' については「6.9.2: 改行文字を使った複数行の文字列の表現 (上級)」をご覧ください。

第 10 章

音声と動画を活用しよう

PsychoPy Builder では音声ファイルや動画ファイルを再生したり、マイクを使った音声の録音やカメラを使った動画の記録をしたりすることができます。動画機能についてはいろいろと技術的な難しさがあって、どのような動画ならば問題なく再生、録画できるか簡単には判断できないのですが、試してみることはそんなに難しくないのでぜひみなさん自身の PC で動かして確認してみてください。前章に引き続き、まとまった実験を作成するのではなくデモと解説を中心に進めます。

10.1 Sound コンポーネントで音声ファイルを再生しよう

Sound コンポーネントは Builder で音声刺激を扱うためのコンポーネントです。 図 10.1 に Sound コンポーネントのアイコンを示します。Sound コンポーネントのプロパティの内、これまでに紹介済みのコンポーネントと共通ではないのは「基本」タブの **[音]** と、新たに登場する「再生」タブ、「デバイス」タブです。

図 10.1　Sound コンポーネントのアイコン。

[音] には無圧縮 WAV 形式の音声ファイルを指定できるほか、A や Bfl (B ♭)、Csh (C#) のようにキーコードで音を指定することもできます。また、2000 という具合に正の数値を入力すると、その周波数の音が鳴ります。実行環境によっては WAV 以外に OGG などの音声ファイルを再生できますが、無圧縮 WAV ならほとんどの環境で再生できるので無難です。

「再生」タブの **[ボリューム $]** は 0.0 から 1.0 の範囲でボリュームを指定します。再生環境や音声ファイル形式によってはうまく機能しませんので、可能なら音声データ作成の時点でボリュームを調整していた方が良いでしょう。 ルーチンが終了した時点でまだ再生が続いていた場合、 **[Routine 終了時に停止]** がチェックされていれば音声ファイルの再生が途中で終了します。**[ハミング窓]** は音声のオンセットによるプチノイズを軽減するフィルタを使用するか否かを設定します。チェックしておいた方が無難ですが、およそ 1 ミリ秒ほど音の立ち上がりが遅れるので、極めて正確な時間制御が必要な場合はチェックをオフにした方が良い結果が得られ

るかもしれません。

「デバイス」タブでは、PC に複数のオーディオ出力デバイスがある場合にどのデバイスから音声を再生するのかを指定します。**[スピーカー]** には PsychoPy が検出しているオーディオデバイスがリストアップされていて、その中から再生に使用するデバイスをひとつ選択できるようになっています。Blutooth audio など、PsychoPy を起動した後に接続したオーディオデバイスは検出されない場合があるので、PsychoPy の起動前に接続しておくことをお勧めします。初期値の「既定値」を選択すると、実行時に自動的にデバイスが選択されます。「デバイス」タブにもうひとつある **[デバイスラベル]** は少々ややこしいのですが、Builder がオーディオ出力デバイスを管理する際に使用する名前をつけます。入力欄にマウスカーソルを合わせると表示されるヘルプにあるように、同一デバイスで再生する Sound コンポーネントには同じラベルを指定することが推奨されています。省略した場合は Sound コンポーネント名と同じラベルが自動的に設定されます (つまりすべての Sound コンポーネントは別のラベルとなる)。

音声ファイルを用いた実験を行う時にしばしば困るのが、「音声ファイルが再生されている間文字列が表示され、再生終了と共に消える」といった処理や、「音声ファイルの再生が終わったら文字列が表示されるようにしたいが、ルーチンは継続したいので **[Routine を終了]** は使いたくない」という場合です。使用する音声ファイルの再生時間がすべて同じであれば **[開始]** や **[終了]** の値を再生時間に合わせて設定すればいいのですが、ファイルによって再生時間が異なる場合は工夫が必要です。具体的には、Sound コンポーネントに対応する PsychoPy クラスが持っている status というデータ属性を利用します。音声または動画ファイルが再生されていなければ、status は NOT_STARTED という値が設定されています。再生中であれば PLAYING (または STARTED)、再生が終了していれば STOPPED (または FINISHED) です。これを利用すると、Code コンポーネントを用いて以下のように stim という名前の Sound コンポーネントの再生終了時にルーチンを強制終了させることができます。

```
if stim.status == FINISHED:
    continueRoutine = False
```

ルーチン全体を終了させるのではなく、特定のコンポーネントの描画を開始したり終了したりしたい場合は、そのコンポーネントの **[開始]** および **[終了]** で「条件式」 を使用すると便利です。 図 10.2 に音声ファイルの再生開始、終了に合わせてコンポーネントの開始、終了する例を示します。

最後にふたつ注意点を挙げておきます。まず、Sound コンポーネントによる音声の再生タイミングは結構「いいかげん」です。例えば「ぴぴっ」と音を短い音を 2 回鳴らしたいとします。再生時間 0.1 秒の Sound コンポーネントを 2 個配置して、それぞれの **[開始]** を 0.5 秒ずらしてやると「ぴぴっ」となるはずですが、実行する PC によっては 1 回しか音がならなかったり、全く音がならなかったりします。元々、PC のオーディオ機能はエラー音などを鳴らしたり、ひとつの音声ファイルを鳴らしたりするためのもので、短時間に複数の音声を正確に再生する機能は保証されていません。このような場合は、2 つの音を 1 つの音声ファイルにまとめるべきです。視覚－聴覚の相互作用の研究を考えておられる方は刺激を動画として作成するのもひとつの対策でしょう。なお、再生タイミングの問題は、実験設定ダイアログの「オーディオ」タブの **[オーディオライブラリ]** を ptb に設定して、 **[オーディオ遅延の優先度]** を厳しくすることでかなり改善されます。優先度の設定は一般的には 3. で十分ですが、4. にするとハードウェアがベストな設定に対応していない場合にエラーとなるので確実です (エラーとなる場合は実験用 PC を変えるか妥協するかを選ぶことになるでしょう)。

もうひとつの注意点は、異なるサンプリングレートで作成された音声ファイルをひとつの実験で使用しない方がよいということです。例えば、実験で音声ファイルを 10 個使用しているうちの 8 個が 44.1kHz、2 個が 48kHz でサンプリングされているといった状況です。実験の実行環境によっては音声ファイル読み込みの時

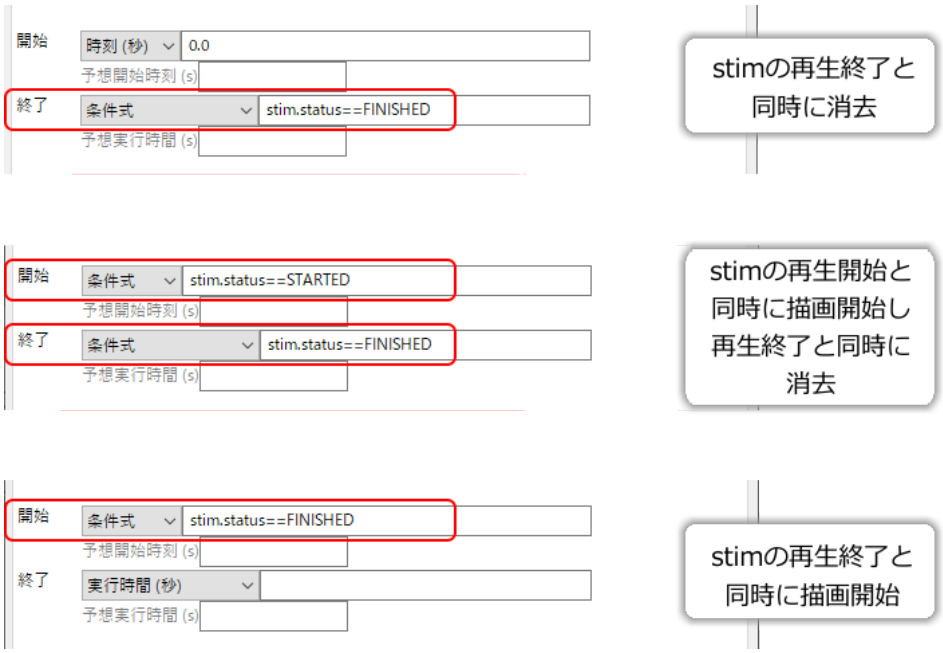


図 10.2　**[開始]** および **[終了]** に 「条件式」 を指定すると、条件式によってコンポーネントの開始、終了を制御できます

点でエラーが起こって実験が強制終了されてしまうことがあります。原因がわかりにくいエラーなので気をつけてください。

チェックリスト

- 無圧縮 WAV 形式の音声ファイルを再生できる。
- 指定された周波数の音を鳴らすことができる。
- 指定されたキーコードの音を鳴らすことができる。
- 音声のボリュームを指定できる。
- 音声ファイルの再生を指定された時刻に途中終了できる。
- 様々な再生時間の音声ファイルの再生開始、終了に合わせて他のコンポーネントを開始または終了させることができる。
- 短時間に複数の Sound コンポーネントを鳴らそうとした時に期待した結果が得られない理由を説明できる。
- 異なるサンプリングレートの音声ファイルをひとつの実験で混ぜて使用してはいけない理由を説明できる。

10.2 Movie コンポーネントで動画を再生しよう

Builder で動画を再生するには Movie コンポーネントを使用します (図 10.3)。Movie コンポーネントのプロパティの内、これまでに紹介済みのコンポーネントと共通ではないのは 「基本」 タブの **[動画ファイル]** と 「再生」 タブの **[バックエンド]**、**[音声無し]**、**[ループ再生]** です。

[バックエンド] は、これは PsychoPy が動画データを再生するときに使用するライブラリの指定です。

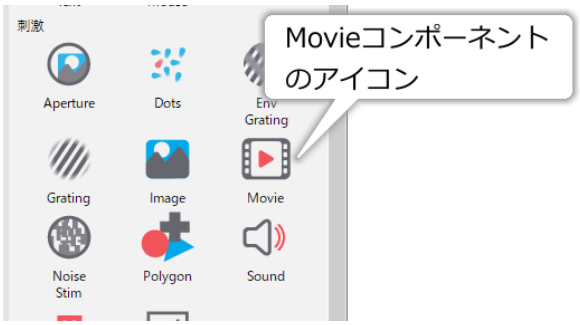


図 10.3　Movie コンポーネントのアイコン。

PsyhcoPy Builder のユーザーから見ると「Movie コンポーネント」が操作画面に見えていて実際に操作する対象であり、これを「フロントエンド」と呼びます。それに対して、Movie コンポーネントが動画再生のために内部で利用しているライブラリが「バックエンド」です。ffpyplayer、moviepy、opencv、vlc の 4 つが選択できます。バージョン 2024.2.5 現在、推奨されているのは ffpyplayer で、その他のバックエンドは主に過去のバージョンで作成された実験との互換性のために残されています。

[動画ファイル] には、再生する動画ファイル名を指定します。再生できる動画ファイルの形式はバックエンドによって決まりますが、moviepy(FFmpeg) なら一般的な形式はほとんど再生できると思います。動画ファイルのフォーマットはお勧めできる定番がないのですが、筆者は MP4 形式をよく使用しています。

「レイアウト」タブの **[サイズ [w, h] $]** を動画ファイルと異なる値に設定することによって、動画を縦横に拡大縮小して再生することができます。ただ、PsychoPy 上で拡大縮小できるからといって、実際に描画するサイズより解像度が高い動画ファイルを縮小表示するべきではありません。具体的にいうと、実験用に撮影した動画の解像度が 1920 × 1080 で、実験に使用する時の表示サイズが 480 × 270 であるならば、実験に使用する前に動画編集ソフトを用いて 480 × 270 に縮小すべきです。といいますのも、第一に解像度の高い動画ファイルは (よほど画質を落としていない限り) ファイルサイズが大きいので、その分 PC のリソースを消費してリソース不足を起こすかもしれません。第二に、縮小処理を PsychoPy に任せると縮小の品質が悪くて細い線や小さな文字などが鮮明に描画されないかも知れません。できる限り高品質な (恐らく時間がかかる) 方法で前もって縮小しておき、画質に問題がないことを確認しておくべきです。

「再生」タブの **[音声無し]** は文字通り音声なしで再生します。音声を再生せずに済むならその分負荷を軽減できます。実験の目的上音声が必要ないならチェックしておくとよいでしょう。 **[ループ再生]** は文字通り、これがチェックされていると動画をループ再生します。

チェックリスト

- 動画ファイルを拡大縮小して再生することができる。
- 動画ファイルを音声なしで再生することができる。

10.3 Movie コンポーネントを使ってみよう

それでは実際に Movie コンポーネントを使ってみましょう。なにか手ごろな動画ファイルを用意してください。動画ファイルはそのフォーマットと動画データの符号化方法が一対一対応していないので非常にややこしいのですが、mp4 形式の動画ファイルなら多くの場合 ffpyplayer バックエンドで再生できるはずです。動画ファイルが用意できたら作業を始めましょう。

- 実験設定ダイアログ
  - **[単位]** を pix にする (標準の height でないので注意)。
  - 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。
- trial ルーチン
  - Movie コンポーネントをひとつ配置して以下の通り設定する。
    - 「基本」タブの **[名前]** を `movie` にする (初期値)。 **[終了]** を空欄にする。
    - 「基本」タブの **[動画ファイル]** に使用する動画ファイルを指定する。相対パスが使える点などは Image コンポーネントと同様である。
    - 「レイアウト」タブの **[サイズ [w, h] $]** を動画の解像度に合わせる (例えば 640 × 480 の動画なら `(640, 480)`)。
  - Text コンポーネントをひとつ配置して以下の通りに設定する。
    - **[終了]** を「条件式」にして `movie.status == FINISHED` と入力する。
    - **[文字の高さ]** を 48 にする。
    - **[文字列]** に `$delay` と入力し、「フレーム毎に更新」にする。
  - Code コンポーネントをひとつ配置して、他のコンポーネントより先に実行されるよう一番上に並び替える。

作業が終了したら、Code コンポーネントの **[フレーム毎]** に以下のコードを入力する。

```
delay = 1000*(t - movie.getCurrentFrameNumber()/movie.getFPS())
```

完成したら実行してみましょう。動画のフォーマットが非対応でなければ、画面中央に動画が再生されてその上に数値が表示されます。getCurrentFrameNumber() は現在再生中の動画のフレーム番号、getFPS() は動画が毎秒何フレームかを返すメソッドで、現在の動画フレーム番号を毎秒フレーム数で割ることで動画のタイムスタンプが得られます。t からこの値を引くことによって、ルーチンの時計 t と動画のタイムスタンプがどれだけずれているかを計算できます。1000 倍しているのは単位を秒からミリ秒に変換するためです。この値は 0 になるのが理想ですが、どうしてもある程度の差は生じます。他のコンポーネントと連携させたいときに、この程度の時間のズレはあるというつもりで実験を作成するようにしてください。

数値の更新が速すぎて読めないという方は「9.11: 軌跡データを間引きしよう」の方法で 変数 `frameN` を使って以下のように間引きをするといいでしょう。

```
if frameN % 10 == 0:
    delay = 1000*(t - movie.getCurrentFrameNumber()/movie.getFPS())
```

再生の遅延や時間のズレは、同一の PC でも再生する動画の負荷によって変動します。いろいろなサイズの動画を用意して実行してみると良いでしょう。また、 **[サイズ [w, h] $]** に動画の解像度と異なる値を指定してみて、どの程度の影響が出るかを試してみてください。

t と getCurrentFrameTime() の差がどの程度だったか実験のたびに保存しておかないと心配だという方は、第 7 章 で解説した方法を使って差を変数に保持し、trial-by-trial 記録ファイルに出力するとよいでしょう。まず、trial ルーチンを繰り返すようにループを作成してください。 **[名前]** は `trails` 、 **[繰り返し回数]** は 1 でいいでしょう。ループを作成したら、の Code コンポーネントの **[Routine 開始時]** に以下のコードを追加します。

```
delay_list = []
```

続いて **[フレーム毎]** の最後に以下のコードを追加しましょう。間引きをした人は字下げに注意してください。

```
delay_list.append(delay)
```

最後に **[Routine 終了時]** に以下のコードを追加します。average() と std() は 表 5.2 で出てきた平均値と標準偏差を計算する関数です。

```
trials.addData('delay_mean', average(delay_list))
trials.addData('delay_std', std(delay_list))
```

これで trial-by-trial 記録ファイルに t と getCurrentFrameTime() の差の平均値と標準偏差が出力されるようになりました。なお、このコードを実際の実験で使用するときは、動画再生終了後直ちにルーチンを終了するようにしてください。動画再生終了後もルーチンを継続する必要がある場合は、動画再生中のみ差を append するようにすればいいでしょう。Code コンポーネントに慣れていないとちょっと難しいかもしれませんので、これも例を出しておきましょう。動画再生終了後もルーチンを 5 秒間継続して、計算した平均値と標準偏差を画面上に表示することにします。以下の通り作業してください。 **[文字列]** のところでは「 6.9.2: 改行文字を使った複数行の文字列の表現 (上級) 」で解説した方法を使用しています。

- trial ルーチン
  - Text コンポーネントをひとつ配置して、以下の通り設定する。
    - **[開始]** を「条件式」にして `movie.status == FINISHED` と入力する。
    - **[終了]** を「実行時間 (秒)」にして 5 と入力する。
    - **[文字の高さ]** を 48 にする。
    - **[文字列]** に `$' 平均: ' + str(average(delay_list)) + '\n 標準偏差:' + str(std(delay_list))` と入力し、「フレーム毎に更新」にする。

そして、Code コンポーネントの **[フレーム毎]** に入力しているコードに if 文を追加しましょう。2 行目と 3 行目が既に入力済みの部分です。

```
if movie.status == PLAYING:
    delay = 1000*(t - movie.getCurrentFrameNumber()/movie.getFPS())
    delay_list.append(delay)
else:
    delay = 0
```

動画が再生されている時にはデータ属性 status の値が PLAYING になっているので、if 文を使ってその時のみ

差を計算して append するようにしました。else の後の部分は動画が再生されていないときにも delay という変数が存在するのを保証するために設けています。else 以下を削除してしまうと実験開始直後に「delay という変数がない」というエラーメッセージが表示されて実験が止まります。 **[Routine 開始時]** に delay=0 と書いておくことでも回避できます。

チェックリスト

- Code コンポーネントを使って動画の再生中のフレーム時刻を得ることができる。
- Code コンポーネントを使って動画の再生中のみ実行する処理を記述することができる。

10.4 動画の再生位置を変更してみよう

実験に使用する動画は、できる限り実験の準備段階で実際に提示するとおりの状態にしておくことが理想ですが、実験によっては条件に応じて動画の特定の時点から再生したいということがあるかも知れません。そのような時に便利なのが動画のシークです。Builder で動画のシークを行うには Code コンポーネントを通じて動画オブジェクトの seek() メソッドを用います。これもデモを作成してみましょう。 **5 秒以上の長さがある動画** を用意してください。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
- trial ルーチン
  - Code コンポーネントをひとつ配置する。
  - Movie コンポーネントをひとつ配置して以下の通り設定する。
    - **[名前]** を `movie` にする (初期値)。
    - **[終了]** を空欄にする。
    - **[動画ファイル]** に使用する動画ファイルを指定する。5 秒以上の長さがあるものを使用してください。

そして、Code コンポーネントの **[Routine 開始時]** に以下のコードを入力してください。

```
movie.seek(5.0)
```

完成したら実行してみましょう。動画の冒頭を 5 秒飛ばしたところから再生が始まったはずです。ここで用いた seek() というメソッドは、動画の再生位置を引数で指定した値に変更します。引数の単位は秒です。

「動画の終了の 5 秒前」のように終了時刻を基準に指定したい場合は、動画の再生時間を保持している duration というデータ属性を利用して (単位は秒)、以下のように指定すればよいでしょう。

```
movie.seek(movie.duration - 5.0)
```

というわけで動画の頭出しができるようになりましたが、実験作成の時点で何秒飛ばすかがすでに決まっている場合はその分をカットした動画を作成した方が良いのは間違いありません。そうすると、実際にこのテクニックを使う状況は、実験中の参加者の反応によって飛ばす時間が変化する場合くらいかもしれません。

チェックリスト

- 動画を途中から再生開始することができる。
- 動画の再生開始位置を先頭から、または末尾からの秒数で指定することができる。

10.5 マウスで一時停止やスキップを行えるようにしよう (上級)

シークの話題があっさり終わってしまったので、ここで少し遊んでみましょう。画面中央下に 図 10.4 のようなボタンを表示し、に「5 秒戻る」、「5 秒進む」、「一時停止／再開」、「終了」の機能を割り振ってマウスで操作できるようにしてみます。「遊び」というのは実験としての実用性があまりないからですが、Builder でマウスベースのユーザーインターフェースを作る際の参考になるのではないかと思います。新たに実験を作成し、以下の通り作業してください。

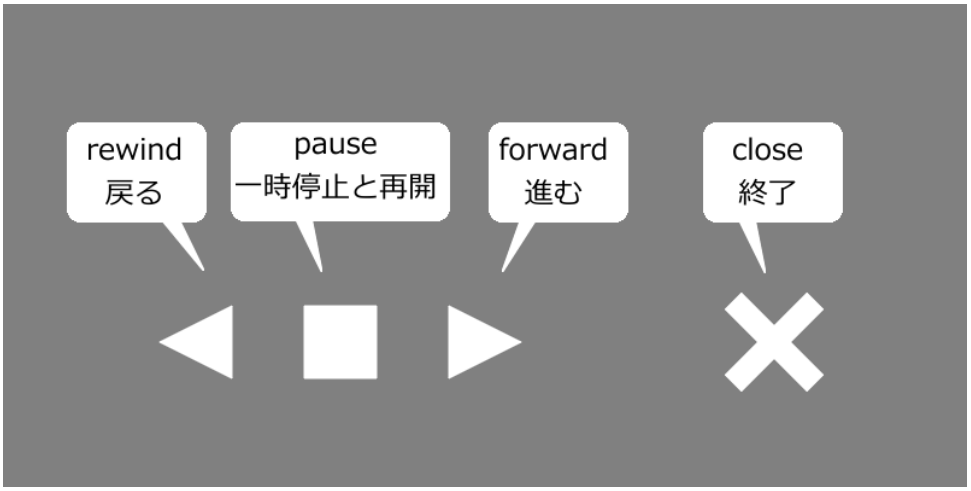


図 10.4　マウスでボタンをクリックして動画再生をコントールしてみます。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
  - 実験設定ダイアログの「スクリーン」タブの **[マウスカーソルを表示]** をチェックする。
  - 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。
- trial ルーチン
  - Movie コンポーネントをひとつ配置して以下の通り設定する。
    - 「基本」タブの **[名前]** を `movie` にする (初期値)。 **[終了]** を空欄にする。
    - 「基本」タブの **[動画ファイル]** に使用する動画ファイルを指定する。数十秒以上の長さがあるものが望ましい。
    - 「レイアウト」タブの **[サイズ [w, h] $]** を動画ファイルの縦横比に合わせて指定する (例えば 1920x1080 のように 16:9 の映像なら `(0.64,0.36)`、640x480 のように 4:3 の映像なら `(0.64,0.48)` など)。Y 軸の-0.4 の位置に Polygon コンポーネントでボタンを描くので、それらと重ならない程度の大きさにすることを勧める。

- Polygon コンポーネントをひとつ配置して以下の通り設定する。
  - 「基本」タブの **[名前]** を `rewind` にする。 **[終了]** を空欄にする。
  - 「基本」タブの **[形状]** を「三角形」にする。
  - 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.05, 0.05)` にする。
  - 「レイアウト」タブの **[位置 [x, y] $]** を `(-0.2, -0.4)` にする。
  - 「レイアウト」タブの **[回転角度 $]** を `-90` にする。
  - 「外観」タブの **[塗りつぶしの色]** を `white` にする (初期値)。

  ~~rewind~~ **をコピーして、** `forward` **という名前で貼り付けて以下の通り設定する。**
  - 「レイアウト」タブの **[位置 [x, y] $]** を `(0.0, -0.4)` にする。
  - 「レイアウト」タブの **[回転角度 $]** を `90` にする。
- さらにコンポーネントの貼り付けの操作をして (rewind がコピーされた状態になっている)、`pause` という名前で貼り付けて以下の通り設定する。
  - 「基本」タブの **[形状]** を「長方形」にする。
  - 「レイアウト」タブの **[位置 [x, y] $]** を `(-0.1, -0.4)` にする。
  - 「レイアウト」タブの **[回転角度 $]** を `0` にする。
- さらにコンポーネントの貼り付けの操作をして (rewind がコピーされた状態になっている)、`close` という名前で貼り付けて以下の通り設定する。
  - 「基本」タブの **[形状]** を「十字」にする。
  - 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.07, 0.07)` にする。
  - 「レイアウト」タブの **[位置 [x, y] $]** を `(0.2, -0.4)` にする。
  - 「レイアウト」タブの **[回転角度 $]** を `45` にする。
- Mouse コンポーネントをひとつ配置して以下の通り設定する。
  - 「基本」タブの **[名前]** を mouse に (初期値)、 **[終了]** を空欄にする。
  - 「基本」タブの **[ボタン押しで Routine を終了]** を「有効なクリック」にし、 **[クリック可能な視覚刺激]** に `close` と入力する。
  - 「データ」タブの **[マウスの状態を保存]** を「なし」にする。
- Code コンポーネントをひとつ配置する。

ここまで作業が終わったら、Code コンポーネントにコードを入力します。未解説のメソッドとデータ属性をいくつか使いますのが、それらは本章の最後に表としてまとめておきます。

まず **[Routine 開始時]** に以下のコードを入力します。この変数 step はボタンを押したときに進む/戻る時間を保持しています。この値を変更すると進む/戻る時間が変わります。

```
step = 5.0
```

続いて **[フレーム毎]** に以下のコードを入力します。ここで pause() と play() というメソッドが出てきますが、これはそれぞれ動画再生の一時停止、再開を行うものです。

```
ct = movie.getCurrentFrameNumber()/movie.getFPS()

if mouse.isPressedIn(rewind):
    movie.pause()
    movie.seek(max(0,ct-step))
    movie.play()
elif mouse.isPressedIn(forward):
    movie.pause()
    movie.seek(min(movie.duration,ct+step))
    movie.play()
elif mouse.isPressedIn(pause):
     if movie.isPaused:
        movie.play()
        pause.fillColor = 'white'
    else:
        movie.pause()
        pause.fillColor = 'red'
```

簡単に処理内容を解説しておくと、まず getCurrentFrameNumber() と getFPS() を使って現在の再生位置を得て変数 ct に代入しておきます。続いて 第 9 章 で少し触れたマウスオブジェクトのメソッドである getPressedIn を使って rewind、forward、pause の各 Polygon オブジェクト上でマウスボタンが押されたかを判定していきます。isPressedIn の引数に指定されたオブジェクト内にマウスカーソルがあってボタンが押されていたら True が返されるので、if 文で処理を分岐します。

rewind と forward でマウスのボタンが押されていた時の処理では、先ほどの変数 ct に step を加算、もしくは減算して seek() を実行しています。これで「現在再生中の位置から step 秒戻る、または進む」が実現できます。ただ、再生中に seek() を行うと音声は進んでいるのに映像が止まったままになったりすることがあるので、seek() の前に pause() でいったん再生を止め、そして seek() 後に play() で再開しています。pause() と play() をコメントアウトして比較してみるとよいでしょう。

pause でマウスのボタンが押されていた時は、動画の再生状態に応じて処理を分岐します。再生中であればデータ属性 status の値が PLAYING、一時停止中であれば PAUSED となるのですが、筆者の動作環境 (PsychoPy 2024.2.5, Windows 11, ffpyplayer) では一時停止しても status が変化しないという問題が生じたので、一時停止中であれば True、そうでなければ False を保持している isPaused というデータ属性を使用しています。if 文で分岐して isPaused が真なら一時停止中なので再開のために play() を実行し、偽ならば再生中なので pause() を実行して一時停止します。ついでに、一時停止中かわかりやすいように pause ボタンの色を一時停止中には赤色に変更しています。一時停止を解除するときには白色に戻します。

ここには close を押したときの終了処理が書かれていませんが、それは Mouse コンポーネントの **[クリック可能な視覚刺激]** に close を設定することで実現されているので、Code コンポーネントを使う必要がありません。

ここまで作業ができたら、一度保存して実行してみましょう。画面中央に動画が再生され、画面中央下に 図 10.4 のように表示されますので、クリックして動作を確認してください。PsychoPy にとって動画の一時停止、再生再開は重い処理なので、一般的なスペックの PC だと各ボタンをクリックしてから効果が表れるまで一呼吸待たされますのでそのつもりでいてください。なお、動画の再生が終了してもデモは終了しませんので、close を押して終了してください。

いかがだったでしょうか。「戻る」と「進む」、「終了」は (やや待たされるかもしれませんが) 特に問題なく動作したと思います。でも、「一時停止」はボタンが高速に赤と白に点滅して、一瞬だけ止まってすぐ動きだしたり、一時停止した状態から再生を開始できなかったりしなかったでしょうか？ なぜそうなるかというと、isPressedIn() は「現在マウスのボタンが押されているか」を返すメソッドだからです。PC の画面が 60fps で描かれている場合、**[フレーム毎]** の処理は 1/60 秒に 1 回実行されます。今回のデモでは pause() や play() が少々重い処理なので 1/60 秒内で終わらないこともありますが、それでも人がマウスのボタンを「カチッ」とクリックした間に数フレームは過ぎてしまいます。そうすると、現在のコードでは 1 回「カチッ」とボタンを押しただけで pause() と play() が繰り返し実行されてしまいます。なので、クリックした後にうまく一時停止する場合と再生されて続けてしまう場合があるのです。

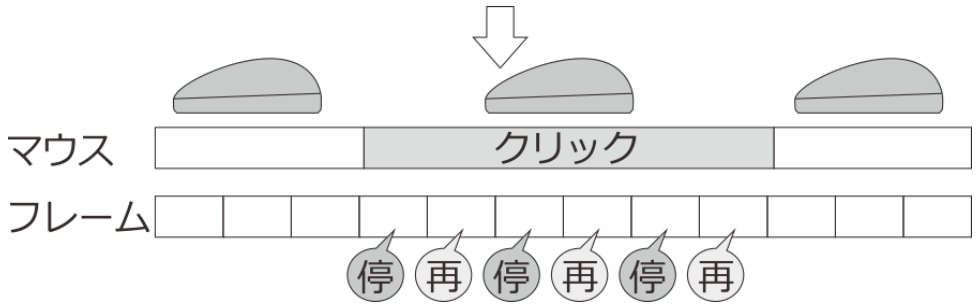


図 10.5　クリックが数フレームにわたった場合に対応する必要がある

ではどうすればいいかというと、いくつか方法があります。ここでは mouse.isPressedIn(pause) が True になったときの時刻を保持しておいて、一定時間が経過するまでは mouse.isPressedIn(pause) が True であっても無視するという方法を取り上げましょう。まず Code コンポーネントの **[Routine 開始時]** に以下の行を追加します。

```
wait = 0
```

続いて **[フレーム毎]** に以下のように追加します。追加した行の最後に目印としてコメントを入れています。皆さんが試してみるときはこれらのコメントを入力する必要はありません。

```
ct = movie.getCurrentFrameNumber()/movie.getFPS()
if wait > 0:    # (1)
    wait -= 1   # (1)

if mouse.isPressedIn(rewind):
    movie.pause()
    movie.seek(max(0,ct-step))
    movie.play()
elif mouse.isPressedIn(forward):
```

(次のページに続く)

(前のページからの続き)

```
    movie.pause()
    movie.seek(min(movie.duration,ct+step))
    movie.play()
elif mouse.isPressedIn(pause) and wait <= 0:  # (2)
    wait = 30                                 # (3)
     if movie.isPaused:
        movie.play()
        pause.fillColor = 'white'
    else:
        movie.pause()
        pause.fillColor = 'red'
```

この例では、wait という変数を用意して、一時停止と再開の処理を行う前に wait=30 をセットしています (コメント (3))。その後、フレーム毎に wait の値が 0 より大きければ 1 を減算していきます (コメント (1))。そしてコメント (2) のところでボタン押しを判定する際に wait<=0 を条件として追加することで、wait の値が 0 より大きい間はボタンを押しても一時停止/再開が行われないようにしています。この wait という変数はルーチン開始時に存在していないといけないので、**[Routine 開始時]** に wait=0 としているわけです。実行してみて、一時停止が機能することを確認してください。

なお、この方法ではボタンが押されてからの時間をフレーム数でカウントしていることになります。コメント (3) で wait=30 としているので 30 フレーム、60fps で刺激提示しているのなら待ち時間は 0.5 秒です。長すぎる場合は数値を減らしてみましょう。また、この例では戻る、進む処理について対策していませんが、これらについても対策を追加するにはいくつか書き方があるので、考えてみると良い練習になるでしょう。

他の方法としては、直前のフレームのボタンの状態を mouse.getPressed() で保持しておいて、「直前のフレームでボタンが押されていなくて、今のフレームでは押されている」という時だけ処理するという方法が考えられます。この場合、もうボタンが押されていることは確実なので isGetPressedIn() ではなく contains() でクリックされた Polygon オブジェクトを判別できます。

フレーム数を数える方法は、操作している人がボタンをずっと長押しすれば処理が繰り返されます。それに対して、直前のフレームの状態を保持する方法では、長押ししても処理されるのは 1 回のみです。どちらがの方が望ましいかは状況によるでしょうから、どちらの方法もマスターしておくのが理想です。

以上、Movie コンポーネントと Mouse コンポーネントで「遊んで」みましたがいかがだったでしょうか。最後に、この章で使用した Movie オブジェクトのデータ属性とメソッドを 表 10.1 にまとめておきます。

表 10.1　Movie オブジェクトの主なデータ属性とメソッド

status	現在の状態をあらわす。再生前なら NOT_STARTED、再生中なら PLAYING (STARTED)、一時停止中なら PAUSED、再生終了なら STOPPED (FINISHED)。ただし一時停止亭も
duration	動画の長さを保持している。単位は秒。
isPaused	動画再生中に一時停止していれば True、そうでなければ False を保持している。
getCurrentFrameNumber()	動画の現在のフレームを返す。
getFPS()	動画の 1 秒あたりのフレーム数を返す。
play()	動画の再生を開始する。一時停止している場合は再生を再開する。
pause()	動画の再生を一時停止する。
seek()	動画の再生位置を変更する。単位は秒。

チェックリスト

- 動画の再生を一時停止、再開できる。
- Code コンポーネントで動画が一時停止中であることを条件に処理を分岐できる。
- マウスでオブジェクトを「クリック」した際にボタンが押されている期間が複数フレームにわたる場合を考慮したコードを記述できる。

10.6 Microphone コンポーネントで録音してみよう

音声と動画の再生の解説が終わったので、続いて録音、録画の解説に進みましょう。まず音声の録音を行う Microphone コンポーネントを取り上げます。Microphone コンポーネントは「反応」カテゴリにあります (図 10.6)

図 10.6　Microphone コンポーネントのアイコン

Microphone コンポーネントの「基本」タブは他のコンポーネントと同様なので解説は必要ないと思います。「デバイス」タブは Sound コンポーネントと同様、録音に使用するデバイスの設定を行います。**[デバイス]** は、PsychoPy によって検出された録音デバイスの中から使用するデバイスを指定します。初期値は default で、

OS で設定されている標準の録音デバイスを使用します。Buletooth デバイスなどを PsychoPy 起動後に接続しても検出されない場合がありますので、PsychoPy 起動前に接続しておいてください。**[最大録音データサイズ (kb)]** は、録音が長時間に及んで巨大な音声ファイルが出力されてしまうことを防止するためのものです。この上限を超えた後は自動的に録音が停止しますが、対応する Microphone オブジェクトの isRecBufferFull() というメソッドで上限に達したかどうかを調べることができます (上限に達したら True が返される)。必要に応じて変更してください。

「音声文字変換」タブは録音した音声から自動的に発話を書き起こす機能の設定を行います。 **[音声の文字変換]** をチェックするとこの機能が有効になり、他の項目の設定が可能になります。まず **[音声文字変換バックエンド]** で使用するライブラリを指定します。初期状態は「なし」で書き起こし機能は使えません。Google は Google Cloud API を使用します。Google Cloud API はインターネット接続が必要で、Google Cloud API のキーを持っている必要がありますが、日本語にも対応しているという強みがあります。Google Cloud API のキーは JSON ファイルに保存して PsychoPy の設定ダイアログの「一般」タブにある **[GoogleCloud アプリケーションキー]** に指定してください。

「データ」タブの **[出力ファイル形式]** は音声ファイルの形式を指定します。かなりたくさんの形式がリストアップされるのでどれにしたらよいか迷うかもしれませんが、問題 (後述) が生じないなら default のままでよいでしょう。 **[発話開始/終了時刻の記録]** をチェックすると、音量が基準以上/以下になった時刻を記録します。 **[無音期間のトリム]** をチェックすると、音量が基準以下の部分をファイルに出力しません。

録音が成功するとデータファイルの保存フォルダに「データファイル名に_mic_recorded とついたフォルダ (例えば foo_expn_2022-09-01-_20h00.00.000.csv というデータファイル名なら foo_expn_2022-09-01-_20h00.00.000_mic_recorded)」が作成され、その中に音声ファイルが出力されます。ループで繰り返し実行すると、繰り返し毎にファイルが作成されます。

以上で主要なプロパティは解説したので、動作確認してみましょう。新たに実験を作成して、以下のように作業してください。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
  - 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。
- trial ルーチン
  - Keyboard コンポーネントをひとつ配置して、以下の通り設定する。
    - 「基本」タブの **[終了]** を空欄にし、**[Routine を終了]** にチェックが入っていることを確認する。
  - Text コンポーネントをひとつ配置して、以下の通り設定する。
    - 「基本」タブの **[終了]** を空欄にする。 **[文字列]** に `$int(t)` と入力し、「フレーム毎に更新」にする。
  - Microphone コンポーネントをひとつ配置して以下の通り設定する。
    - 「基本」タブの **[終了]** を空欄にする。
    - 「デバイス」タブの **[デバイス]** が default になっていることを確認する。実行してうまく動作しなかった場合はここを変更してみる。

＊「音声文字変換」タブの **[音声の文字変換]** がチェックされていないことを確認する。

できたら実行してみましょう。画面上で時間のカウントアップが始まったら適当に音を鳴らして、キーボードのスペースキーなどを押して実験を終了してください。data フォルダを開いて音声ファイルを格納したフォルダができていること、中の音声ファイルを再生して音が録音できていることを確認しましょう。

うまく動作しない場合、まずシステムのオーディオ設定で録音可能なデバイスが確かに存在すること、アプリケーションから使用できるように設定されていることを確認してください。セキュリティの設定が厳しい場合、録音デバイスが存在しても任意のプログラムから使用できないようになっている可能性があります。マイクを使用する他のアプリを起動して動作することも確認するとよいでしょう。

チェックリスト

- Microphone コンポーネントで録音を行うことができる。

10.7 Camera コンポーネントで動画撮影してみよう

録音に続いて動画撮影を取り上げます。使用するのはコンポーネントペインの「反応」カテゴリにある Camera コンポーネントです (図 10.7)。

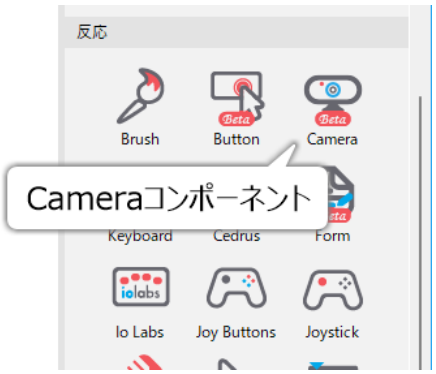


図 10.7　Camera コンポーネントのアイコン

Camera コンポーネントも Sound コンポーネントや Microphone コンポーネント同様「デバイス」タブがあり、ここで映像に関する主要な設定を行います。**[バックエンド]** は Movie デバイス同様、使用するバックエンドライブラリを指定します。**[ビデオデバイス]** には PsychoPy によって検出されたカメラデバイスがリストアップされていて、撮影に使用するカメラをここで指定します。**[解像度]**、**[フレームレート]** は文字通り録画される映像の解像度とフレームレートを指定します。残念ながら、これらのパラメータで選択できるすべての組み合わせが利用可能というわけではありません。各自の環境でどの組み合わせなら利用できるか試行錯誤する必要があります。

「オーディオ」タブには録音に関する設定項目が並んでいて、Microphone コンポーネントと基本的に同じですが **[チャネル]** や **[サンプリングレート (Hz)]** などの独自項目もあります。これも映像のパラメータと同様、選択できるすべての組み合わせが利用可能とは限りませんので、各自の環境でどの組み合わせが有効か試行錯誤してください。

「データ」タブには **[ファイルへ保存]** という項目があります。これをチェックしていたら動画がファイルに保存されます。Microphone デバイスと同様、データファイルの保存フォルダに「データファイル名に _cam_recorded をつけた名前のフォルダ」が作成され、その中に保存されます。Camera コンポーネント独自

の項目は以上なので、さっそくサンプルを作ってみましょう。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
  - 安定して使用できるパラメータの組み合わせが見つかるまでは Pilot モードで実行するか **[フルスクリーンウィンドウ]** のチェックを外しておく。
  - 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。
- trial ルーチン
  - Keyboard コンポーネントをひとつ配置して、以下の通り設定する。
    - 「基本」タブの **[終了]** を空欄であること、 **[Routine を終了]** にチェックが入っていることを確認する。
  - Text コンポーネントをひとつ配置して、以下の通り設定する。
    - 「基本」タブの **[終了]** を空欄にする。 **[文字列]** に `$int(t)` と入力し、「フレーム毎に更新」にする。
  - Camera コンポーネントをひとつ配置して以下の通り設定する。
    - 「基本」タブの **[終了]** を空欄にする (初期値)。
    - 「デバイス」タブの **[ビデオデバイス]** 、 **[解像度]** 、 **[フレームレート]** や「オーディオ」タブの **[マイク]** 、 **[チャネル]** 、 **[サンプリングレート (Hz)]** を自分の環境に合わせて設定する。どれにしたらいいかわからない場合、320x240 (QVGA) か 640x480 (VGA) などの多くのデバイスがサポートする解像度で 30fps から試してみるとよい。

完成したら実行してみましょう。実験が始まった直後に終了してしまって、Runner の標準出力に Specified camera format is not supported. といったエラーメッセージが出る場合は **[ビデオデバイス]** 等の設定を変更して実行する作業を繰り返して、安定的に動作する選択肢を探してください。エラーで停止した時の復帰をしやすくするため、安定的に動作する選択肢が見つかるまでは **[フルスクリーンウィンドウ]** のチェックを外しておくのがお勧めです。

無事に実験が動作したら、カメラの前で手を叩いて「ぱん！」と鳴らすなど、「動きと同期して音が鳴る」様子を記録してからキーボードのスペースキーなどを押して実験を終了し、動画ファイルが保存されていることを確認しましょう。動画ファイルが出力されていない場合や、出力されていても通常の動画プレイヤーで再生できない場合は **[ビデオデバイス]** 等の設定が適切ではないので、面倒ですが選択肢探しの作業に戻ってください。

無事に動画が保存されていて再生できた場合は、映像と音のずれ具合を確認してください。ずれの大きさはカメラや PC の性能に依存しますが、筆者がこの原稿の執筆時に使用した環境 (Windows11 x64/PsychoPy2022.2.4/LogiCool C922) では 0.8 秒ほど映像が遅れます (音が先に鳴る)。皆さんの PC でどの程度ずれるかはわかりませんが、いずれにしても時間と映像の精確な同期が必要な用途には向いていないということです。Camera コンポーネントを実験に使用する場合は、このずれのことをよく頭においておいてください。

さて、時間的なずれにはもうひとつ、画面上に提示した刺激と動画のタイミングのずれもあります。これを確認するために、カメラを PC 画面に向けて実験を実行し、画面上に表示されているカウントアップの数字を録画してみましょう。理想的には録画が始まると同時に画面上では 0 が表示されていて、ちょうど 1 秒再生したところで数字が 1 になるはずです。コマ送りできるプレイヤーを使っている場合は、数字が 1 増える瞬間のコマから 1 フレームずつコマ送りして、録画時に設定した fps のコマ数 (例えば 30fps なら 30 コマ) だけ進めたタイミングで数字が増えるかどうかも確認しましょう。筆者の環境の場合、動画の最初から数えて 0 が表示されるまで約 23 コマの遅延がありました。30fps で 23 コマの遅延ということは 0.76 秒の遅延ですから、先ほど手を叩く動画で音に対する映像の遅延とほぼ一致しています。数字が増えるタイミングは録画時の fps(30) と一致していましたが、実行後に Runner に表示されている出力をよく見ると real-time buffer too full or near too full! frame dropped!といった警告が度々出力されていたので、時々フレーム落ちしていたものと思われます。皆さんの環境ではどのような結果になるかわかりませんが、本番の実験に使う前にこういった「ずれ」をしっかり検討しておくことをお勧めします。

続いて、もうひとつサンプルを紹介しましょう。次のサンプルでは動画保存をおこなうのではなく、画面上にカメラの映像をリアルタイムに表示してみます。以下のように作業してください。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
  - 安定して使用できるパラメータの組み合わせが見つかるまでは Pilot モードで実行するか **[フルスクリーンウィンドウ]** のチェックを外しておく。
  - 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。
- trial ルーチン
  - Camera コンポーネントをひとつ配置して以下の通り設定する。
    - 「基本」タブの **[名前]** が `cam` であることを確認する。 **[終了]** を `10` にする。
    - 「デバイス」タブと「オーディオ」タブの各項目を自分の環境に合わせて設定する。
    - 「データ」タブの **[ファイルへ保存]** のチェックを外す。
  - Code コンポーネントをひとつ配置する。
  - Image コンポーネントをひとつ配置して以下の通り設定する。
    - 「基本」タブの **[終了]** を `10` にする。 **[画像]** に `$frame_img` と入力し、「フレーム毎に更新」に設定する。
    - 「レイアウト」タブの **[サイズ [w, h] $]** に **[ビデオデバイス]** で設定したカメラ映像の縦横比に合わせて適当な値を入力する (例えば 1920x1080 のように 16:9 の映像なら `(0.64,0.36)`、640x480 のように 4:3 の映像なら `(0.64,0.48)` など)。
    - 「レイアウト」タブの **[垂直に反転]** にチェックを入れる。

Code コンポーネントが Image コンポーネントより先に実行されるように並んでいる (つまりルーチンペイン上で Code コンポーネントの方が上にある) ことを確認したうえで、Code コンポーネントの「フレーム毎」の Python のコード欄に以下のように入力する。

```
frame = cam.getVideoFrame()
if frame is not None and frame.colorData is not None:
    frame_img = frame.colorData.astype(np.float32).reshape(
        (frame.size[1],frame.size[0],3))
    frame_img /= 256
else:
    frame_img = np.ones((16,16,3),dtype=np.float32)
```

入力したら実行してみましょう。うまくいけば 10 秒間カメラの映像が画面上に描かれるはずです。動画が保存されない点も確認しておいてください。Camera コンポーネントと Image コンポーネントの使い方には特に難しいところはないでしょうから、ここでは Code コンポーネントのコードを集中的に解説します。

1 行目の cam.getVideFrame() というのは cam、つまり Camera コンポーネントによって作成された Camera オブジェクトの getVideFrame() というメソッドの呼び出しです。このメソッドは呼び出された時点で最新のビデオフレームを MovieFrame オブジェクトとして返します。MovieFrame オブジェクトは単なる画像データではなくタイムコードなど動画のさまざまな情報を含んでいて、画像データは colorData というデータ属性に格納されています。ただ厄介なことに、これはそのまま Image コンポーネントで表示できるような PsychoPy の画像データではなく、画像の各ピクセルの RGB 値をべたっと 1 次元に並べただけのものです (1920 × 1080 の解像度なら 1920 × 1080 × 3 = 6,220,800 個)。Image コンポーネントで使用するには、縦×横× 3 の 3 次元のデータで、なおかつ値が 0.0 から 1.0 の小数でなければいけません。

そこで今回のコードでは、まず画像データが得られていることを確認して (2 行目の if 文)、3 行目で astype() による小数型への変換と reshape() によるデータの 3 次元化を一気に行っています。続く 5 行目で RGB 値を 0.0 から 1.0 に変換していますが、ここでは変換前の RGB の各チャネルの数値が 0 から 255 の整数であることを前提にしています。一般的な web カメラならこれで問題ないはずです。7 行目は MovieFrame オブジェクトが画像を含んでいなかった場合のために 16 × 16 の解像度の真っ白の画像を作って代入しています。どうせ Image コンポーネントによって拡大されるのですから低解像度で十分です。これで変数 frame_img に Image コンポーネントで表示する画像データが作成されました。あとは Image コンポーネントで表示するだけです。

ひとつ注意が必要なのは Image コンポーネントの **[垂直に反転]** です。すべてのカメラでそうなるかわかりませんが、筆者が試した範囲では MovieFrame の画像データは PsychoPy の Imagae コンポーネントの画像データと比べると上下方向に反転しています。そこで正しい向きに表示するためにはデータ自体をさらに変換するか、この例のように **[垂直に反転]** をチェックする必要があります。もし鏡像のようにしたいのであれば、**[水平に反転]** もチェックすると良いでしょう。

この例では Camera コンポーネントの映像をそのまま画面上に描画しましたが、リアルタイムに動画を解析して参加者のジェスチャーを検出したりできればぐっと応用範囲は広がるでしょう。現状の PC の性能ではフレームレートや遅延の観点から自然な速度で実行するのは難しいと思いますが、この分野の技術の進歩はとても速いので、近い将来に実用的になるかもしれません。

チェックリスト

- Camera コンポーネントを使って動画記録を行うことができる。
- Camera コンポーネントで記録された動画の遅延を確認することができる。
- Camera コンポーネントの映像をリアルタイムに画面上に表示することができる。

10.8 この章のトピックス

10.8.1 Static コンポーネントを用いた動画の読み込み

動画ファイルは一般的にファイルサイズが大きく、読み込みに時間がかかります。ループで繰り返しのたびに異なる動画を読み込むと、ルーチン開始前に読み込みを行いますので、ここで時間がかかると繰り返しのたびに意図しない空白画面が表示され続けることになります。ただ時間がかかるだけならまだしも、動画ファイルによって読み込みの時間が異なると試行間の間隔がばらばらになってしまって実験によっては望ましくありません。こういう時に便利なのが Static コンポーネントです。

Static コンポーネントはコンポーネントペインの「カスタム」の中に含まれる 図 10.8 のアイコンです。他のコンポーネントとは異なり、**[名前]** の初期値がコンポーネント名と同じではなく ISI となっているので注意してください。**[開始]** と **[終了]** は他のコンポーネントと同様、Static コンポーネントが有効な期間を指定します。

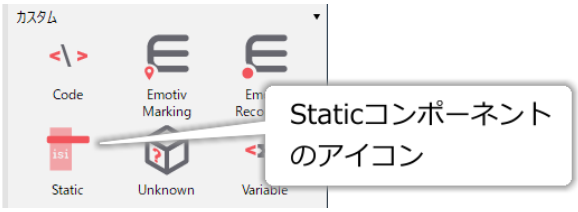


図 10.8　Staic コンポーネントのアイコン

プロパティ設定ダイアログの OK をクリックしてダイアログを閉じると、ルーチンペインに 図 10.9 上のように赤色の領域が現れます。これが Static コンポーネントです。削除するときはこの赤い領域内のどこかで右クリックしてメニューから「削除」を選んでください。

Static コンポーネントを配置した後に、他のコンポーネントのプロパティ設定ダイアログを開いた際に、各プロパティの更新方法として「更新方法: trial.ISI」のような項目が追加されます (図 10.9 下)。ここで trial は Static コンポーネントを置いているルーチン名、ISI はスタティックコンポーネントの名前です。test というルーチンに load_stim という名前で Static コンポーネントを配置したなら test.load_stim となります。この項目を選択すると、プロパティの更新が指定した Static コンポーネントの期間中に行われます。例えば実験で使用する動画が最も読み込みに時間がかかるものでも 0.4 秒で完了するなら、Static コンポーネントの長さを (少し余裕をもって)0.5 秒にしておけば、どの動画も 0.5 間で読み込むことができてばらつきが生じません。Static コンポーネントで設定した期間内に終わらない処理を行わせると元も子もないので、あらかじめ動作確認して余裕を持たせておくことが重要です。

「Static コンポーネント」の名前の通り、この期間には刺激を描画したりキー押しを検出したりするべきではありません。やってできない事はないのですが、時間的な精度が保障されなくなります。Static コンポーネントの期間中 (あるいは期間開始と同時に) に静止した刺激を描画しておくことには何の問題もありません。

ファイルの読み込みタイミングとして他のルーチンに配置した Static コンポーネントを選択することも可能なので、実験期間中で都合がよいタイミングにファイルを読み込んでくことが可能です。Static コンポーネントはファイルサイズが大きくなりがちな動画ファイルの読み込みに特に力を発揮しますが、音声ファイルを読み込む時や、画像ファイルを十数枚一気に読み込む必要がある時などにも役に立ちます。

「カスタム」タブの **[カスタムコード]** は、他のコンポーネントのパラメータの更新以外の作業を行わせたいときに使用します。入力欄が狭いので、複雑な処理を行わせる場合は Code コンポーネントで関数を定義してその関数を呼び出すなどの工夫が必要かもしれません。上級者向けの機能だと思います。

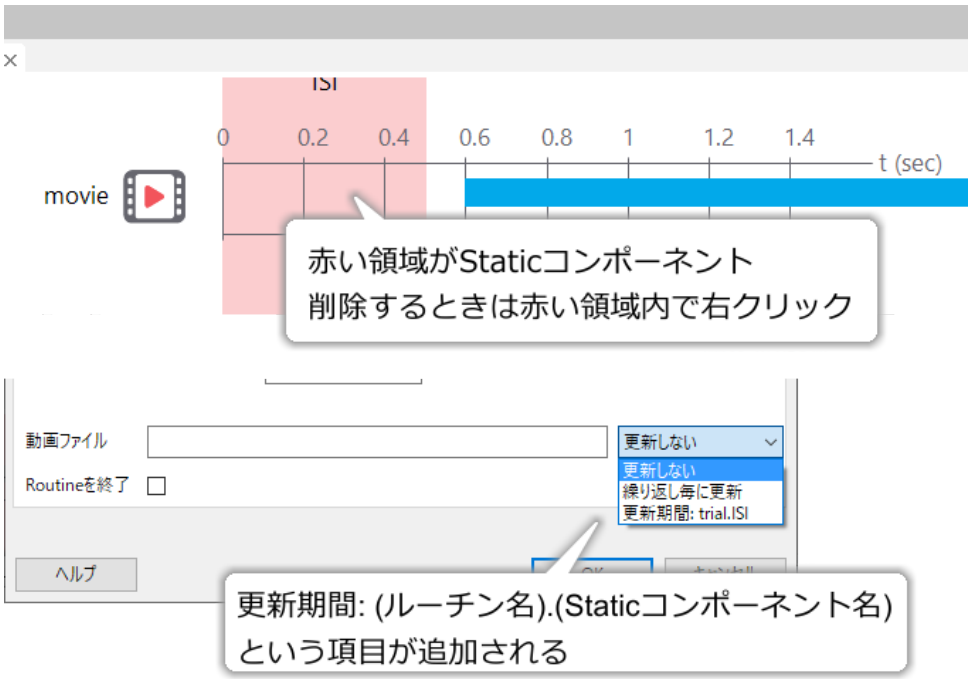


図 10.9　Static コンポーネントを設置すると他のコンポーネントの更新方法のところに項目が追加されます

第 11 章

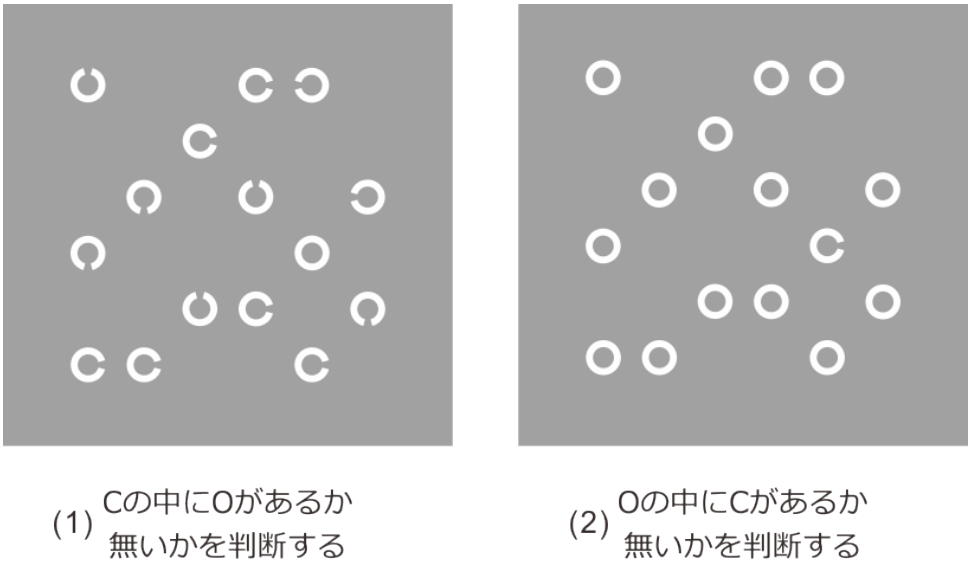
無作為化しよう―視覚探索

11.1 この章の実験の概要

ここまでの解説において、Builder がおこなう無作為化は「繰り返し順序の無作為化」でした。つまり、繰り返されるパラメータはすべて事前に条件ファイルで与えられていて、繰り返し順序だけが無作為になっていました。これで充分な実験手続きも多いのですが、パラメータ自体を無作為に決定したい場合もあります。そういった手続きの例として、本章では視覚探索課題を取り上げます。

図 11.1 に本章の実験の概要を示します。 図 11.1 の (1) では、スクリーン上に切れ目が入っている円 (以下 C と表記) が複数個提示されていますが、50% の確率で一つだけ切れ目がない円 (以下 O と表記) が含まれています。実験参加者は、できるだけ速く正確に、O が含まれているか居ないかを判断しなければいけません。容易に予想できる事ですが、O の有無を判断するまでに要する平均時間はスクリーン上に提示されている図形 (以下アイテムと表記) の個数にほぼ比例して増加します。ところが、 図 11.1 の (2) のように、O と C を入れ替えて、複数個の O の中から C の有無を判断する課題に切り替えると、アイテム数が増えても (1) ほど反応時間が増加しません。この現象を視覚探索の非対称性と呼び、 図 11.1 下のようにアイテム数と平均探索時間の関係をプロットしたグラフを探索関数と呼びます。参加者が探し出すべきアイテムをターゲットと呼び、それ以外のアイテムをディストラクタと呼びます。(1) の課題では O がターゲットで C がディストラクタ、(2) の課題では C がターゲットで O がディストラクタです。この章では、アイテム数を 5 個、10 個、15 個と変化させながら (1) の課題と (2) の課題を行う実験を作成します。

図 11.2 に具体的な手続きを示します。実験を実行すると、準備ができたらカーソルキーの左右を押すように促す教示が提示されます。このスクリーンを 1. とします。実験参加者が自分でキーを押すことによって実験が始まります。各試行の最初には、スクリーン中央に固視点として **[文字の高さ $]** が 24pix の「＋」の文字が提示されます。このスクリーンを 2. とします。実験参加者は固視点を注視しながら刺激の提示を待ちます。待ち時間は試行毎に 1.0 秒、1.5 秒、2.0 秒のいずれかから無作為に選びます。待ち時間が終了したら、スクリーン上に刺激が提示されます。刺激はアイテム数が 3 種類 (5 個、10 個、15 個) ×アイテムの種類が 4 種類 (すべて O、すべて C、O の中にひとつだけ C、C の中にひとつだけ O)=12 種類のいずれかです。実験参加者は、刺激の中に「ひとつだけ周囲と異なるアイテムが存在するか否か」を判断します。ひとつだけ周囲と異なるアイテムが存在する場合はカーソルキーの右、すべて同じアイテムの場合はカーソルキーの左をできるだけ速く、正確に押します。反応に制限時間は設けず、参加者が反応するまで刺激を提示します。参加者が反応したら自動的に次の試行が開始されスクリーン 2.(固視点が提示される画面) へ戻りますが、80 試行毎に休憩のためにスクリーン 1. へ戻って参加者のスペースキー押しを待ちます。12 種類の条件に対して 20 試行ずつ、合計 240 試行を無作為な順序に実施したら実験は終了します。総試行数が 240 試行で 80 試行毎にスクリーン



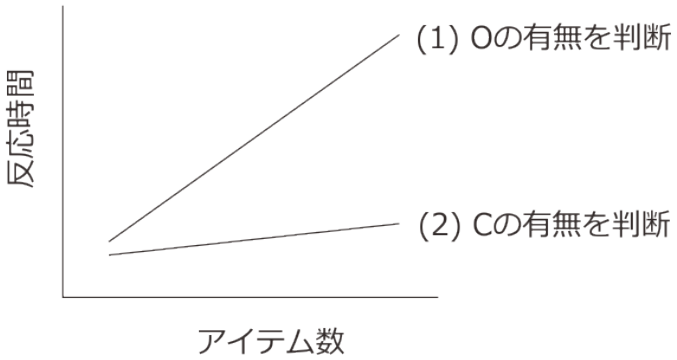


図 11.1　視覚探索の非対称性。スクリーン上に提示されているアイテム数が多いほど判断に時間がかかりますが、C のなかから O を探すより、O の中から C を探す方がアイテム数増加に伴って反応時間が急激に増加します。

1. が挿入されるのですから、スクリーン 1. は実験開始直後、80 試行終了時、160 試行終了時の 3 回提示されます。

刺激の詳細を 図 11.3 に示します。アイテムの位置はスクリーン中央に設定された仮想的な 6 × 6 の格子上から無作為に選ばれます。格子の各マスの幅および高さは 100pix です。スクリーン中央の座標が [0, 0] で、グリッドの全幅は 500pix ですから、グリッドの一番右上の座標は [250,250]、一番左下の座標は [-250, -250] です。一番上の段の座標を左から右に向かって順番に書くと、[-250, 250]、[-150, 250]、[-50, 250]、[50, 250]、[150, 250]、[250, 250] です。

アイテムの直径は O、C ともに 30pix とし、C の場合は円の一部に幅 10pix の切れ目を入れます。切れ目の位置は、アイテムの中心から見て右を 0 度として、時計回りに 0 度、90 度、180 度、270 度の 4 種類の中からアイテム毎に無作為に選択します。

以上が実験の概要です。Builder が苦手とするポイント、できる事なら Builder で作りたくないなあと思ってしまうポイントがいくつか含まれています。これらのポイントはお互いに関連しあっているのですが、以下の 4 つにまとめられると思います。

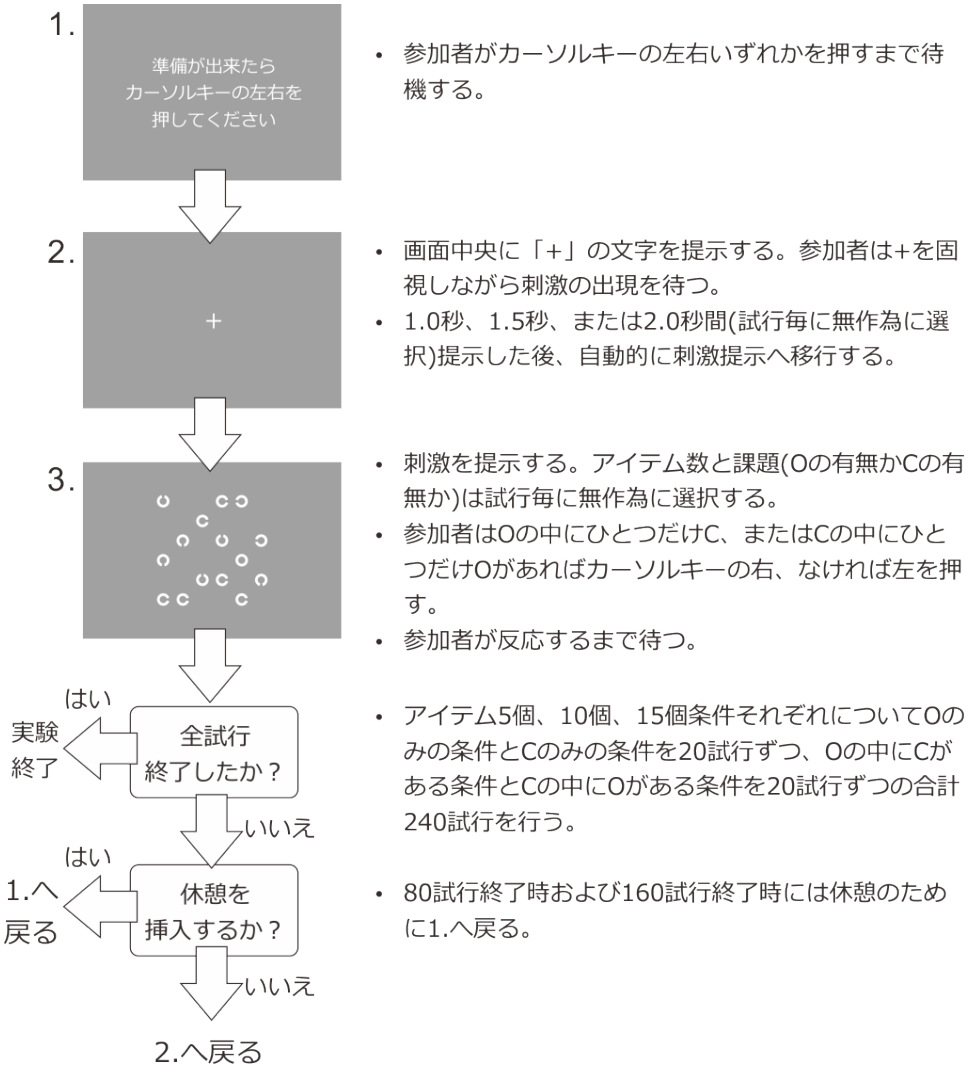


図 11.2　実験の手続き。

1. 独立に位置や形状が変化するアイテムが最大 15 個スクリーン上に存在する。
2. 試行毎にスクリーン上に出現するアイテムの個数が異なる。
3. 無作為に設定するパラメータが複数個ある。
4. 休憩が挿入される試行が条件数の倍数になっていない。

まず 1. と 2. についてですが、大量の視覚刺激が配置されている実験を準備するとき、まっさきに考えたいのが「複数の刺激をまとめてひとつの画像にできないか」ということです。ひとつの画像にできるのなら Image コンポーネントひとつで解決するので話は簡単です。しかし、今回の実験の場合、個々の刺激がそれぞれ O になったり C になったり、位置が変化したりするので、事前に画像ファイルとして用意するなら膨大な枚数が必要となります。頑張って画像ファイルを作成するのも選択肢の一つですが、本書の目的は Builder でのテクニックを解説することなので、Builder の力で何とかしたいところです。一方、個々の刺激をひとつの Image コンポーネントで描画すれば形状や位置の変化に対応しやすくなりますが、試行毎に個数が変化するというのも今までの実験にはなかったことなので、やはり対策を考えいないといけません。

続いて 3. と 4. ですが、これらはいずれも条件ファイルに関わる問題です。今までの章では、無作為に変化す

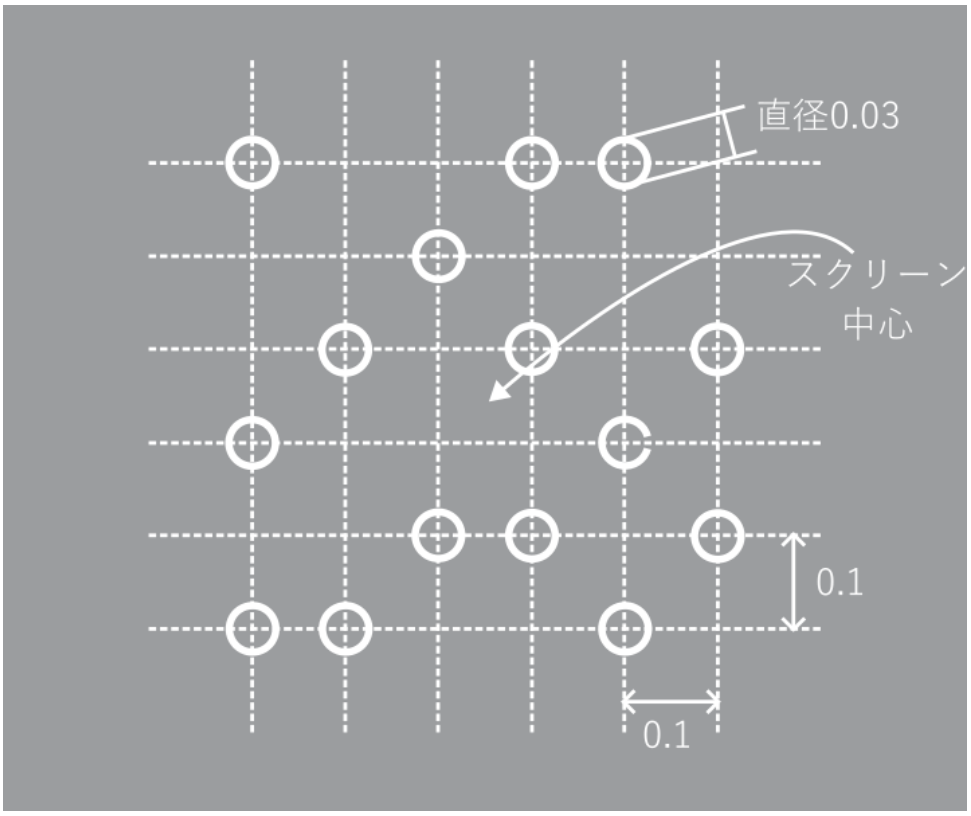


図 11.3　刺激の配置。アイテムの位置は仮想的な 6 × 6 のグリッド上から試行毎に無作為に選択されます。

るパラメータは条件ファイルで値を設定してきました。条件ファイルを使う場合は、すべてのパラメータの「組み合わせ」を明示的に記述しなければいけません。今回の実験を今までの章のように条件ファイルで作成しようとすると、固視点の提示時間が 3 種類ありますので、12 種類の刺激と掛け合わせて 36 条件の条件ファイルになります。この条件ファイルを使うと実現可能な試行数は 36 の倍数になりますが、240 は 36 で割り切れませんので、この時点で全試行数を 240 試行にすることは不可能になってしまいました。固視点の提示時間を 1.0 秒と 1.5 秒の 2 種類に減らすと 12 種類の刺激との掛け合わせで 24 条件の条件ファイルとなり、240 試行にすることが可能になります。しかし、今回の実験ではアイテムの位置も C の向きもすべて無作為なのです。どう工夫しても、これまでの章の条件ファイルと同様の考え方では、総試行数が 240 試行となる条件ファイルを作成することはできません。

さて困りましたが、本章でのアプローチは「きちんと各条件の試行回数などのバランスがとられてきて、繰り返し順序だけが無作為なもの」と「本当に無作為でいいもの」を切り分けて、「本当に無作為でいいもの」は実行時に Builder に決めさせようというものです。今回の実験では、アイテム数が 5、10、15 個の条件が均等な回数実施されること、ターゲットありの条件となしの条件が均等な回数実施されること、O がターゲットの条件と C がターゲットの条件が均等な回数実施されることの 3 点は死守したいと思いますので、これらは条件ファイルに任せます。一方、アイテムが描画される位置、C の角度、固視点の提示時間は「本当に無作為でいいもの」とします。無作為なので、実験を実施した後に確認すると均等になっていないかもしれませんが、無作為とはそもそもそういうものです。前置きが長くなりましたが、そろそろ実験の作成に入りましょう。

11.2 実験の作成

準備作業

- この章の実験のためのフォルダを作成して、その中に exp11.psyexp という名前で新しい実験を保存する。

- 実験設定ダイアログ
  - PsychoPy の設定で height 以外の単位を標準に設定している場合は **[単位]** を height にしておく。
  - 「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。

- trial ルーチン
  - Code コンポーネントをひとつ配置して、 **[名前]** を `codeTrial` にする。今はコードを入力しない。
  - Polygon コンポーネントをひとつ配置して、以下のように設定する。
    - 「基本」タブの **[名前]** を `fixpoint` に **[終了]** を `delay` にする。 **[形状]** を十字にする。
    - 「レイアウト」タブの **[サイズ [w, h] $]** を `(0.03,0.03)` にする。
  - Keyboard コンポーネントをひとつ配置して、以下のように設定する。
    - 「基本」タブの **[名前]** を `key_resp_trial` にする。 **[開始]** を `delay` に、 **[終了]** を空白にする。
    - 「基本」タブの **[Routine を終了]** がチェックされていることを確認し、 **[検出するキー $]** を `'right', 'left'` とする。
    - 「データ」タブの **[正答を記録]** をチェックし、 **[正答]** に `$correctAns` と入力する。
  - Image コンポーネントをひとつ配置して、以下のように設定する。
    - 「基本」タブの **[名前]** を `image00` にする。 **[開始]** を `delay` に、 **[終了]** を空白にする。 **[画像]** に `imagefile[0]` と入力し、「繰り返し毎に更新」設定する。
    - 「レイアウト」タブの **[サイズ [w, h] $]** に `(0.03, 0.03)` と入力する。 **[位置 [x, y] $]** に `pos[0]` と入力し、「繰り返し毎に更新」に設定する。 **[回転角度 $]** に `ori[0]` と入力し、「繰り返し毎に更新」に設定する。

  ~~i~~mage00 **をコピーし、** `image01` **の名前で貼り付けて以下のように設定する。**
    - 「基本」タブの **[画像]** を ``imagefile[1]``にする。
    - 「レイアウト」タブの **[位置 [x, y] $]** を `pos[1]` 、 **[回転角度 $]** を `ori[1]` にする。

— 続けてコンポーネントの貼り付けを操作して (image00 をコピーした状態になっているはず)、名前が image14 に達するまで 1 ずつ数値を増やしながら作業を繰り返す。 **[画像]** 、 **[位置 [x, y] $]** 、 **[回転角度 $]** の [ ] 内の数値も 1 ずつ増やしていく。作業が終了したら、「実験内を検索...」を使って imagefile、pos、ori を検索してインデックスが適切に変更されているか (これらのパラメータの [ ] の中の値が image07 であれば 7、image11 であれば 11 といった具合に一致してるか) 確認する。

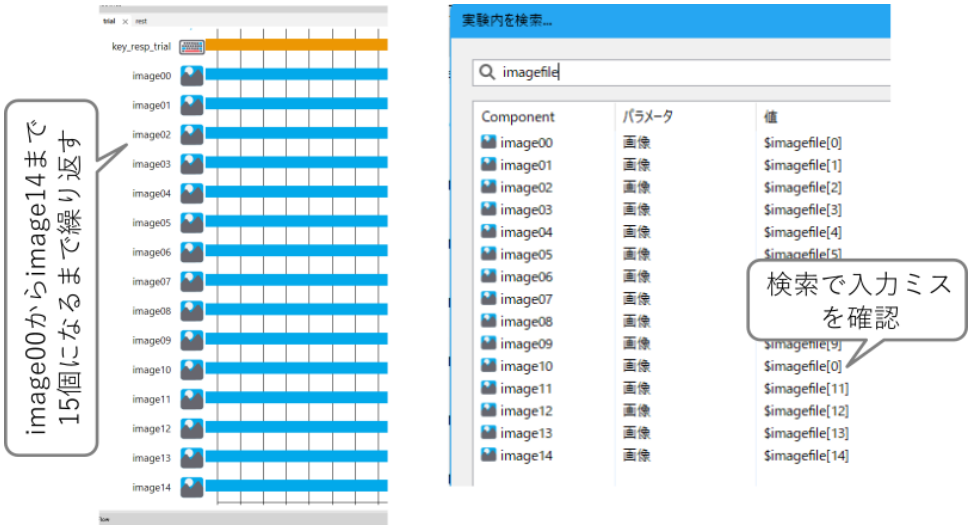


図 11.4　iamge00 をコピーして名前の数値部分とパラメータのインデックスを 1 ずつ増やしながら image14 に達するまで繰り返す。終了したら「実験内を検索...」を使って入力ミスがないか確認するとよい。図右の例では image10 に対する imagefile の [ ] 内のインデックスが 0 のままになっており修正の必要がある。

- rest ルーチン (作成する)
  - フローの trial ルーチンの直前に挿入する。
  - Code コンポーネントをひとつ配置して、 **[名前]** を `code_rest` にする。今はコードを入力しない。
  - Text コンポーネントをひとつ配置して、以下のように設定する。
    - 「基本」タブの **[名前]** を `textRest` にし、 **[開始]** を `0` 、 **[終了]** を空白にする。 **[文字列]** に **準備ができたらカーソルキーの左右どちらか一方を押してください** と入力する。
    - **[文字の高さ $]** を `0.04` にする。
  - trial ルーチンの key_resp_trial をコピーして、rest ルーチンに `key_resp_rest` という名前で貼り付けて以下のように設定する。
    - 「データ」タブの **[記録]** を「なし」にする。 **[正答を記録]** のチェックを外す。
- trials ループ (作成する)
  - rest ルーチンと trial ルーチンを繰り返すように挿入する。
  - **[繰り返し回数 $]** に `20` と入力する。
  - **[条件]** に `exp11cnd.xlsx` と入力する。先に exp11cnd.xlsx を作成してから「選択…」ボタンをクリックして選択してもよい。
- exp11cnd.xlsx(条件ファイル)
  - numItems、firstItem、otherItems、correctAns という名前のパラメータを定義する。
  - numItems は 5、10、15 の 3 種類、firstItem は o.png、c.png の 2 種類、otherItems も o.png と c.png の 2 種類の値をとる。これらの全ての組み合わせを入力する。パラメータ名を定義する行を除いて

3 × 2 × 2=12 行の条件ファイルとなる。

- firstItem と otherItems が同じ行の correctAns の列に left と入力する。firstItem と otherItems が異なる行の correctAns の列に right と入力する。

- o.png および c.png(刺激画像ファイル)
  - 背景が透過した PNG ファイルを作成し、o.png という名前で exp11.psyexp と同じフォルダに保存する。o.png には白色の円を描く。o.png を c.png という別名で保存し、円の中心右側に切れ目を入れて保存する (30 × 30pix で作成したものを http://s12600.net/psy/python/ppb/index.html#ppb-files からダウンロードできます)。

11.3 特定の条件を満たさない時に実行されるルーチンを活用して休憩画面を設けよう

ではまず、80 試行毎に表示される休憩画面の設定から始めましょう。古いバージョンの PsychoPy では Code コンポーネントを使って工夫しないといけなかったのですが、「4.9: 動作確認のために一部の動作をスキップしよう」で紹介した「Routine の設定」を使って簡単に実現できるようになりました。

rest ルーチンを開いて、ルーチンペインの左上にある「Routine の設定」をクリックしてください。rest ルーチンのプロパティ設定ダイアログが開くので、「Flow」タブに **[条件に合致する場合はスキップ... $]** という項目があるのを確認してください。ここに評価結果が True または False になる式を書くと、True のときにこのルーチンをスキップすることができます。今回の実験では、80 試行毎に休憩画面として rest ルーチンを表示させたいので、「9.11: 軌跡データを間引きしよう」の剰余演算子を使ったテクニックを使って

```
trials.thisN % 80 != 0:
```

とすると trial.thisN が 0、80、160 以外の時に True となり目標を達成できます。trial ループのデータ属性 thisN は 表 7.1 で出てきた「このループで実行済みの繰り返し回数」でしたね。thisN は 0 から始まって 240 回目の繰り返しでは 239 なので、240 になることはない点に注意してください。では、上記の式を **[条件に合致する場合はスキップ... $]** に入力してください。

式を入力すると 図 11.5 のように != が ≠ に変換されてしまいますが (2024.2.5 で確認)、表示だけの問題なので気にしないでください。個人的には Python の式が書かれるべき場所なのでこういった小細工なしで正しい Python の式を表示してほしいのですが、まあ仕方ありません。

なお、この方法だと 0 回目、つまり trials ループに入って 1 回目の繰り返しにも rest ルーチンが表示されます。しかし、実験の手続きによっては、最初はループに入る前に詳細な教示画面を出して 80 回目、160 回目は休憩を促す簡素なメッセージにしたいといったケースも考えられます。その場合は論理演算を使って「trials.thisN を 80 で割った余りが 0 でないか、trials.thisN が 0」といった条件式にすることもできますが、80 回目と 160 回目の 2 回しかないのですから

```
trials.thisN not in (80, 160)
```

のように列挙してしまう方が楽かもしれません。

チェックリスト

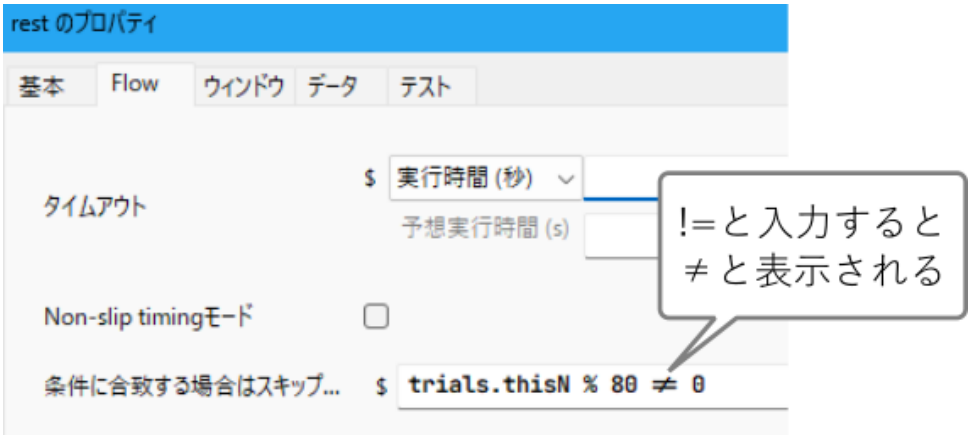


図 11.5　iamge00 をコピーして名前の数値部分とパラメータのインデックスを 1 ずつ増やしながら image14 に達するまで繰り返す。終了したら「実験内を検索...」を使って入力ミスがないか確認するとよい。図右の例では image10 に対する imagefile の [ ] 内のインデックスが 0 のままになっており修正の必要がある。

- 指定した条件に当てはまる時にルーチンの実行をスキップさせることができる。

11.4 Code コンポーネントを使って無作為に固視点の提示時間を選択しよう

続いて配置済みの Code コンポーネントにコードを入力していきましょう。以下の処理を実現する必要があります。

- trial ルーチンで使われている変数 delay の値を決定する。固視点の **[開始]** が 0 で **[終了]** が delay に設定され、image00 から image14 の **[開始]** が delay に設定されているので、固視点が出現した delay 秒後に固視点が消失して代わりに image00 から image14 が提示される。
- ori[0] から ori[14] の値を決定する。値は試行毎に 0、90、180、270 から無作為に選択する。
- pos[0] から pos[14] の値を決定する。値は試行毎に 図 11.3 に示したグリッドから重複がないように無作為に選ぶ。
- imagefile[0] から imagefile[14] の値を決定する。この値の決め方については後述する。

これらの問題はいずれも「無作為に選択する」という点で共通しているので、同じ方法で解決できます。しかし、delay の決定以外は「アイテム数が 5 個、10 個、15 個と変化することにどう対応するか」という問題と併せて考えないといけないので、次節でまとめて考えることにしましょう。まずは delay の問題を解決します。

無作為に値を選択するには、Builder が内部で用意している乱数関数 (表 11.1) を用います。使い方は非常に単純で、randint(0,5) と書けば 0 から 4 の整数の一様乱数からサンプルをひとつ得ることができます。引数 high「未満」ですから 5 を含まない点に注意してください。同様に、normal(50, 10) と書けば平均値 50、標準偏差 10 の正規乱数からサンプルをひとつ得ることができます。引数 size は、randint(0, 5, size=3) のように使用します。この例の場合、戻り値は [0, 2, 1] といった具合に 0 から 4 の整数の一様乱数からサンプルを 3 つ並べたシーケンス型データが得られます。正確に書くとこの戻り値は numpy.ndarray クラスのインスタンスなのですが、numpy.ndarray について説明すると脱線が長くなるので「 11.9.1:numpy.ndarray 型について 」を参照してください。

表 11.1　Builder で利用できる乱数関数。いずれも numpy.random から import されています。

random(size = None)	0.0 以上 1.0 未満の一様乱数のサンプルを返す。size が None の時 (初期値) にはひとつのサンプルを、自然数の場合には size 個のサンプルを返す。
randint(low, high, size = None)	low 以上 high 未満の範囲の整数の一様乱数のサンプルを返す。low と high は整数でなければならない。high が None の時には 0 以上 low 未満の範囲と見なされる。size の働きは random( ) と同様。
normal(loc=0.0, scale=1.0, size = None)	正規分布に従う乱数のサンプルを返す。loc は平均値、scale は標準偏差に対応する。size の働きは random( ) と同様。
shuffle(x)	リストなどの要素を変更可能なシーケンス型データの要素を無作為に並べ替える。戻り値はない。x の元の順序は失われてしまう点に注意。
randchoice(x)	リストなどの要素を変更可能なシーケンス型データの要素からひとつの値を無作為に取り出す。

さて、今回の実験のように、複数個の選択肢からひとつを無作為に選びだすという用途には randchoice( ) が便利です。delay の値は 1.0、1.5、2.0 の 3 通りなので、これを並べたシーケンスを引数として以下のように randchoice( ) を呼ぶだけです。

```
delay = randchoice( (1.0, 1.5, 2.0) )
```

exp11.psyexp の trial ルーチンを開いて、codeTrial にこの式を入力しましょう。試行毎に delay の値を変化させるのですから、入力すべき場所は **[Routine 開始時]** です。

あっさり解決してしまったので、もう少し無作為選択について触れておきましょう。まず、randchoice はシーケンスの要素にできるものなら何に対しても使用できます。以下の例では文字列を並べたタプルを実験開始時にあらかじめ tasklist という変数に代入しておいて、ルーチンの開始前に randchoice( ) で選択を行っています。現在の PC の処理能力だとルーチン開始のたびにタプルを作成しても問題になることはないでしょうが、まあ繰り返しのたびに変化しないものは繰り返しが始まる前に一度処理するだけにする習慣をつけておくのは良いことだと思います。

```
# 「実験開始時」にタプルを初期化
tasklist = ('一致', '不一致')

# 「Routine 開始時」に選択
tasktype = randchoice( tasklist )
```

今回のように数個しか候補がなければ randchoice が便利なのですが、多数の数値の中からひとつを選択しないといけない場合、random( ) や randint( ) を使った方がよい場合があります。例として 1.0、1.5、2.0 を randint( ) で生成してみましょう。randint(0, 3) とすれば戻り値として 0、1、2 の乱数が得られますから、戻り値に 0.5 を掛ければ 0.0、0.5、1.0 の乱数が得られます。ここへさらに 1.0 を加えると、1.0、1.5、2.0 の乱数が得られ

ます。従って、以下のコードで delay の値に 1.0、1.5、2.0 の中から無作為にひとつ選んで設定できます。

```
delay = randint(0, 3)*0.5 + 1.0
```

なお、web 上で Python の乱数について検索すると、randint(low, high) は「low 以上 high『以下』の値を返す」と書かれている資料がヒットするかもしれません。非常に紛らわしいのですが、このような資料で紹介されている randint( ) と、Builder が内部で参照している乱数関数の randint( ) とは別の関数です。「low 以上 high『以下』の値を返す」randint( ) は、Python の random モジュールから import されています。ですから、モジュール名を省略せずに書けば random.randint( ) という関数です。一方、Builder が内部で準備している randint( ) は numpy というパッケージのサブモジュール numpy.random から import されています。省略せずに書けば numpy.random.randint( ) です。気をつけてください。

チェックリスト

- 試行毎にシーケンスからひとつの値を無作為に選択するコードを書くことができる。
- low 以上 high 未満の整数を範囲とする一様乱数のサンプルをひとつ得るコードを書くことができる。(low、high は整数)

11.5 Code コンポーネントを使って無作為にアイテムの各パラメータを決めよう

delay の問題が解決したので、続いてアイテムの個数、種類、位置、回転角度を決める方法について考えましょう。一度にすべてのパラメータについて考えるのは大変なので、まずは「アイテムが 15 個の場合」に限定して考えます。

まず、簡単に解決できるのが回転角度の決定です。15 個の Image コンポーネントの **[回転角度 $]** は、ori[0]、ori[1]、…、ori[14] というリストの値がすでに入力されています。ですから、ori という要素数 15 のリストを作成して、要素に 0、90、180、270 のいずれかの値を無作為に割り当てればよいだけです。どうせ毎試行 ori の値は更新するので、最初に ori を作成する時には値は何を設定しても構いません。例えば以下のように 0 を 15 個並べたリストを作成してもよいでしょう。

```
ori = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

別にこのコードで全く問題はないのですが、もし要素が 100 個必要になった場合、100 個も 0 を並べたリストを入力するのは面倒です。そのような時に便利な関数が range( ) です。range( ) は 1 個から 3 個の整数を引数として取ることができます。引数が 1 個の場合は、0 から引数より 1 小さい整数まで順番に取り出すことができる「range オブジェクト」というオブジェクトが得られます。range オブジェクトの詳細については「 11.9.2:range( ) オブジェクトについて 」をご覧ください。range(　) を list オブジェクトを作る関数 list( ) に渡すと、0 から引数より 1 小さい整数までを並べたリストが得られます。例えば list(range(15)) とした場合、以下のリストが戻り値として得られます。

```
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]
```

引数が 2 個 (x, y とします) の場合は、x から y-1 までの整数を取り出す range オブジェクトが得られます。例えば list(range(10, 15)) を実行すると以下のリストが得られます。

```
[10,11,12,13,14]
```

引数が 3 個の場合、整数が 1 ずつ増加するのではなく 3 個目の引数の値ずつ増加します。list(range(0,15,3)) を実行すると、以下のように 3 ずつ増加する整数のリストが得られます。15 は含まない点に注意してください。

```
[0,3,6,9,12]
```

3 個目の引数が負の場合は、だんだん値が小さくなっていきます。list(range(15,0,-3) を実行すると、以下のように 15 から始まって 3 ずつ減少するリストが得られます。上の例と同様に 0 は含まない点に注意してください。

```
[15, 12, 9, 6, 3]
```

range( ) が本領を発揮するのは、list( ) ではなく for 文と組み合わせるときです。for 文と組み合わせると、ブロックが繰り返される毎に range( ) から順番に値が取り出されるので、要素数が等しい複数のリストに対してまとめて処理を行うコードが簡単に書けます。まず、0 が 15 個並んだリストを作成してみましょう。

```
ori = [ ]
for i in range(15):
    ori.append(0)
```

これで 0 が 15 個並んだリストが変数 ori に格納されましたが、これだけでしたら先ほどのようにずらっと 0 を 15 個並べたほうが楽だと思われるかもしれませんね。ここへ、各アイテムの種類 (O か C か) を格納する変数 imagefile を準備する処理も組み込んでみましょう。imagefile は Image コンポーネントので使用されますので、その要素は画像ファイル名でなければいけません。こちらも試行毎に値を変更するのでとりあえず O と C のどちらを設定しておいても構いません。とりあえず O で初期化しておくことにしましょう。O の刺激は o.png と画像ファイルに対応していますので、'o.png' という文字列を 15 個並べたリストを作成する必要があります。ori を作成した時と同じ要領で考えると、以下のコードで実現できます。

```
ori = [ ]
imagefile = [ ]
for i in range(15):
    ori.append(0)
    imagefile.append('o.png')
```

刺激の位置を格納する変数 pos の準備もここへ組み込むことができます。要素は **[位置 [x, y] $]** で使用するので、要素数 2 のリスト [0, 0] で初期化しておきましょう。

```
ori = [ ]
imagefile = [ ]
pos = [ ]
for i in range(15):
    ori.append(0)
    imagefile.append('o.png')
```

(次のページに続く)

(前のページからの続き)

```
    pos.append([0, 0])
```

以上で ori、imagefile、pos の準備は完了です。このコードは実験開始時に一回実行すればよいので、**[実験開始時]** に入力しておきましょう。

変数の準備ができたので、続いて各試行の最初に無作為にこれらの変数の値を決定するコードを作成しましょう。まず、ori については delay と同じ方法が使えます。Randint(0, 4) で 0 から 3 の整数の乱数を得て、90 倍すれば 0、90、180、270 の乱数が得られますので、for 文で ori[0] から ori[14] に代入すればいいでしょう。この方法では回転させる必要がない O も回転させてしまいますが、O は回転させても見た目が同じなので実質的に問題とはなりません。以下のコードを **[Routine 開始時]** に追加して下さい。

```
for i in range(15):
    ori[i] = 90*randint(0,4)
```

続いて imagefile の設定ですが、こちらは少し解説が必要です。すべて O の条件、すべて C の条件、O の中にひとつだけ C の条件、C の中にひとつだけ O の条件の 4 条件があるのでした。そして、条件ファイルを確認には firstItem と otherItems というパラメータが定義されています。firstItem は imagefile[0]、otherItems は imagefile[1] から imagefile[14] に設定することを想定しています。 図 11.6 をご覧ください。firstItem と otherItems がともに o.png であれば「すべて O」の条件に、ともに c.png であれば「すべて C」の条件になります。同様に firstItem が c.png で otherItems が o.png であれば「O の中にひとつだけ C」、firstItemg が o.png で otherItems が c.png であれば「C の中にひとつだけ O」になります。「必ず image00 ターゲットになっても問題はないの？」と思われる方がおられるかも知れませんが、試行毎に位置を無作為に決定するので問題ありません。

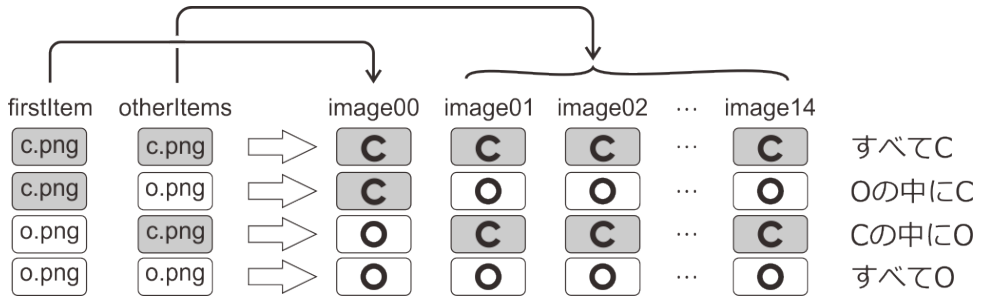


図 11.6　firstItem と otherItems のパラメータ値と刺激条件との対応。

先ほど **[Routine 開始時]** に入力した ori を更新する for 文に、imagefile を更新するコードを追加しましょう。変数 i の値が 0 の時には imagefile[i] に firstItem の値を設定し、i の値が 0 以外の時には imagefile[i] に otherItems の値を設定します。

```
for i in range(15):
    ori[i] = 90*randint(0,4)
    if i==0:
        imagefile[i] = firstItem
    else:
        imagefile[i] = otherItems
```

ori、imagefile の更新ができたので、あとはアイテムの位置に対応する変数 pos の更新です。pos は「無作為に値を決定する」という点では ori と同じですが、重要な違いがあります。ori はアイテム間で値が重複しても構いません。つまり、例えば同時に回転角度が 90 度のアイテムが複数個存在しても構いません。一方、pos はアイテム間で値が重複するとアイテムが重なってしまいますので、値の重複は許されません。次の節では、pos の値を決定する方法を考えます。

チェックリスト

- range( ) を用いて、0 から n (n>0) までの整数を順番に取り出す range オブジェクトを作成することができる。
- range( ) を用いて、m から n (n>m) までの整数を順番に取り出す range オブジェクトを作成することができる。
- range( ) を用いて、m から n まで、s の間隔で整数を順番に取り出す range オブジェクトを作成することができる。ただし m、n は互いに異なる整数、s は非 0 の整数である。

11.6 無作為に重複なく選択しよう

それでは改めて、pos の値の決定方法を考えましょう。pos の候補となる値は 図 11.3 に示したグリッドの座標で、36 個あります。これら 36 個の値の中から 15 個を重複なく無作為に選択しなければいけません。心理学実験においては、この例のように複数個の値を重複なく無作為に選択しなければいけないことがよくあります。このようなときは、「無作為に選択する」のではなく「無作為に並べ替える」という方法が有効です。

まず、36 個の座標値をすべて並べたリストを作成して poslist という変数に格納しておきましょう。以下のように for 文を重ねると簡単に作成できます。append している行の式については、グリッドの間隔が 0.1 で一番左下の座標が [-0.25, -0.25] だったことを思い出してください。この多重 for 文を実行したときに poslist に値が追加されていく様子を 図 11.7 に示しましたので、多重 for 文の動作がイメージしにくい方は参考にしてください。この多重 for 文を codeTrial の **[実験開始時]** に追加してください。

```
poslist = [ ]
for pos_y in range(6):
    for pos_x in range(6):
        poslist.append([0.1*pos_x-0.25, 0.1*pos_y-0.25])
```

続いて、作成した poslist の要素を無作為な順序に並び替えます。並び替えには 表 11.1 で紹介した shuffle( ) を用います。shuffle( ) は引数として受け取ったリストの要素の順番を無作為に並び替えます。戻り値は返さないので、ただ shuffle(poslist) と書けば poslist の要素を並び替えることができます (suffled_list = shuffle(original_list) と書くのはよくある間違い)。試行毎にアイテムの位置を変更したいので、 **[Routine 開始時]** に記入してください。

さて、ここからがポイントです。poslist の要素はルーチンの開始時に無作為に並び替えられているのですから、poslist の先頭から順番に 15 個の要素を取りだせば、poslist の中から重複なしに無作為に 15 個の要素を取り出したことになります。従って、以下のコードで pos[0] から pos[14] に重複なく無作為に位置を割り当てることができるはずです。

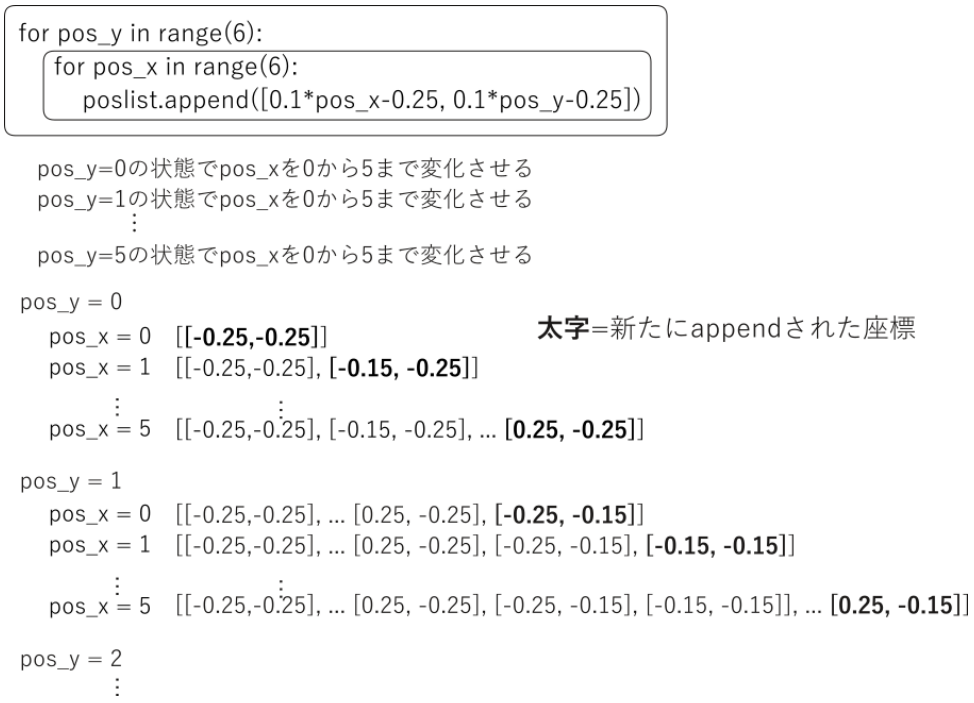


図 11.7　多重 for 文による座標値リストの作成。

```
for i in range(15):
    pos[i] = poslist[i]
```

for i in range(15): という繰り返しは先ほど ori と imagefile の値を設定する時にも使用したのですから、同じ for 文に pos[i] = poslist[i] を挿入するだけで OK です。確認のため、delay の設定や shuffle も含めた **[Routine 開始時]** 全体のコードを示しておきます。

```
delay = randchoice( (1.0, 1.5, 2.0) )
shuffle(poslist)
for i in range(15):
    ori[i] = 90*randint(0,4)
    pos[i] = poslist[i]
    if i==0:
        imagefile[i] = firstItem
    else:
        imagefile[i] = otherItems
```

これで「アイテム数が 15 個の場合」に限定した条件での実験が完成しました。一度 exp11.psyexp を保存して実行してみましょう。常にアイテムが 15 個提示されてしまいますが、アイテムの位置や向きが試行毎に無作為に変化していることが確認できます。これで残りはアイテムの個数を numItems パラメータの値に従って変化させるだけです。

チェックリスト

- リストの要素を無作為に並べ替えることができる。

- m 個の要素を持つリストから、n 個の要素 (m>n) を重複なく無作為に抽出することができる。

11.7 アイテムの個数を可変にしよう

いよいよ最終段階、numItems の値に従って試行毎にアイテム数を変化させる問題に取り組みましょう。いろいろな方法が考えられるのですが、ここでは簡単に実現できる「スクリーン外にアイテムを配置する」という方法を紹介します。今まで自分で実験を作成していて、height ではスクリーン上端の Y 座標が 0.5 なのに「**[位置 [x, y] $]** に [0.0, 1.0] などと指定して、刺激がスクリーンに描画されずに困ったことはないでしょうか。スクリーンの上下左右の限界からはみ出た位置を指定してもエラーにならないせいでこういった困ったことが生じるのですが、今回はこれがエラーにならない事を逆手に取ります。 **[実験開始時]** 入力済みのコードのうち、アイテムの位置を決定する処理だけを抜き出してみましょう。

```
for i in range(15):
    pos[i] = poslist[i]
```

ここへ if 文を追加して、i が numItems 未満の時は上記と同様の処理、i が numItems 以上の時はスクリーンの描画範囲を超えた位置を設定する処理を行うように変更します。今回の刺激はサイズが (0.03, 0.03) で、単位が height のためスクリーン上端の Y 座標が 0.5 ですから、[0.0, 1.0] であれば刺激全体がスクリーンの外になります。

```
for i in range(15):
    if i<numItems:
        pos[i] = poslist[i]
    else:
        pos[i] = [0.0, 1.0]
```

if 文の条件式が i<numItems であって、i<=numItems ではないことに注意してください。Python においてリストのインデックスは 0 から数えますので、5 個のアイテムを表示するときにはインデックス 0、1、2、3、4 の合計 5 個のアイテムに poslist の値を設定する必要があります。同様に、n 個のアイテムを表示するためにはインデックス 0 から n-1 までのアイテムに poslist の値を設定しなければいけません。if 文の条件式が i<=numItems だと、0 から numItems までの numItems+1 個のアイテムに poslist の値が設定されてしまいます。

なお、アイテムを描画しないようにさせるには、今回のようにスクリーンの描画範囲外の位置を指定するという方法の他にも、**[不透明度 $]** を 0.0 にして完全な透明にしてしまうという方法もあります。ただし、透明化する方法の場合は、あくまで描画されていないだけで PsychoPy にとってはその位置に刺激があると認識されますので、第 9 章 で紹介した contains( ) や overlaps( ) を使う時に注意する必要があります。刺激がそこに存在していないように見えるのに、マウスカーソルが「刺激の上に重なっている」と判定されてしまうなどの恐れがあるからです。もっとも、この問題ですら「実験参加者がスクリーン上のある領域にマウスカーソルを置いているか否か、参加者に悟られないように記録する」という用途にも使えますので、一概に不備だとは言えません。こういった一見不備に思える現象を積極的に利用することによって、Builder で実現できる実験の幅は飛躍的に広がります。ぜひ、いろいろと工夫していただきたいと思います。

さて、上記のコードを trial ルーチンの **[実験開始時]** に組み込んだ、最終版のコードを以下に記します。pos[i] への代入部分が変化したことを確認してください。

```
delay = randchoice( (1.0, 1.5, 2.0) )
shuffle(poslist)
for i in range(15):
    ori[i] = 90*randint(0,4)
    if i<numItems:
        pos[i] = poslist[i]
    else:
        pos[i] = [0.0, 1.0]
    if i==0:
        imagefile[i] = firstItem
    else:
        imagefile[i] = otherItems
```

exp11.psyexp に上記の変更を加えたら、exp11.psyexp を保存して実行してみましょう。今度は試行毎に無作為な順番にアイテム数が 5 個、10 個、15 個と変化することを確認してください。十数試行ほどスクリーンに描画されたアイテム数をメモして Escape キーを押して実験を中断し、描画されたアイテム数と trial-by-trial 記録ファイルに出力された numItems の値が一致していることも確認しましょう。これで今回の目標はすべて達成できました。

最後に、後の分析でアイテム位置の情報が必要になった場合に備えて、アイテム位置を保持している変数 poslist の値を実験記録ファイルに出力する処理を付け加えておきましょう。独自の変数の値を出力する方法についてはすでに 第 7 章 で解説しましたので、出力自体はもう皆さん解説なしでできると思います。ただ、poslist は要素数 36 のリストである一方、後の分析で実際に必要となる可能性がある要素は実際にスクリーン上に提示された刺激の位置に対応する要素のみです。言い換えると、各試行で先頭から numItems 個の要素のみが必要です。何の工夫もせずに poslist を addData( ) メソッドに渡してしまうと、36 個全部が出力されてしまうため、分析時に不必要な値を除去しなければならず、非常に無駄です。必要な値だけを抜き出して出力するのが理想的です。

for 文を用いると、リストの先頭から numItems 個の要素を取り出したリストを作成するのは簡単です。例えば以下のコードのようにすれば変数 displayedPos に実際に提示に利用された位置をまとめることができるでしょう。

```
displayedPos = []
for i in range(numItems):
    displayedPos.append(poslist[i])
```

if 文や for 文の利用はプログラミングの基本中の基本なので、こういったコードがぱっと頭に浮かぶようにしっかりとこれらの文に慣れて欲しいと思います。しかし、今回の用途に関しては Python にスライスと呼ばれる非常に便利な機能がありますので、そちらもぜひ覚えて欲しいと思います。

スライスとは、リストやタプルなどのシーケンス型のデータから、連続する要素を抜き出す演算です。シーケンス型データが格納された変数 var に対して var[a : b] の書式で用い、インデックス a からインデックス b の間に含まれる要素を抜き出したリストを返します。a と b の間の記号は半角のコロンです。第 9 章 で用いたリストの例をもう一度使って解説しましょう。 図 11.8 例 1 をご覧ください。[100, 200, 300, 400, 500, 600] というリストを格納した変数 var があります。正のインデックスは、先頭から順番にそれぞれの要素の「前」に

あると考えます。var[1:4] と書くと、インデックス 1 からインデックス 4 までの間の要素を取り出すのですから、[200, 300, 400] が得られます。初心者の方によくある勘違いに、スライスを「a 番目の要素から b 番目の要素を抜き出す」と考えてしまうというものがあります。「var[1] が 200、var[4] が 500 ですから、var[1:4] は [200, 300, 400, 500] じゃないの？」というのがこの勘違いの典型です。飽くまで 4 というインデックスは 500 の前にあり、var[1:4] というスライスは「インデックス 1 からインデックス 4 までの間の要素を抜き出す」のですから、500 は含まれません。

リストから要素をひとつ取り出す時に負のインデックスを利用できたのと同様に、スライスでも負のインデックスを用いることができます (図 11.8 例 2)。正と負のインデックスを混ぜて使うこともできます。ただし、var[a:b] の a の方が b よりもリストの前方でなければいけません。 図 11.8 例 3 つめの例のように、a が省略された時には、先頭から抜き出されます。 図 11.8 例 4 のように b が省略された時には、末尾までを抜き出します。

このスライスを利用すれば、poslist から実験記録ファイルに出力すべき要素を抜き出したリストを簡単に作ることができます。poslist の先頭から numItems 個の要素を刺激提示に使ったのですから、poslist[:numItems] とすればよいだけです (コロンの前は省略している点に注意)。このリストを実際に実験記録ファイルに出力するコードを書くのは練習問題としましょう。

チェックリスト

- ルーチンに配置された視覚刺激コンポーネントをスクリーン上に描画させないようにすることができる。
- スライスを用いて、あるリストから連続する要素を抽出したリストを作り出すことができる。
- リストの先頭から要素を抽出する場合のスライスの省略記法を用いることができる。
- リストの末尾までの要素を抽出する場合のスライスの省略記法を用いることができる。

11.8 練習問題：透明化によるアイテム数変更と無作為な位置の調整をおこなおう

exp11.psyexp を改造して、この章の解説で出てきた二つのテクニックを実際に試してみてください。さらに、特にアイテム数が 15 個の時に、アイテムが無作為に配置されているというよりは整然と並んでいるように見えてしまうことを防ぐために、アイテムの位置を無作為に上下にずらす処理も追加してください。

- **[不透明度$]** を 0.0 にすることによって numItems 個のアイテムがスクリーンに描画されるようする。
- アイテムの位置を実験記録ファイルに出力するコードを完成させる。
- アイテムの位置を、変数 pos によって指定された位置から上下方向、左右方向ともに-0.01、-0.005、0.005 または 0.01 ずらす。ずらす量は試行毎、アイテム毎、方向毎に無作為に決定する。

11.9 この章のトピックス

11.9.1 numpy.ndarray 型について

これまでの章ではずっと、刺激の位置 (座標値) や大きさといった二次元の量を指定するためにリストを使用してきました。本書の用途のように静的な位置を表現するだけならリストで十分なのですが、座標値に対す

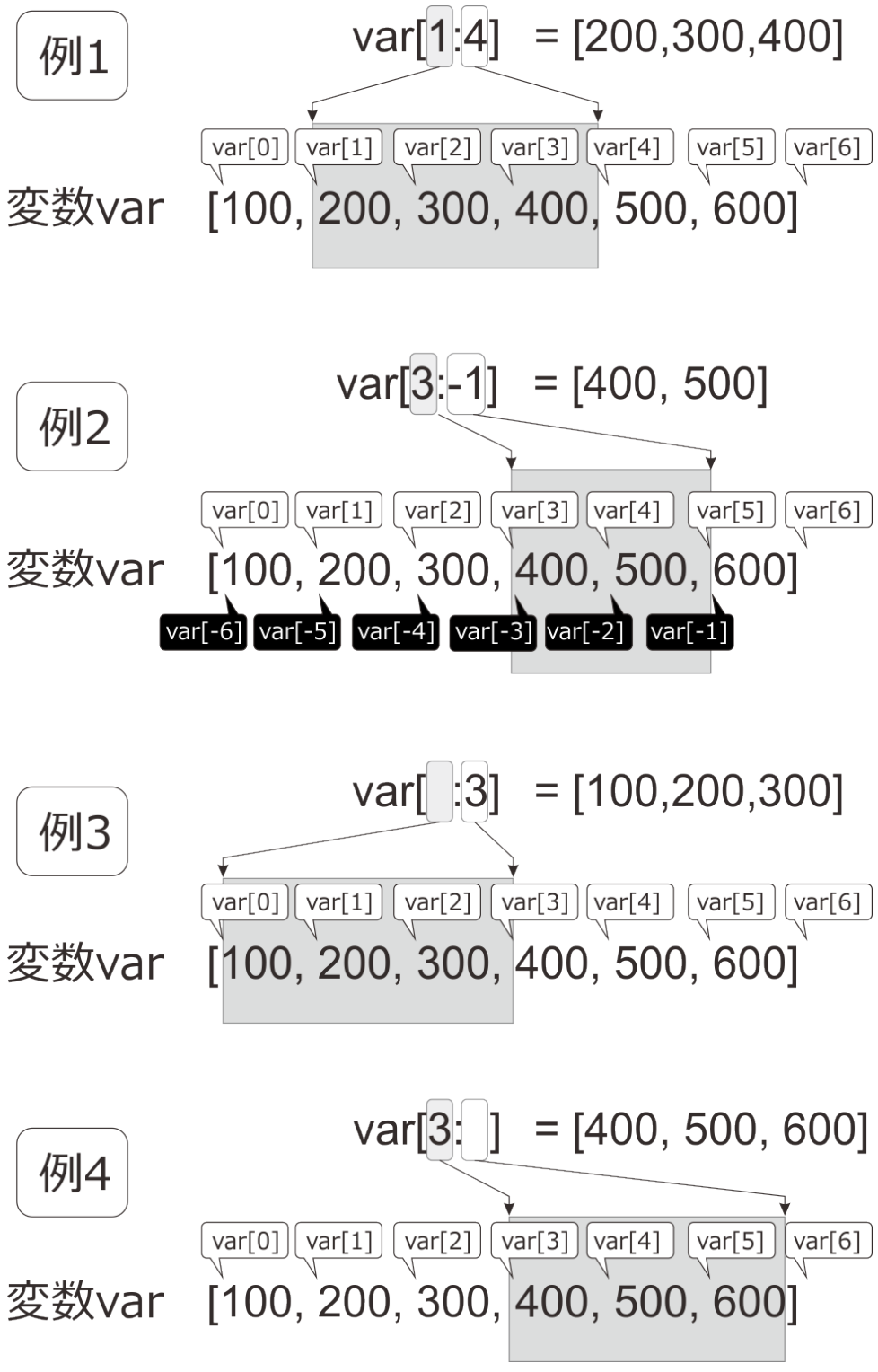


図 11.8　スライスによるリスト要素の抽出。var[a:b] と書くと、変数 var のインデックス a からインデックス b の間にある要素を抜き出します。a が省略されたときは先頭が、b が省略されたときは末尾が指定されたものとします。

る演算を行おうとするとリストは非常に不便です。例えば [5,3] という座標値を X 軸方向に 1、Y 軸方向に 2 移動させたい場合、ベクトルの演算をご存知の方は直感的には [5,3]+[1,2] と書きたくなるでしょう。しかし、Python におけるリストは数値以外にも文字列なども要素になり得ますので、[5,3]+[1,2] をベクトルの和と解釈することにすると要素に数値以外の値があったときに演算が定義できなくなってしまいます。そのようなわけで、かどうかはわかりませんが、Python はリスト同士に対する + 演算子はリストの結合として解釈します。つまり、[5,3]+[1,2]=[5,3,1,2] です。同様に、リストに対する数値の積は、ベクトルとスカラーの積ではなく、ベクトルの繰り返しとして解釈されます。[5,3] * 4 でしたら [5,3] を 4 回繰り返したリストである [5,3,5,3,5,3,5,3] が得られます。

これでは本格的なベクトル演算を行う時に不便で仕方がないので、Python では NumPy というパッケージが用意されています。NumPy を導入すると、直感的なベクトル演算が可能となります。NumPy における演算の基本となるのが numpy.ndarray 型のオブジェクトです。Builder ではリストなどのデータを numpy.ndarray に変換する numpy.asarray という関数が asarray という名前で利用できるように準備されています。asarray を使うと、先ほどのようなベクトル風の演算が可能になります。

```
asarray([5,3]) + asarray([1,2])  # 計算結果は array([6,5])
asarray([5,3]) * 4               # 計算結果は array([20,12])
```

これらの演算で得られた戻り値も numpy.ndarray 型のオブジェクトです。numpy.ndarray 型オブジェクトは、リストと同じように [ ] 演算子で要素を取り出したり、スライスを適用したり、len( ) で要素数を求めたりすることができます。ですから、[ ] や len( ) に関しては今まで学んできたリストと全く同等に使えます。しかし、+ 演算子や*演算子を適用した時の働きがリストと異なります。違いを十分に理解できれば asarray( ) を使って積極的に numpy.ndarray の機能を活用していただければ良いのですが、区別に自信がない場合は使わない方がよいでしょう。

11.9.2 range( ) オブジェクトについて

本文の説明では「range( ) 関数が返す range オブジェクトというものがわからない」という方が多いのではないかと思います。この range オブジェクトというのは Python3 から導入されたもので、旧バージョンである Python2 の range( ) は数値を並べたリストを返していました。つまり、本文では 0 から 14 まで並べたリストを得るときに list(range(15)) としましたが、Python2 の頃は単に range(15) と書くだけでよかったのです。これがなぜ Python3 で変更されたのかをお話すれば、range オブジェクトというもののイメージがもう少しはっきりするのではないかと思います。

例として、for 文と range() を使って 1000 万回の繰り返しをする場合を考えてみましょう。通常の心理実験では 1000 万回も繰り返すことはなさそうですが、分野によっては普通にあり得る回数です。Python2 のように range( ) がリストを返すとすると、0 から 9999999 までの 1000 万個の数値を並べたリストが作成されて for 文に渡されることになります。この巨大なリストはコンピュータのメモリ上に保持されますが、はっきり言って非効率的です。必要なのは今が何回目の繰り返しなのかという値だけなのに、1000 万個分ものメモリ領域が食いつぶされてしまうからです。

この問題を解決するのが Python3 の range オブジェクトです。range オブジェクトは 図 11.9 右のように

- 1 ずつ増やす
- 次は 7 を渡す

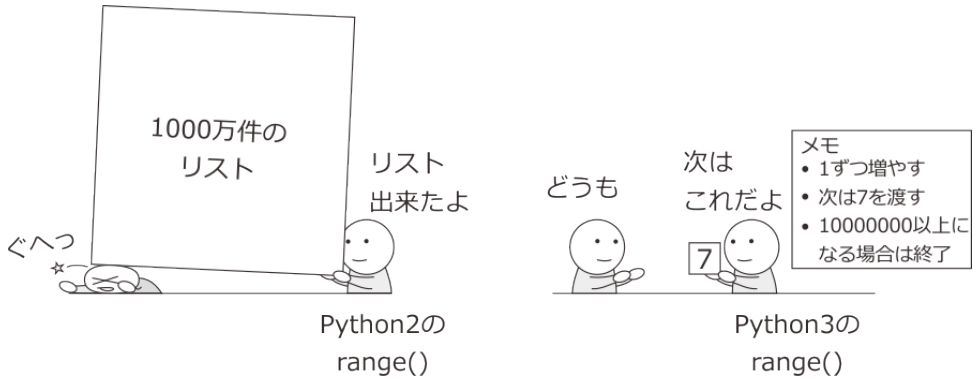


図 11.9　Python2 と 3 の range( ) の違い。

- 10000000 以上になる場合は終了

といったルールを保持していて、値を要求される度に値を生成します (厳密に言うとこれは range オブジェクトから作られるイテレータというオブジェクトの機能)。図 11.9 右の例では、7 を渡したら「1 ずつ増やす」というルールに従って「次は **8** を渡す」と更新しておくわけですね。これなら繰り返し回数が 1000 万回だろうが 1000 億回だろうがメモリへの負担が変わりません。圧倒的に効率がよいです。

range オブジェクトの弱点は、一連の数値をバラバラな順番で取り出す必要がある場合です。いきなり「1852 番目の数値が欲しいんだけど」と言われると、その数値がいくつであるのかをルールに従って延々と計算しなければいけません。このような場合はリストの方が優れています。Python3 でリストが必要な場合は、本文で紹介したように list( ) と組み合わせて list(range(x)) とします。

第 12 章

正答率や経過時間で終了する課題を作成しよう

12.1 この章の実験の概要

Builder の繰り返しは、基本的に回数を指定して使用します。しかし、心理学実験では事前に繰り返し回数が決まらない手続きもよく用いられます。「連続して 5 問正答するまで繰り返す」、「正答率が 80% に達するまで繰り返す」といった「ある条件を満たすまで繰り返す」といった手続き、「制限時間 1 分の中で出来る限りたくさん繰り返す」といった「一定時間経過するまで繰り返す」といったものです。この章ではまとまった「実験」を作るのではなく、これらのような特殊な繰り返し回数の課題を作成してみたいと思います。

12.2 繰り返しを中断しよう

本章で中心的な役割を果たすテクニックが「繰り返しの中断」です。第 7 章 で continueRoutine を使ってルーチンを抜ける方法を解説しましたが、これはあくまでルーチンを抜けるだけで、ループによる繰り返しが中断するわけではありません。ループが中断されないからこそ、 第 7 章 のように「次の試行 (=次の繰り返し) へ進む」ということができたわけです。ループに含まれる全てのルーチンに対して、開始したら直ちに continueRoutine=False を実行するようにすると、見た目上はルーチンを全く実行せずにループを進行させることができます。例えば task_completed という変数を False で初期化しておいて、「連続して 5 問正答するまで繰り返す」などの条件を満たしたら task_completed に True を代入するとします。そのうえで、ループ内のすべてのルーチンに Code コンポーネントを配置して **[フレーム毎]** に

```
if task_completed:
    continueRoutine = False
```

というコードを挿入しておけば、ひとたび task_completed が True になるとすべてのルーチンが即座に終了していずれループが終了します。これでも「条件を満たしたら繰り返しを終了する」ように見えるので構わないかも知れませんが、ループそのものは繰り返されているので、実験記録ファイルにはループの **[繰り返し回数]** や条件ファイルの条件数などから決定される繰り返し回数ぶんだけデータが出力されてしまいます。Builder の実験では一度繰り返しが始まってから繰り返し回数を増減させるのは難しいので、この方法で「条件を満たしたら繰り返しを終了する」を実現するためには「どれだけ長く続いてもこの回数内におさまるだろう」という繰り返し回数を設定しなければいけません。その回数分だけ実験記録ファイルに出力されてしまうのですから、ファイルサイズの無駄になりますし分析の時に邪魔です。

ではどうすればいいかというと、ループの実体である TrialHandler の finished というデータ属性を利用します。finished にはまだ繰り返し回数が残っていれば False、繰り返しを終えていれば True が代入されているのですが、finished に True を代入することで無理やり「繰り返しが終了した」状態にすることができます。従って、先ほどのコードは、ループの名前が trials とすると

```
if task_completed:
    trials.finished = True
```

と書けばよいということです。実際の使用例を見た方が早いと思いますので、さっそく作業に入りましょう。まずはストループ課題で、直近 10 試行を 90% 以上の正答率を達成したら終了する課題を組んでみます。実験開始前におこなう、指示内容を理解していることの確認のための練習試行などに適しています。

12.3 直近 10 試行の正答率が基準値に達したら終了するループを組もう

準備作業

- この実験のためのフォルダを作成して、その中に exp12a.psyexp という名前で新しい実験を保存する。
- 実験設定ダイアログの「スクリーン」タブの **[単位]** が height であることを確認する。
- 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。

trial ルーチン

- Text コンポーネントをひとつ配置し、以下のように設定する。
  - 「基本」タブの **[名前]** を `textTrial` にする。 **[文字列]** に **本試行をここで実行する** と入力する。

- abort ルーチン (作成する)
  - trial ルーチンの前に挿入する。
  - Text コンポーネントをひとつ配置し、以下のように設定する。
    - 「基本」タブの **[名前]** を `textAbort` に、 **[終了]** を空欄にする。 **[文字列]** に **実験者の指示に従ってください** と入力する。

practice ルーチン (作成する)

- abort ルーチンの前に挿入する。
- Text コンポーネントをひとつ配置し、以下のように設定する。
  - 「基本」タブの **[名前]** を `textPracticeStim` に、 **[開始]** を `0.5` 、 **[終了]** を空欄にする。 **[文字列]** に `$stim_word` と入力して「繰り返し毎に更新」にする。
  - 「外観」タブの **[前景色]** に `$stim_color` と入力し、「繰り返し毎に更新」にする。
  - 「書式」タブの **[文字の高さ]** を `0.1` にする。

  - Text コンポーネントをひとつ配置し、以下のように設定する。
    * 「基本」タブの **[名前]** を `textPracticeInst` に、 **[終了]** を空欄にする。 **[文字列]** に ← **文字が赤色** ↓ **文字が緑色** **文字が青色** → と入力する。
    * 「レイアウト」タブの **[位置 [x, y] $]** を `(0, -0.4)` にする。
    * 「書式」タブの **[文字の高さ]** を `0.03` にする。
  - Keyboard コンポーネントをひとつ配置し、以下のように設定する。
    * 「基本」タブの **[名前]** を `key_resp_practice` に、 **[開始]** を `0.5` 、 **[検出するキー $]** を `'left', 'down', 'right'` にする (キー名の順番は問わない)。
    * 「データ」タブの **[正答を記録]** をチェックして、 **[正答]** に `$correct_ans` と入力する。
  - Code コンポーネントをひとつ配置し、 **[名前]** を `codePractice` にする。コードは後で入力する。

- practices ループ (practie ルーチンのみを繰り返すように作成する)
  - **[Loop の種類]** を random にする。
  - **[繰り返し回数 $]** を 5 にする。
  - **[条件]** に `exp12a.xlsx` と入力する。
- exp12a.xlsx (条件ファイル)
  - 図 12.1 に示すように、表示する単語を定義する stim_word、文字色を定義する stim_color、正答キーを定義する correct_ans の 3 つの列からなる条件ファイルを作成する。内容は 図 12.1 とまったく同じでなくてもよいが、key_resp_practice が right、down、right の 3 つのキーを検出しているので、3 種類の語と色を用意すること。作成した条件ファイルを exp12a.xlsx の名前で exp12.psyexp と同じフォルダに保存する。

	A	B	C	D	E
1	stim_word	stim_color	correct_ans		
2	あか	red	left		
3	あか	lime	down		
4	みどり	lime	down		
5	みどり	blue	right		
6	あお	red	left		
7	あお	blue	right		
8					

図 12.1　ストループ課題のための条件ファイル

以上で準備は OK です。practice ルーチンで練習試行をおこなって、直近 10 試行の正答率が 0.8 以上であれ

ば trial ルーチンへ進みますが、30 試行 (条件ファイルに 6 条件× 5 回繰り返し) の間にこの基準に達しなかった場合は参加者が課題を理解していないなどの問題が生じている可能性があると考えて abort ルーチンで「実験者の指示に従ってください」と表示したいと思います。abort ルーチンにはルーチンを終了させるコンポーネントが配置せず、ESC キーで実験を中断しないといけないようにしてあります。

practice ルーチンに配置した Code コンポーネントですることをまとめると

1. 直近 10 試行の正答率を求めること
2. 直近 10 試行の正答率が基準を満たしていたらループを中断すること
3. 練習試行が基準を満たして終了したのか否かを次のルーチンへ伝えること

といったことが挙げられるでしょう。まず 1 は、複数回の繰り返しにわたって practice ルーチンでの正誤を保持しないといけないので、正誤を保持するリストを用意するといいでしょう。以下のように codePractice の **[実験開始時]** で空のリストを用意します。名前は practice_results としておきました。

```
practice_results = []
```

続いて、codePractice の **[Routine 終了時]** で反応の正誤を practice_results に追加します。反応の正誤を得るには Keyboard コンポーネントのデータ属性 corr を使えばよいのでした (表 6.3)。

```
practice_results.append(key_resp_practice.corr)
```

直近 10 試行の正答率を求めるには、 第 11 章 で出てきたスライスが早速役に立ちます。practice_results[-10:] とすれば practice_results の末尾から 10 個の要素を取り出せますので、sum( ) 関数で合計すれば直近 10 試行中で正答した試行数が得られます (正答なら 1、誤答なら 0 であることを思い出してください)。10 で割れば正答率となります。

正答率が求められたので、次は 2 のループの中断です。今回は「正答率が 0.8 以上」を基準としたいので、この条件が True になったら practices.finished = True を実行する if 文を書けばいいでしょう。ここまでまとめると、codePractice の **[Routine 終了時]** で

```
practice_results.append(key_resp_practice.corr)

if suum(practice_results[-10:])/10 >= 0.8:
    practices.finished = True
```

を実行すれば目標を達成できます。最後の 3. は、上記 if 文の条件式がそのまま「基準を満たして終了したか否か」の判定に使えますが、後から見てわかりやすいように変数に代入しておいた方が良いと思います。ここでは practice_complete という変数にしておきます。

```
practice_results.append(key_resp_practice.corr)

if suum(practice_results[-10:])/10 >= 0.8:
    practices.finished = True
    practice_completed = True
```

これだけでは、基準を満たして終了しなかったときに practice_completed に値が代入されていない状態になるので、どこかで False を代入しておく必要があります。上記のコードの if 文に else 節をつけてそこで代入してもいいのですが、基準を満たすまで何度も False を代入し続けるのも無駄な作業なので、 **[実験開始時]** に

```
practice_results = []
practice_completed = False
```

と代入しておくとよいでしょう。これで practiceCode でするべき 3 つの作業がすべてできました。

後はこの practice_completed を後のルーチンでどう活用するかですが、abort ルーチンの「Routine の設定」をクリックしてプロパティ設定ダイアログを開き、「Flow」タブの **[条件に合致する場合はスキップ... $]** に practice_completed と入力しましょう。基準を満たして practices ループが終了したなら、practice_completed が True になっているので、abort ルーチンがスキップされて trial ルーチンへ処理が進みます。もし基準を満たさずに practices ループが終了していれば、practice_completed が False なので abort ルーチンが実行されます。先述した通り、abort ルーチンに進むと参加者は先へ進めなくなりますので、実験者が ESC キーを押して実験を中断して、改めて課題を説明するなり実験を中止するなりすることになるでしょう。exp12a.psyexp を実行してみて、基準を満たせば「本試行をここで実行する」と画面に表示されて終了すること、わざと間違え続けて基準を満たさずに終われば「実験者の指示に従ってください」と表示されて ESC キーで中断しないといけなくなることを確認してください。

チェックリスト

- ループ内で反応の正誤を順番にならべたリストを作ることができる。
- リストの末尾から指定された個数の値を取り出せる。
- ループを中断することができる。

12.4 練習試行で基準を満たさなければ自動的に実験が中断されるようにしよう

前節では練習試行の成績が基準に達しなかったときに通常のキー押し等で終了しないメッセージを表示して、実験者が実験の中断操作をすることを想定していましたが、自動的に実験が終了した方が望ましいケースもあるでしょう。そのような場合、「基準に達しなかったらフローの残りをすべて実行しない」という処理が必要になります。**[条件に合致する場合はスキップ... $]** を後続の全てのルーチンに設定するという方法でもできなくはありませんが、後続のルーチンがたくさんある場合はかなり面倒ですし、ループがある場合はループも中断させないと実験記録ファイルに出力されてしまいます。フローに条件分岐の機能があればいいのですが、残念ながら現在のバージョンの BUilder にはありません。

そこで注目したいのが、「 4.9: 動作確認のために一部の動作をスキップしよう 」で紹介した「ループの **[繰り返し回数]** を 0 にする」というテクニックです。これは「ループが 0 回実行される = 1 度も実行されない」ためループ内のルーチンがまとめてスキップされるというものでした。4.9 章 で紹介したときは、実験の動作確認の時に確認不要な部分をスキップするために、実行前に手作業で 0 に設定するという使い方をしたのですが、 **[繰り返し回数]** に変数を指定して実行時の条件に応じて変更してやろうというわけです。前節で作成した exp12a.psyexp で引き続き作業しましょう。

- trials ループ (trial ルーチンのみを繰り返すように挿入する)

  - **[繰り返し回数 $]** に num_repeats_trials と入力する

- abort ルーチン
  - textAbort の「基本」タブの **[終了]** に 3.0 と入力する。 **[文字列]** に 「実験を中断します」 と入力する。

abort ルーチンでどういうメッセージを表示するかは、実験者が横についているのか、別室で待機しているのか、中断した場合にその理由をメッセージで参加者に伝えるか否かなど、状況によって異なると思いますので、ここでは簡潔に「実験を中断します」だけにしておきました。3 秒で自動的に終了するようにしましたが、参加者にキーを押させるなどの方法もあり得るでしょう。

さて、ここから本題ですが、practice ルーチンに配置してある codePractice の **[Routine 終了時]** のコードに以下のように if 文に num_repeats_trials = 1 という行を追加してください。

```
practice_results.append(key_resp_practice.corr)

if suum(practice_results[-10:])/10 >= 0.8:
    practices.finished = True
    practice_completed = True
    num_repeats_trials = 1
```

practice_completed の時と同様、基準を満たさなかったときに num_repeats_trials の値が未定義になってしまわないよう、 **[実験開始時]** に num_repeats_trials = 0 を追加しておきましょう。

```
practice_results = []
practice_completed = False
num_repeats_trials = 0
```

以上で作業は終了です。exp12a.psyexp を実行して、わざと間違え続けて基準を満たさずに終われば「実験を中断します」と表示されて、3 秒経過すると自動的に実験が中断される (=trial ルーチンが実行されない) ことを確認してください。この方法であれば、trials ループ内にルーチンがいくつあってもまとめてスキップすることができます。また、練習試行の成績に応じて 3 通り以上にフローを分岐するといったことも可能になります。応用範囲が広いので、ぜひ覚えておいてください。

チェックリスト

- 実験実行時に条件に応じてループの実行をスキップできる。
- ループを活用して、フローで条件分岐を実現できる。

12.5 グローバルクロックを利用して一定時間経過したら終了するループを組もう

続いてもう一つの例として、記憶課題などで、再生や再認を開始するまでの間に「1 分間計算問題を繰り返す」といった遅延課題を Builder で実現することを考えてみましょう。前節の練習課題の例と異なるのは、練習課題では参加者が反応したタイミングでループの中断判断をおこなえばいいのに対して、遅延課題では反応

を待っている期間であっても制限時間がきたら直ちにループ (と現在のルーチン) を中断しないといけない点です。

ルーチンの中断もループの中断もすでに解説しましたので、特に難しいことはないように思われるかもしれませんが、ひとつ問題があります。それは「課題を始めてから何秒経過したか？」を知る方法です。これまで時間というと Routine が開始してからの時間を表す内部変数 t が出てきましたが、t はルーチンを繰り返すたびに 0 にリセットされてしまうので、「ループが始まってから何秒経過したか」を測ることはできません。そこで登場するのが Builder のグローバルクロックです。これは実験開始時に 0 に初期化されるストップウォッチのようなもので、globalClock という Builder の内部変数に格納されています。その実体は psychopy.core.Clock というクラスのオブジェクトで、getTime( ) というメソッドを用いて現在の経過時間を取得することができます。さっそく実験を作成して確かめてみましょう。

準備作業

- この実験のためのフォルダを作成して、その中に exp12b.psyexp という名前で新しい実験を保存する。
- 実験設定ダイアログの「スクリーン」タブの **[単位]** が height であることを確認する。
- 実験設定ダイアログの「入力」タブの **[キーボードバックエンド]** が PsychToolbox か確認する。

delay_task ルーチン (trial ルーチンの名前を変更する)

- Code コンポーネントをひとつ配置し、codeDelayTask という名前にする。

–Text コンポーネントをひとつ配置し、以下のように設定する。

* 「基本」タブの **[名前]** を textDelayTaskQuestion にする。 **[開始]** を 0.1 にして、 **[文字列]** に $delay_task_question と入力する。
* 「書式」タブの **[文字の高さ]** を 0.1 にする。

–Text コンポーネントをもうひとつ配置し、以下のように設定する。

* 「基本」タブの **[名前]** を textDelayTaskInst にする。 **[文字列]** に「←割り切れない　割り切れる →」と入力する。
* 「レイアウト」タブの **[位置 [x, y] $]** を (0, -0.4) にする。
* 「書式」タブの **[文字の高さ]** を 0.05 にする (初期値でそうなっている)。

–Keyboard コンポーネントをひとつ配置し、以下のように設定する。

* 「基本」タブの **[名前]** を key_resp_DelayTask にする。 **[開始]** を 0.1 にして、 **[検出するキー $]** を'left', 'right' にする。
* 「基本」タブの **[Routine を終了]** がチェックされていることを確認する。
* 「データ」タブの **[正答を記録]** をチェックして、 **[正答]** に $delay_task_correct_ans と入力する。

delay_tasks ループ (delay_task ルーチンを繰り返すように挿入する)

- **[繰り返し回数 $]** を 12 にする。

  - **[条件]** に exp12b.xlsx と入力する。

- exp12b.xlsx (exp12b.psyexp と同じフォルダに作成する)
  - delay_task_question と delay_task_correct_ans という 2 列のパラメータを持つ条件ファイルを作成する。
  - delay_task_question には 78 ÷ 7 のような 2 桁の整数を 1 桁の整数 (0, 1 は除く) で割る式を入力し、delay_task_correct_ans にはその式が割り切れないなら left、割り切れるなら right と入力する。
  - 少なくとも 10 条件は作成する。

以上で準備終了です。あとは Code コンポーネントに入力するコードを考えていきましょう。まず、遅延課題を開始してからの経過時間がわからないことにはどうしようもありませんから、開始時刻を測らないといけません。これには globalClock.getTime() を使うとして、Code コンポーネントのどこへコードを入力すればいいでしょうか。delay_tasks ループによる繰り返し全体が「遅延課題」を構成しているので、「delay_tasks ループ開始時」に入力するというのが正解ですが、残念ながら Code コンポーネントには「ループ開始時」というタブがありません。候補としては

1. delay_tasks ループに入る直前の **[Routine 終了時]** に入力する
2. delay_tasks ループに入って最初の **[Routine 開始時]** に **初回に一度だけ実行されるように** 入力する

の 2 つが考えられます。1. の方法はここまでに解説したテクニックの範囲で十分にできますし、「遅延課題のための準備をその前の課題で行わなければならない」点が「この遅延課題を別の実験に使いまわせるようにしたい」というところまで考えると望ましくないので、ここでは 2. の方法を考えます。

「**[Routine 開始時]** で初回に一度だけ実行される」ようにするには、「実行済みか否か」を保持する変数を利用します。こういった変数をフラグと呼びます。フラグが False であれば「まだ実行されていない」と判断して時刻の測定を行うと同時にフラグを True にします。以後、ふたたび **[Routine 開始時]** になってもすでにフラグが False ではないので時刻の測定は実行されないというわけです。ここでは「遅延課題の実行中である」ということで delay_task_running という名前の変数をフラグにしましょう。以下のように codeDelayTask の **[Routine 開始時]** に入力します。

```
if not delay_task_running:
    delay_task_start = globalClock.getTime()
    delay_task_running = True
```

遅延課題開始時刻を保持する変数は delay_task_start という名前にしておきました。このままだと初回実行時には delay_task_running という変数が一度も値を代入されないまま if 文で参照されてしまうため、**[実験開始時]** に

```
delay_task_running = False
```

と入力しておきましょう。当然、実験開始時には遅延課題は開始していないので False を代入しています。続いて **[フレーム毎]** ですが、ここでは 1 分間経過したらルーチンの中断とループの中断を一気に行う必要があります。「1 分間経過したら」の判断は、globalClock.getTime() から delay_task_start を引けば現在の経過時間

が得られるので、この値が 60 を超えたか否かを条件とする if 文を書けばよいでしょう。ルーチンの中断は continueRoutine=False 、 ループの中断は delay_tasks.finished=True ですから、以下のような文になります。

```
if globalClock.getTime() - delay_task_start > 60.0:
    continueRoutine = False
    delay_tasks.finished = True
    delay_task_running = False
```

最後の delay_task_running=False は「遅延課題を終了する」という処理自体には不要ですが、もしこの遅延課題を別のループの中に入れて繰り返し実行するなら (というか繰り返し使う方が一般的でしょう)、delay_task_running を False に戻しておかないと次の繰り返しの時に「遅延課題が開始された時刻を測る」という処理ができません。次の繰り返しまでのどこかで False にすれば問題ないのですが、実際に遅延課題が終了する直前のこのタイミングで False にするのがスマートです。

以上でコードは完成です。exp12b.psyexp を実行して、画面に表示される式が割り切れるか割り切れないかカーソルキーを使って反応すると次々と問題が表示されることと、時間が経過したらキーを押さなくても直ちに終了することを確認してください。

なお、ここでは一定時間で終了するループを組むテクニックを学ぶことが目的だったので、exp12b.xlsx の条件数は「少なくとも 10 条件作成してください」とだけしか指定しませんでしたが、実際の実験では「遅延課題の長さを何秒にするか」、「同じ問題が繰り返し表示されても構わないか」といった事に注意して条件数を決定する必要があります。

最も避けなければいけないことは、条件数と繰り返し数が足りなくて、遅延課題の時間が終了する前に繰り返しが完了してしまうことです。exp12b.psyexp の場合、Keyboard コンポーネントの **[開始]** が 0.5 秒に設定されているため、参加者が全力でキーを連打しても絶対にループ 1 回に 0.5 秒かかります。そんなことをする参加者はいないだろうと思うかもしれませんが、ずっと付き添っていない実験ならあり得ないとは言えません。ループ 1 回に 0.5 秒以上かかるということは、1 分 (=60 秒) ÷ 0.5 秒で 120 問以上の問題があれば、最短の反応時間であっても遅延時間内に終わってしまうことはありません。exp12b.psyexp では delay_tasks ループの **[繰り返し]** を 12 に設定しているので、exp12b.xlsx に 10 条件あれば 12 × 10=120 問となりギリギリこの条件を満たすことができます。同じ問題が出現してはいけない場合は、条件ファイルに定義された条件数だけで確実に遅延時間以上かかるようにしないといけません。今回の場合なら 120 条件以上です。実際には最短の反応時間で反応し続けるなんて不可能でしょうからもう少し問題数が少なくても大丈夫でしょうが、確実に 1 分以上かかる問題数を確保しておいた方がいいと個人的に思います。

チェックリスト

- ループを中断することができる。
- 実験開始からの経過時間を取得することができる。
- 開始から一定時間経過したら終了するループを組むことができる。

12.6 残り時間を表示しよう

この節では、前節の遅延課題を題材として、残り時間の表示をしてみたいと思います。時間の表示は 第 5 章 でフレーム毎のアニメーションをする際にも少し触れたのですが、その時は小数点以下すごい桁数が表示されてしまって非常に見難いものでした。この問題を解決する方法を学ぶのが目的です。

文字列オブジェクトには format( ) というメソッドがあり、このメソッドを用いると引数として与えられた他の文字列や数値などを書式指定しながら埋め込むことができます。以下に format() の例を示します。

```
' 平均反応時間:{} 正答率:{}'.format(mean_rt, n_correct)
```

文字列の中に{}という部分が 2 か所あることに注目してください。format() は、引数として与えられた値を文字列中の{}の場所へ順番に埋め込んでいきます。いま、引数として与えている変数 mean_rt の値が 1218.7、n_correct の値が 31 ならば、このメソッドの実行結果は

```
' 平均反応時間:1218.7 正答率:31'
```

という文字列になります。format( ) はただ引数の値を埋め込むだけでなく、{}のなかに書式指定文字列と呼ばれる文字を書き込むことによって、埋め込まれた後の文字列を細やかに指定することができます。非常に多くの機能があるのですが、ここで紹介したいのは数値を埋め込むときの桁数指定です。以下に小数点以下 3 桁まで表示するよう指定する例を示します。

```
' 実験開始から{:.3f}秒'.format(t)
```

ちょっと難しいですが、{}内の : が「これ以降は書式指定」ということを表す記号だと思ってください。 : の後ろの .3f というのが書式指定で、f は小数として出力することを指定してします。f の前の.3 が小数点以下 3 桁を出力するという指定です。. の前に整数を書くと全体の桁数の指定となり、さらに + 記号をつけると符号つきで出力されます。文章で説明してもなかなかわかりにくいと思いますので、具体例を 図 12.2 に示します。

図 12.2　数値の書式指定の例。変数 p には 3.14159264359 という値が代入されているものとします。

書式指定には小数を指定する f の他にも、10 進数の整数を指定する d や、16 進数の整数を指定する x などがあります。小数を 10 進数整数や 16 進整数で出力しようとするとエラーで実験が停止してしまうのでご注意ください。小数の値を整数で出力する場合は以下のように int() を使って整数に変換するか、小数点以下の桁数として 0 を指定すればよいでしょう。

```
'実験開始から{:d}秒'.format(int(t))  # int で整数に変換
'実験開始から{:.0f}秒'.format(t)     # 小数点以下の桁数として 0 を指定
```

ここまで数値を埋め込む例ばかり挙げてきまし t が、format( ) は教示文や刺激文へ単語の埋め込む場合などにも便利です。以下の例では実験情報ダイアログの値 (＝文字列) を埋め込んでいます。

```
'あなたの参加者 ID は{}です'.format(expInfo['participant'])
```

埋め込み後の文字列に { や } の記号を出力したい場合は {{ や }} といった具合に 2 つ並べて書きます。以下の例で、変数 id の値が 712 なら '{参加者 ID}: 712' という文字列が得られます。

```
'{{参加者 ID}}: {}'.format(id)
```

第 6 章 では + 演算子と str( ) 関数を使用する方法を紹介しましたが、文字列に埋め込む値が多数ある場合は format() を使った方が簡潔に記述できるので、ぜひ活用してください。

それでは、前節の exp12b.psyexp に残り時間を表示してみましょう。

- delay_task ルーチン
  - Text コンポーネントをひとつ配置して以下のように設定する。
    - 「基本」タブの **[名前]** を　textDelayTaskRemaining にして、 **[終了]** を空欄にする。
    - 「基本」タブの **[文字列]** を $' 残り{:3.1f}秒'.format(delay_task_remaining_time) にして、「フレーム毎に更新」にする。
    - 「レイアウト」タブの **[位置 [x, y] $]** を (0, 0.3) にする。
  - codeDelayTask の **[フレーム毎]** の先頭に以下のように delay_task_remaining_time への代入文を挿入し、if 文の条件節を変更する。

```
delay_task_remaining_time = delay_task_start + 60 - globalClock.getTime()
if delay_task_remaining_time < 0.0:
    # 以下の行はそのままでよい
```

変更前は開始からの経過時刻を **[フレーム毎]** で計算していましたが、残り時間を表示したいので delay_task_start+60 として終了時刻を求め、そこから globalClock.getTime() で得た現在の時刻を引いています。この式を textDelayTaskRemaining の **[文字列]** に伴う format( ) の中と if 文の条件説の両方に書くのは長ったらしいですし、変更する必要が生じた時 (例えば遅延時間を変更するとき) などに 2 箇所修正するのも面倒ですので、delay_task_remaining_time という変数に代入しています。delay_task_remaining_time という変数名自体長ったらしいじゃないかと怒られるかもしれませんが、これは次のことを考えてわざとこのような名前にしてあります。

あと、ひとつ注意すべき点として、codeDelayTask が textDelayTaskRemaining より先に実行されないと、delay_task_remaining_time という変数が定義されていなくてエラーになってしまいます。ルーチン内でのコンポーネントの順序を間違えないようにしましょう (今回は解説通りの手順で作業していれば適

切な順番になるはず)。そういった順序関係を気にしたくない人は、codeDelayTask の **[実験開始時]** で delay_task_remaining_time = 60 などと適当な値を代入して変数を用意しておくと良いでしょう。

以上で作業は終了です。実験を実行して、残り時間が表示されることを確認してください。

チェックリスト

- 小数点以下の桁数を指定して数値を文字列に埋め込める。
- ひとつの文字列に複数の数値や文字列を位置を指定して埋め込める。

12.7 テンプレートにしてみよう

前節で「delay_task_remaining_time という長ったらしい名前をつけたのは次のことを考えて…」と書きましたが、本章の締めくくりとしてやっておきたいことは「テンプレート化」です。「3.12.9: 独自のルーチンテンプレートを登録する方法 (上級)」で少し触れたように、Builder で新しいルーチンを追加するときに選択できるテンプレートは独自に追加することができます。十分に活用するにはよく考えてテンプレートを設計する必要があるので 第 3 章 では詳しく触れなかったのですが、そろそろ解説しても問題ないでしょう。前節までに作成した遅延課題のルーチンを使って、実際にテンプレートを作成してみます。

テンプレートは通常の psyexp ファイルで、Builder が起動するときにユーザー設定フォルダの routine_templates というサブフォルダ内に置かれていればテンプレートとして登録されます。ですから、作業としては exp12b.psyexp をこのフォルダへコピーするだけです。問題は「ユーザー設定フォルダ」とはどこかということですが、Windows ではアプリケーション固有のデータを保存するフォルダ中にある psychopy3 というフォルダが該当します。この「アプリケーション固有のデータを保存するフォルダ」は Windows の APPDATA という環境変数で参照することができるので、ファイルエクスプローラのアドレスバーに %APPDATA% と入力して Enter キーを押せば開くことができます (図 12.3)。一度でもその Windows 上で PsychoPy のアプリケーション (Builder, Coder, Runner) を起動したことがあれば、そのフォルダの中に psychopy3 というフォルダが見つかるはずです。

図 12.3　Windows で PsychoPy のユーザー設定フォルダを見つける方法

MacOS や Linux では、PsychoPy のアプリケーションを一度でも立ち上げたことがあれば、ユーザーのホームディレクトリに.psychopy3 という隠しフォルダが作成されているはずです。これがユーザー設定フォルダです。隠しフォルダなので標準の状態では見えませんが、MacOS では Finder で Command + Shift + . (ピリオ

ド) キーを同時押しすると名前が . から始まる隠しフォルダおよび隠しファイルを表示することができます。Linux ではウィンドウマネージャによって操作が違うでしょうから自分が使用している環境での表示方法は各自で調べてください。Linux を使う人なら shell での操作に抵抗がない人が多いでしょうから、shell で操作した方がいいかも知れません。

ユーザー設定フォルダを開いたら、その中に routine_templates というフォルダがあるか確認し、もしなければ作成してください。そして、本章で作成した exp12b.psyexp を routine_temples フォルダにコピーして、PsychoPy を一旦すべて閉じてから起動してください。以後、Builder で新しいルーチンを追加しようとすると、 図 12.4 のようにテンプレートの中に exp12b というカテゴリができていて、その中に delay_task というテンプレートが追加されているはずです。このテンプレートを選んでルーチンを追加すれば、exp12b.psyexp で作った delay ルーチンが追加されます。

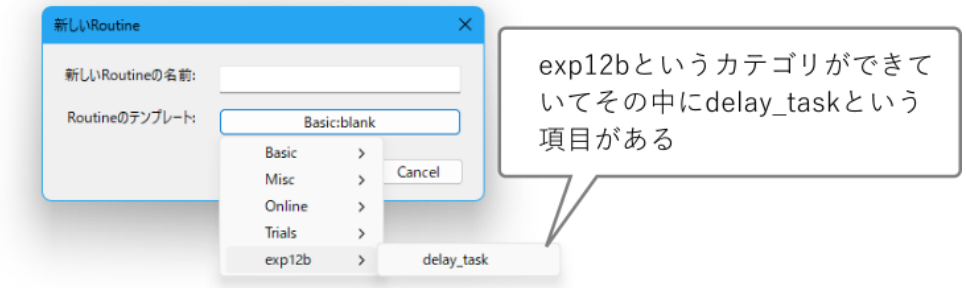


図 12.4　routine-templates フォルダに追加した psyexp ファイルと同名のカテゴリが追加されていて、そのファイルに含まれていたルーチンがテンプレートとして表示される。

この delay_task ルーチンが適切に機能するためには、exp12b.psyexp でそうしたように delay_tasks というループで囲む必要があります。delay_task ルーチン内に配置した Code コンポーネントに delay_tasks.finished=True というコードが含まれているので、ループの名前は delay_tasks でなければいけません。ループの名前を変えたければ、コードを修正する必要があります。delay_tasks ループで **[条件]** に指定する条件ファイルも用意しなければいけませんし、Code コンポーネントで使用している delay_task_start や delay_task_running といった変数を他のルーチンで使用しないよう注意しなければいけません。こういったことを考えると、コードなどを駆使した複雑なルーチンは作るのが面倒なので、繰り返し使うのならどんどんテンプレートしていきたいところですが、「このテンプレートを使う時には何を用意しないといけないか、何をしてはいけないか」を理解しておく必要があります。先ほど「十分に活用するにはよく考えてテンプレートを設計する必要がある」と書いたのはこういった点を指しています。

実は、exp12b.psyexp の作成手順を執筆していた時、テンプレート化を考えて決めたことがいくつかあります。まず、変数名として delay_task_remaining_time などという「長ったらしい名前」をわざわざ選んだのは、他のルーチンで使用する名前と重複しにくくするためです。配置するコンポーネントの名称や、条件ファイルの列名などにも、テンプレートの名前にちなんで delay_task とか delayTask といった文字列を組み込んでいます。また、遅延課題の測定時刻の測定は「ループの開始直前の **[Routine 終了時]** 」におこなった方がフラグ変数が不要となってシンプルになるのですが、それではテンプレート内にコードを含めることができません。「ループに入って最初の **[Routine 開始時]** に初回だけ実行する」方法であれば delay_task ルーチンに配置する Code コンポーネント内ですべて完結するので、テンプレート化したときに考えないといけない点が減ります。

ルーチンの独立性を高めることにこだわるなら、exp12b.psyexp で条件ファイルで定義することにしていた計算問題と回答を 第 11 章 のような方法で無作為に生成するという事も考えられます (「 12.8.1: 遅延課題の計算問題を実行時に生成する 」)。どの実験でも同じ計算問題を遅延課題として使用するならそれも良いでしょう

が、exp12b.psyexp のように条件ファイルとして問題を切り離しておけば、計算問題以外のカーソルキーの左右で反応できる課題 (例えば単語が生物か無生物かを判断する課題など) にも使いまわすことができます。どちらが優れているというわけではないので、自分が使いやすいようにカスタマイズしてください。

さて、テンプレート化について、あといくつか確認してから本節を終えたいと思います。**ユーザー設定フォルダ内の routine_tenplates フォルダにコピーした方の exp12b.xlsx** で以下の作業をしてください。

- exp12b.psyexp の名前を 遅延課題.psyexp に変更する。
- Builder で 遅延課題.psyexp を開く。
- delay_task ルーチンの「Routine の設定」をクリックして設定ダイアログを開き、「基本」タブの **[説明]** に以下のように入力する。

```
遅延課題のテンプレートです。delay_task_question と delay_task_correct_ans という 2 つのパラメータを定義した条件ファイルを指定した delay_tasks というループで囲んで使用します。

delay_task_question は問題文、 delay_task_correct_ans は正答となるキーを right、left のいずれかで指定します。

以下の変数を Code コンポーネントで使用しているので、このテンプレートを使用する実験では他の場所で使用しないでください。
- delay_task_running
- delay_task_start
- delay_task_remaining_time
```

- delay_task_feedback ルーチン (新たに作成し、delay_task ルーチンの後ろに挿入する)
  - Code コンポーネントをひとつ置き、 **[名前]** を codeDelayTaskFeedback にする。
  - Text コンポーネントをひとつ置き、以下のように設定する
    - 「基本」タブの **[名前]** を textDelayTaskFeedback にする。 **[終了]** が 1.0 であることを確認する (初期状態でそうなっている)。
    - 「基本」タブの **[文字列]** を $delay_task_feedback_msg として「繰り返し毎に変更」にする。
  - codeDelayTaskFeedback、textDelayTaskFeedback の順番に実行されるよう並んでいることを確認する (ここまで手順どおりに作業していればそうなっているはずである)。そうしないとルーチン開始時に Text コンポーネントで delay_task_feedback_msg が定義されていないと言われてエラーになる。
  - delay_task ルーチンから textDelayTaskRemaining をコピーして、delay_task_feedback ルーチンに貼り付ける。 **[名前]** は textDelayTaskRemaining_2 でよい。以下のように設定する。
    - 「基本」タブの **[終了]** 1 にする (textDelayTaskFeedback と同時に終了するようにする)。
  - 「Routine の設定」をクリックして設定ダイアログを開き、以下の通り設定する。

* 「flow」タブの **[条件に合致する場合はスキップ... $]** に delay_tasks.finished と入力する。
* 「基本」タブの **[説明]** に以下のように入力する。

遅延課題のテンプレート `delay_task` **と組み合わせて使用するフィードバック画面です。単独で使用しないでください。**

以下の変数を `Code` **コンポーネントで使用しているので、このテンプレートを使用する実験では他の場所で使用しないでください。**
- `delay_task_feedback_msg`

続いて delay_task_feedback ルーチン の codeDelayTaskFeedback にコード入力しましょう。まず **[Routine 開始時]** に以下のコードを入力します。

```
if key_resp_DelayTask.corr:
    delay_task_feedback_msg = '正解'
else:
    delay_task_feedback_msg = '不正解'
```

[フレーム毎] に以下のコードを入力します。 delay_task ルーチンに配置されている codeDelayTask の **[フレーム毎]** に書いてあるコードと同じなので、コピー&ペーストするとよいでしょう。

以上で作業は終了です。遅延課題.psyexp を保存して PsychoPy をすべて終了し、もう一度 PsychoPy を起動してください。そして Builder で新しいルーチンを挿入しようとしたら、 図 12.5 のように、「遅延課題」と言うカテゴリができていて delay_task と delay_task_feedback というテンプレートが含まれているはずです。以上のことから、routine-templates フォルダに保存した psyexp ファイル名がテンプレートのカテゴリとして利用されること、カテゴリ名 (=psyexp ファイル名) は日本語では問題ないこと、psyexp ファイルに含まれているルーチンはそのファイル名のカテゴリのテンプレートになることがわかります。

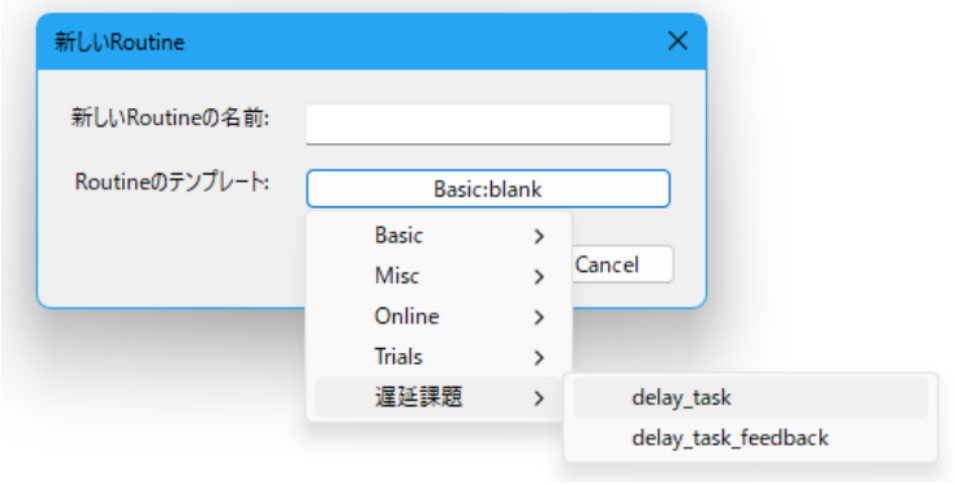


図 12.5　routine-templates フォルダに追加した psyexp ファイルと同名のカテゴリが追加されていて、そのファイルに含まれていたルーチンがテンプレートとして表示される。

ルーチン名に日本語などの非 ASCII 文字を使用することは引き続きお勧めしませんが、筆者がバージョン 2024.2.5 + Python 3.10 で試した限り、テンプレートとして使用する psyexp ファイルのルーチンに非 ASCII

の文字を使ってもテンプレート名として問題なく機能しました。テンプレートを使って新しいルーチンを挿入するときには名前をつけないといけませんが、その際に非 ASCII 文字を使わないように心がければ、テンプレート名に日本語を使っても問題ないかもしれません。

delay_task または delay_task_feedback テンプレートを使って新しいルーチンを挿入したら、「Routine の設定」をクリックして設定ダイアログを開き、「基本」タブの **[説明]** に先ほど入力した注意書きが入力されていることを確認してください。「**[説明]** にテンプレートを使用する際の注意事項を書く」ということを研究室内で徹底しておけば、かなり便利にテンプレートを使えるのではないかと思います。

チェックリスト

- PsychoPy のユーザー設定フォルダを開くことができる。
- Builder にルーチンのテンプレートを追加することができる。
- ルーチンテンプレートのカテゴリ名を設定することができる。
- 複数のルーチンテンプレートをカテゴリにまとめることができる。
- テンプレートの **[説明]** にテンプレート使用の際の注意点を記入しておくことができる。

12.8 この章のトピックス

12.8.1 遅延課題の計算問題を実行時に生成する

遅延課題を条件ファイルに分けずにテンプレートに組み込んでしまうには、 **[実験開始時]** などに以下のような問題と正反応のペアを並べたリストを用意すると良いでしょう。

重複を許さないなら shuffle( ) で順序を無作為化し、delay_tasks.thisN を使って順番に問題と正反応のペアを取り出せばいですし、重複しても良いなら randchoice( ) でペアを取り出すという方法もあります。ただ、randchoice( ) だと本当に無作為になってしまうので、可能性としては極めて低いもののひとつの問題が極端に多い回数選出されてしまう可能性があります。それでは困ると言う場合は、「Python においてリストと正の整数の積はリストの要素をその回数だけ繰り返したリストを返すことを利用して

として各問題を 5 回含むリストを作成し、shuffe( ) して順番に取り出せばよいでしょう。

問題と正反応のペアが大量に必要な場合、それらをすべてコードとして記入するのも大変ですので、実行時に生成することも考えてみます。numpy を使った方が速いかも知れませんが、ここでは Builder が標準で import していないモジュールは使わないことにします。まず、被除数を 10 から 99、除数を 2 から 9 まで変化させていきながら、割り切れる場合は divisible、割り切れない場合は indivisible というリストへ追加していきます。もうここで Text コンポーネントと Keyboard コンポーネントに渡せる形式にしてしまいましょう。2 で割れる値は他の除数 (3 から 9) より多いうえに判断も簡単なので、2 から 9 ではなく 3 から 9 にしてもいいかも知れません。

```
divisible = []
indivisible = []
for dividend in range(10,100):
    for divisor in range(2,10):
        if dividend % divisor == 0:
```

(次のページに続く)

(前のページからの続き)

```
            divisible.append(('{}÷{}'.format(dividend, divisor),'right'))
        else:
            indivisible.append(('{}÷{}'.format(dividend, divisor),'left'))
```

divisible と indivisible をそれぞれ無作為に並び替えて、スライスで必要な問題数を取り出して結合します。リスト同士の結合は + 演算子が使えます。割り切れる問題と割り切れない問題の比率を 1:1 以外にしたい場合は取り出す問題数で調整しましょう。

```
shuffle(divisible)
shuffle(indivisible)
delay_task_question_set = divisible[:100] + indivisible[:100]
```

あとは問題を直接コードに記入する場合と同じです。

第 13 章

付録

13.1 チェックリスト一覧

第 1 章

- PsychoPy を起動できる。
- Coder や Runner のウィンドウから Builder のウィンドウを開くことができる。
- Excel または LibreOffice Calc のどちらかを起動できる。
- ファイルの拡張子を表示できる。
- Builder と Runner の役割を説明できる。
- スクリプトという語の意味を説明できる。
- ローカル実験とオンライン実験の違いを説明できる。
- Builder が Python と JavaScript の 2 種類のスクリプト変換に対応している理由を説明できる。

第 2 章

- ルーチンペインにコンポーネントを配置できる。
- Pilot モードと実行モードを切り替えできる。Run ボタンの色の違いから現在どちらのモードなのか判断できる。
- Builder の実験ファイルの拡張子を答えられる。
- 作成した実験を実行できる。
- Polygon コンポーネントを用いて三角形と長方形を表示できる。
- コンポーネントをルーチンから削除することができる。
- 本書で項目名を [ ] で囲んだ表記が何を表しているかを説明できる。
- PsychoPy における座標系の原点と水平、垂直軸の正の方向を答えられる。

- height が単位の状態で **[位置 [x, y] $]** 、 **[サイズ [w, h] $]** 、 **[位置揃え]** を使って任意の大きさの多角形を任意の位置に表示させることができる。
- Polygon コンポーネントで正五角形以上の正多角形を描画できる。
- Polygon コンポーネントで **[頂点数]** が 5 以上の時に **[位置 [x, y] $]** が例外的に図形のどの位置に対応するかを答えることができる。
- cm、deg、norm、height という単位を説明できる。
- 複数のコンポーネントをルーチンペインに配置できる。
- **[位置揃え]** の働きを説明できる。
- **[回転角度 $]** に適切な値を設定して図形を回転させて表示させることができる。
- 図形の正の回転方向を答えられる。
- 単位が norm の時に図形を回転させた時に生じる図形のひずみを説明できる。
- Text コンポーネントを用いて文字列を表示できる。
- 文字列を指定された位置に表示できる。
- 文字列を指定された大きさで表示できる。
- 文字列を上下反転、左右反転表示することができる。
- 文字列の自動折り返し幅を設定できる。どのような文字列では自動折り返しが起きないか説明できる。
- web/X11 color name による色指定で文字列の色を白、灰色、黒、赤、オレンジ色、黄色、黄緑色、緑、水色、青、ピンク、紫にすることができる。
- **[色空間]** を rgb に設定して、数値指定によって文字列の色を白、灰色、黒、赤、黄色、緑色、青色にすることができる。
- Polygon コンポーネントを用いて内部が塗りつぶされていない枠線だけの多角形を描画することができる。
- Polygon コンポーネントを用いて枠線がない多角形を描画することができる。
- ルーチンペイン上における視覚刺激コンポーネントの順番とスクリーン上での重ね順の関係を説明できる。
- ルーチンペイン上におけるコンポーネントの順番を変更できる。
- 視覚刺激コンポーネントの透明度を設定して完全な透明、完全な不透明とその中間の透明度で刺激を表示させることができる。
- 刺激の表示開始時刻と表示時間を指定して表示させることができる。
- 刺激の表示開始時刻と表示終了時刻を指定して表示させることができる。
- 刺激の表示終了時刻を定めずに表示させることができる。
- 実行中の実験を強制的に終了させることができる。

- foo_lastrun.py (foo は psyexp 実験ファイル名) の役割を説明することができる。
- data フォルダの役割を説明することができる。
- 実験結果を保存する必要がない場合、どのファイルを削除しても問題ないかを判断できる。
- 作製済みの psyexp ファイルを Builder で開くことができる。
- psyexp ファイルを別の名前で保存することができる。
- 実験設定ダイアログを開くことができる。
- 実験開始時に実験情報ダイアログを表示させるか否かを設定することができる。
- 登録済みのモニターのうちどれを実験に使用するかを実験設定ダイアログで設定できる。
- 実験をフルスクリーンモードで実行するか否かを設定することができる。
- フルスクリーンモードを使用しない時に、視覚刺激提示ウィンドウの幅と高さを指定できる。
- 視覚提示ウィンドウの背景色を指定できる。
- 「実験の設定に従う」で参照される単位を指定することができる。
- ESC キーによる実験の強制終了を有効にするか無効にするかを指定することができる。
- 実験記録のファイルを保存するフォルダ名を指定することができる。
- フルスクリーンモード時にマウスカーソルを表示するか否かを指定することができる。
- Pilot モード時のウィンドウサイズの設定を変更できる。
- モニターセンターを開くことができる。
- モニターセンターに新しいモニターを登録することができる。
- モニターの観察距離、解像度、スクリーン幅を登録することができる。

第 3 章

- 新たに実験を作成するときは、その実験のためのフォルダを用意してその中に psyexp ファイルを保存する習慣をつける。
- **[キーボードバックエンド]** は ioHub の機能が必要な実験でなければ PsychToolbox にする。
- Polygon コンポーネントを用いて円をスクリーン上に提示することができる。
- Polygon コンポーネントを用いて十字をスクリーン上に提示することができる。
- ルーチン実行中にキー押しが有効となる時間帯を指定できる。
- キーが押されると直ちにルーチンを中断するように設定することができる。
- 有効とするキーを指定できる。
- すべてのキーを有効にするように設定することができる。

- Excel または LibreOffice Calc を用いて、実験に用いるパラメータを列挙した条件ファイルを作成することができる。
- 実験のパラメータを表現するために PsychoPy で使用できる名前を決めることができる。
- psyexp ファイルを保存しているフォルダを Builder から開くことができる。
- フローペインでループの挿入を開始できる。
- ループを挿入する際、フローペインの矢印上に表示される黒い円の意味を説明できる。
- ループのプロパティ設定ダイアログで条件ファイルを指定し、表示された条件ファイルの概要が適切か確認できる。
- **[Loop の種類]** の sequential、random、fullRandom の違いを説明できる。
- **[繰り返し回数 $]** の値、条件ファイルに含まれる条件数、ルーチンが繰り返される回数の関係を説明できる。
- **[乱数のシード $]** に値を設定するとどのような効果が得られるか説明できる。
- 条件ファイルを **[条件]** に指定する前に psyexp ファイルを保存すべき理由を説明できる。
- 条件ファイルで定義したパラメータを用いてコンポーネントのプロパティ値を更新できるように設定できる。
- コンポーネントのプロパティ名の最後に$が付いているものと付いてないものの違いを説明できる。
- expInfo の participant に設定した文字列が記録ファイル名にどのように反映されるかを説明できる。
- data フォルダに作成される拡張子 log と psydat のファイルに何が記録されているか、概要を説明することができる。
- CSV ファイルの CSV とは何の略であり、何を意味しているか説明することができる。
- trial-by-trial 記録ファイルから、各試行で用いられたパラメータの値を読み取ることができる。
- trial-by-trial 記録ファイルから、各試行において押されたキーのキー名と反応時間を読み取ることができる。
- trial-by-trial 記録ファイルに記録されている反応時間の計測開始時点 (0.0 秒になる時点) がどのように決まるか答えることができる。
- trial-by-trial 記録ファイルから実験情報ダイアログで入力した値を読み取ることができる。
- trial-by-trial 記録ファイルから実験実行時のフレームレートを読み取ることができる。
- 分析時に未変更の記録ファイルを保存しておいた方がよい理由を説明できる。
- 正答／誤答を記録するように Keyboard コンポーネントを設定することができる。
- キーを押さないことが正答となるように設定することができる。
- 正答となるキー名を条件ファイルから読み込んで繰り返し毎に更新することができる。
- trial-by-trial 記録ファイルから、各試行の正答／誤答を読み取ることができる。

- ルーチンを新たに作成してフローに追加することができる。
- 既存のルーチンをフローに追加することができる。
- フローからルーチンを削除することができる。
- フローペインの赤いルーチンと緑のルーチンが何に対応しているか説明することができる。
- 特定の Keyboard コンポーネントが検出したキー押しを記録しないように設定することができる。
- 既存のルーチンと同一の内容を持つ新しいルーチンをコピー＆ペーストの機能を使って作成することができる。

第 4 章

- Grating コンポーネントを用いて長方形、または楕円形に縞模様が描かれた刺激を提示することができる。
- Grating コンポーネントで **[空間の単位]** が cm、deg、pix、norm、height のいずれの場合においても、「幅 x に対して y 周期分の縞模様」を描けるように **[空間周波数 $]** の値を決定できる (x、y は正の数値)。
- Grating コンポーネントで描かれる縞模様を初期設定の状態からずらして描画することができる。
- Grating コンポーネントで描画処理の負担を軽くするためにテクスチャの解像度を下げることができる。
- Grating コンポーネントを大きく表示するときに画質を高めるためにテクスチャの解像度を上げることができる。
- Grating コンポーネントの色を指定したときに、何色の縞模様が描かれるのかをこたえることができる。
- 恒常法の実験において、正答率がそのまま心理物理曲線の縦軸の値として使用できるように正答を定義できる。
- 実験情報ダイアログの項目を追加、削除することができる。
- 実験情報ダイアログの項目の初期値を設定することができる。
- 実験情報ダイアログの項目名から、その値を利用するための Builder 内における表記に変換することができる。
- 条件ファイル名を実験情報ダイアログの項目から取り出してループのプロパティに設定することができる。
- 実験情報ダイアログの項目をドロップダウンメニューにできる。
- 多重繰り返しを挿入できる。
- 多重繰り返しの内側のループで条件ファイル名を外側のループの条件ファイルから読み込んで設定することができる。
- 一時的に、特定のルーチンを実行しないようにすることができる。
- ルーチンを、まだ実行中のコンポーネントが残っていても指定した実行時間で強制的に中断させることができる。

- 一時的に、特定のループを実行しないようにすることができる。
- Counterbalance ルーチンにおけるグループとスロットの意味を説明できる。
- Counterbalance ルーチンで各グループに異なるスロット数を設定できる。
- xlsx ファイルでグループとスロットを定義する際に、追加で定義したパラメータの値を利用することができる。
- Counterbalance ルーチンで管理している各グループのスロットの残りをリセットすることができる。

第 5 章

- 変数の役割を説明できる。
- 関数の役割を、引数、戻り値という用語を用いながら説明できる。
- Builder で使用できる数学関数を挙げることができる
- 関数の引数に別の関数の戻り値を使うことができる。
- 条件ファイルから読み込んだパラメータ値を関数の引数に使うことができる。
- Python の算術演算子 8 つ挙げてその働きを説明することができる。
- 現在のルーチンが開始してから経過した時間を保持している Builder の内部変数を答えられる。
- 現在のルーチンが開始してから描画したフレーム数を保持している Builder の内部変数を答えられる。
- 現在のルーチンが開始してから経過した時間の値を用いて、時間の経過とともに色や位置が変化する実験を作成することができる。
- 実験情報ダイアログに入力されたデータの型を答えることができる。
- データを整数型、浮動小数点型、文字列型、リストに変換することができる。
- 実験情報ダイアログを用いて刺激の大きさや位置などの値を実験開始時に指定するように実験を作ることができる。
- Code コンポーネントを用いて Python のコードをルーチンに配置することができる。
- Code コンポーネントのプロパティである **[実験初期化中]** 、 **[実験開始時]** 、 **[Routine 開始時]** 、 **[フレーム毎]** 、 **[Routine 終了時]** 、 **[実験終了時]** の違いを説明できる。
- Code コンポーネントに Python 用のコード入力欄だけを表示するように設定できる。
- ルーチンにおける Code コンポーネントと他のコンポーネントの実行順序を説明できる。
- 変数に値を代入する式を書くことができる。

第 6 章

- 画像ファイルをスクリーン上に提示することができる。
- 画像ファイルを上下、または左右に反転させて提示することができる。
- 画像ファイルや条件ファイル等の位置を絶対パスで指定することができる。

- 画像ファイルや条件ファイル等の位置を相対パスで指定することができる。
- OS によるパスの書き方の違いを説明できる。
- 複数の OS で実行できる Builder の実験を作成するためにはどの記法でパスを記述したらよいか答えられる。
- 複数の文字列を結合した文字列を得る式を書くことができる。
- 条件ファイルや実験情報ダイアログから読み込んだ文字列が組み込まれた文を提示することができる。
- Text コンポーネントの **[文字列]** に Python の式を書いた時に、表示する文字列を改行させることができる。
- 条件に応じて処理を分岐させる Python コードを書くことができる。
- 数値の大小や一致・不一致に応じて処理を分岐させることができる。
- 文字列の一致・不一致に応じて処理を分岐させることができる。
- Python の比較演算子を 6 つ挙げてそれらの働きを説明することができる。
- Python の論理演算子を 3 つ挙げてそれらの働きを説明することができる。
- クラスとインスタンスの違いを説明することができる。
- データ属性とは何かを説明することができる。
- 変数 x に格納されたインスタンスのデータ属性 foo に値を代入したり値を参照したりするときの Python の式を書くことができる。
- BuilderKeyResponse のデータ属性を参照して押されたキー名を知ることができる。

第 7 章

- 3 通り以上の分岐を処理させる if 文を書くことができる。
- リストの中にある要素が含まれているか否かで処理を分岐させることができる。
- Code コンポーネントからルーチンを終了させることができる。
- TrialHandler のインスタンスから現在ループの何回目の繰り返しを実行中かを取得できる。
- TrialHandler のインスタンスから現在ループの繰り返し回数が残り何回かを取得できる。
- TrialHandler のインスタンスから現在ループの総繰り返し回数を取得できる。
- 上記 3 項目の値を使って「現在第 n 試行」、「残り n 試行」、「全 n 試行」といったメッセージをスクリーン上に提示できる。
- Code コンポーネントを用いて、実験記録ファイルに出力するデータを追加することができる。
- 現在実行中の繰り返しの n 回前、n 回後に使われるパラメータを取得するコードを記述することができる。
- if 文の中に入れ子上に if 文を組み込んだコードを記述することができる。

- if や elif の条件式が真であった時に実行されるコードがどこまで続いているかを判断することができる。if、elif の条件式が全て偽で else まで進んだときに実行されるコードがどこまで続いているかを判断することができる。
- 一連の if-elif-else の組み合わせがどこまで続いているかを判断することができる。
- 論理演算子を用いて複数の条件式をひとつの式にまとめることができる。

第 8 章

- Button コンポーネントで作成したボタンをクリックすることでルーチンを終了することができる。
- Button コンポーネントで作成したボタン上でマウスのボタンを押し続けることでフレーム毎に処理を実行させることができる。
- Button コンポーネントで作成したボタン上でマウスのボタンを 1 回押すごとに 1 回処理を実行させることができる。
- Button コンポーネントで作成したボタンに日本語の文字を表示させることができる。
- Slider コンポーネントで縦向き・横向きのスライダーを描画できる。
- Slider コンポーネントでラジオボタンを用いた多肢選択を描画できる。
- Slider コンポーネントで数値を回答させる際にマーカーの最小間隔を設定できる。
- Slider コンポーネントで参加者の回答を取得するコードを書ける。
- Form コンポーネントで複数の質問項目とスライダーのペアをルーチンに配置することができる。
- Form コンポーネントのすべての項目に回答したらルーチン終了のボタンを表示させることができる。
- Form コンポーネントのすべての項目に回答したらフォームを終了させることができる。
- ループの中で Form コンポーネントを使う時に繰り返しのたびに反応を初期化することができる。

第 9 章

- ルーチン終了時のマウスカーソルの位置およびマウスのボタンの状態を記録することができる。
- ボタンが押された時点のマウスカーソルの位置およびマウスのボタンの状態を記録することができる。
- 実験記録ファイルにマウスカーソルの座標やボタンの状態がリストとして複数件出力されている時に、個々のデータの座標値やその取得時刻を判断できる。
- Code コンポーネントでマウスカーソルの座標を取得するコードを記述できる。
- Code コンポーネントでマウスのボタンの状態を取得するコードを記述できる。
- getPressed() メソッドを用いて取得したマウスのボタンの状態を示すリストの各要素がどのボタンに対応しているか答えられる。また、リストの各要素の値とボタンの状態の関係を答えられる。
- Code コンポーネントを用いずにマウスカーソルの位置に刺激を描画することができる。
- シーケンス型のデータから要素をひとつ取り出して他の変数に代入したり関数の引数に使ったりすることができる。

- シーケンス型データの前から N 番目の要素を取り出す時の式を答えられる。
- シーケンス型データの後ろから N 番目の要素を取り出す時の式を答えられる。
- 関数を用いてシーケンス型データに含まれる要素数を調べることができる。
- シーケンス型のデータが入れ子構造になっている時に、要素であるシーケンス型データや、さらにその要素を取り出すことができる。
- 文字列の中から N 番目の文字を取り出して変数に代入したり関数の引数に使ったりすることができる。
- [1,2,3][0] といった具合に [から始まって、] の後に [ ] 演算子が続く式を評価した時の値を答えられる。
- ある座標が Polygon コンポーネントで描画した多角形の内側に含まれているか判定するコードを書くことができる。
- Polygon コンポーネントで描画した二つの多角形に重なっている部分があるが判定するコードを書くことができる。
- マウスカーソルの位置を設定するコードを書くことができる
- マウスカーソルの表示 ON/OFF を切り替えるコードを書くことができる。
- ある変数に格納されている値だけが異なる処理を繰り返し実行しなければいけない時に、for 文を用いて記述することができる。
- for 文を継続する必要がなくなった時に直ちに終了させることができる。
- for 文で現在処理中の要素に対してこれ以上処理を続ける必要がなくなった時に、次の要素の処理へ直ちに移行するコードを書くことができる。
- ルーチン内に含まれるコンポーネントの一覧を格納した Builder の内部変数の名前を答えられる。
- オブジェクトがある名前のデータ属性 (たとえば'foo') を持っているか判別するコードを書くことができる。
- ある文字列が別の文字列の中に含まれているか (例えば'psych' が'psychophysics' に含まれるか) 判別するコードを書くことができる。
- リストにデータを追加するメソッドである append( ) と extend( ) の違いを説明できる。
- 中身が空のリストを作成することができる。
- ルーチン内でのみ必要なデータを蓄積するリストを作成するコードはどの時点で初期化すべきか判断できる。
- % 演算子を用いて N フレーム (N=2,3,4,…) に 1 回の頻度で処理を実行させることができる。
- 2 次元配列の要素のインデックスから、その要素が何行目、何列目にあるかを計算することができる。
- 関数やメソッドの戻り値に直接 [ ] 演算子を適用した式を理解できる。
- Polygon コンポーネントで描画した多角形にマウスカーソルを重ねてクリックするとルーチンの終了や反応の記録などの処理を行うコードを書くことができる。

第 10 章

- 無圧縮 WAV 形式の音声ファイルを再生できる。
- 指定された周波数の音を鳴らすことができる。
- 指定されたキーコードの音を鳴らすことができる。
- 音声のボリュームを指定できる。
- 音声ファイルの再生を指定された時刻に途中終了できる。
- 様々な再生時間の音声ファイルの再生開始、終了に合わせて他のコンポーネントを開始または終了させることができる。
- 短時間に複数の Sound コンポーネントを鳴らそうとした時に期待した結果が得られない理由を説明できる。
- 異なるサンプリングレートの音声ファイルをひとつの実験で混ぜて使用してはいけない理由を説明できる。
- 動画ファイルを拡大縮小して再生することができる。
- 動画ファイルを音声なしで再生することができる。

第 11 章

- 指定した条件に当てはまる時にルーチンの実行をスキップさせることができる。
- 試行毎にシーケンスからひとつの値を無作為に選択するコードを書くことができる。
- low 以上 high 未満の整数を範囲とする一様乱数のサンプルをひとつ得るコードを書くことができる。(low、high は整数)
- range( ) を用いて、0 から n (n>0) までの整数を順番に取り出す range オブジェクトを作成することができる。
- range( ) を用いて、m から n (n>m) までの整数を順番に取り出す range オブジェクトを作成することができる。
- range( ) を用いて、m から n まで、s の間隔で整数を順番に取り出す range オブジェクトを作成することができる。ただし m、n は互いに異なる整数、s は非 0 の整数である。
- リストの要素を無作為に並べ替えることができる。
- m 個の要素を持つリストから、n 個の要素 (m>n) を重複なく無作為に抽出することができる。
- ルーチンに配置された視覚刺激コンポーネントをスクリーン上に描画させないようにすることができる。
- スライスを用いて、あるリストから連続する要素を抽出したリストを作り出すことができる。
- リストの先頭から要素を抽出する場合のスライスの省略記法を用いることができる。
- リストの末尾までの要素を抽出する場合のスライスの省略記法を用いることができる。

第 12 章

- ループ内で反応の正誤を順番にならべたリストを作ることができる。
- リストの末尾から指定された個数の値を取り出せる。
- ループを中断することができる。
- 実験実行時に条件に応じてループの実行をスキップできる。
- ループを活用して、フローで条件分岐を実現できる。
- ループを中断することができる。
- 実験開始からの経過時間を取得することができる。
- 開始から一定時間経過したら終了するループを組むことができる。
- 小数点以下の桁数を指定して数値を文字列に埋め込める。
- ひとつの文字列に複数の数値や文字列を位置を指定して埋め込める。
- PsychoPy のユーザー設定フォルダを開くことができる。
- Builder にルーチンのテンプレートを追加することができる。
- ルーチンテンプレートのカテゴリ名を設定することができる。
- 複数のルーチンテンプレートをカテゴリにまとめることができる。
- テンプレートの **[説明]** にテンプレート使用の際の注意点を記入しておくことができる。

13.2 本文未解説コンポーネント

PsychoPy Builder 2024.2.5 で利用できるコンポーネントのうち、本文で取り上げなかったものの概要を示します。

Aperture コンポーネント (ローカルのみ) 画

面を「穴」で切り抜くコンポーネントです。このコンポーネントはルーチンペインにおける描画順序の影響を受けません。つまり、Aperture コンポーネントの上に他の刺激を重ね書きしようとしても Aperture コンポーネントに切り抜かれてしまいます。

Dots コンポーネント (ローカルのみ) 一

様に運動する小さな点を大量に描くコンポーネントです。運動視の研究などに用います。Polygon コンポーネントを大量に用いるより PC への負担が軽く、高速に描画できます。以下の解説では、点が描画される範囲をフィールドと呼んでいます。また、指定された方向に動く点をターゲット、それ以外の方向に動く点をノイズと呼んでいます。

Panorama コンポーネント (ローカルのみ) パ

ノラマ画像を表示するコンポーネントです。キーボードやマウスで視点操作できます。

Progress コンポーネント プ

ログレスバーを描画します。

TextBox コンポーネント 枠
: 線を付けたり背景を塗りつぶしたりしたテキストボックスを描画しますが、特筆すべきはテキストを編集可能にできる点です。「 13.7: 実験中に日本語文字入力をさせたい方のために：「オンライン実験」をローカル環境でで実行する 」をご覧ください。

Brush コンポーネント 画
: 面上にマウスカーソルの軌跡を描けるようにするコンポーネントです。今のところ描いた軌跡を簡単に保存する方法がないので、使いどころが難しいかもしれません。

ButtonBox コンポーネント (ローカルのみ)
: PsychoPy がサポートするハードウェアボタンボックスを利用するコンポーネントです。各ボックスのドライバはプラグインとしてインストールする必要があります。Keyboard コンポーネントのように検出するボタンを指定してルーチンを終了させたりすることができます。

JoyButton コンポーネント ジ
: ョイスティックで反応を記録するコンポーネントです。Keyboard コンポーネントのように検出するボタンを指定してルーチンを終了させたりすることができます。

Joystick コンポーネント ジ
: ョイスティックで反応を記録するコンポーネントです。JoyButton コンポーネントと異なり、ボタンのみではなくスティックの状態も記録されます。ルーチンの実行中にジョイスティックの状態にアクセスするには Code コンポーネントを使って Joystick オブジェクトにアクセスする必要があります。

Survey ルーチン (オンラインのみ)
: Pavlovia を用いたオンライン調査を実施するためのルーチンです。Counterbalance ルーチンのように、コンポーネントとして配置するのではなくこれ自体が独立したルーチンとなります。

ResourceManager コンポーネント オ
: ンライン実験用のコンポーネントです。実験に使用するリソース (画像ファイルなど) をダウンロードするタイミングを制御します。

ParallelOut コンポーネント パ
: ラレルポートからのトリガー出力を行うためのコンポーネントです。

SerialOut コンポーネント シ
: リアルポートでの入出力を行うためのコンポーネントです。

EyeTreackerCalibration ルーチン ア
: イトラッカーのキャリブレーションを行います。通常のコンポーネントではなく単独でルーチンとしてフローに組み込んで使用します。PsychoPy がサポートするアイトラッカーのプラグインをインストールしたうえで、実験設定ダイアログの「アイトラッキング」タブでアイトラッカー選択して使用します。

EyeTreackerValidation ルーチン ア
: イトラッカーのバリデーション (キャリブレーション精度の確認) を行います。通常のコンポーネントではなく単独でルーチンとしてフローに組み込んで使用します。

EyeTreackerRecord コンポーネント ア
: イトラッカーによる記録の開始、終了を制御するコンポーネントです。

RegionOfInterest コンポーネント

ROI を定義し、視線の停留などを記録するコンポーネントです。

13.3 パッケージとプラグイン

特殊な実験機器と PsychoPy を連携させる必要がある場合や、実験に伴う何らかの処理のために PsychoPy に含まれていない Python のパッケージを必要する場合があります。PsychoPy にやこういった用途の支援のためにプラグインという仕組みがあります。本書では、この機能はあまり完成度が高くないうえに仕様が何度か変わってきたので取り上げていなかったのですが、最近は動作が安定してきたことや、バージョン 2024 でそれ以前のバージョンでは PsychoPy に組み込まれていた機能がプラグインとして分離されたことなどから、附録で少し解説しておくことにしました。

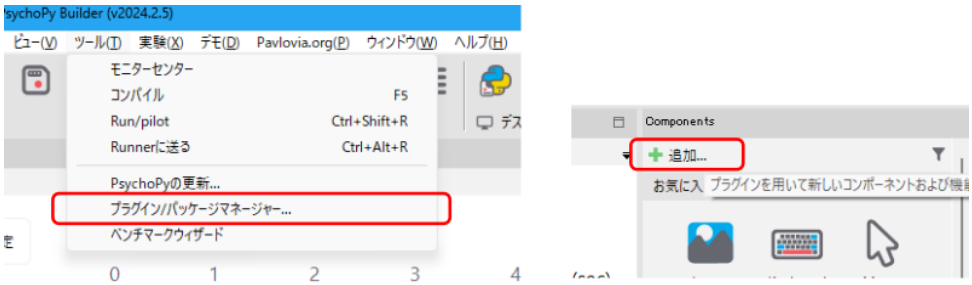


図 13.1　プラグイン/パッケージマネージャーを開く

プラグインの管理は「プラグイン/パッケージマネージャー」でおこないます。プラグイン/パッケージマネージャーは Builder または Coder のメニューの「ツール」から「プラグイン/パッケージマネージャー...」を選択するか、コンポーネントペインの一番上にある緑色のプラスマークに「追加...」と書かれているボタンをクリックして起動します (図 13.1)。特に初回はダイアログが開くまで時間がかかりますのでそのつもりでお待ちください。

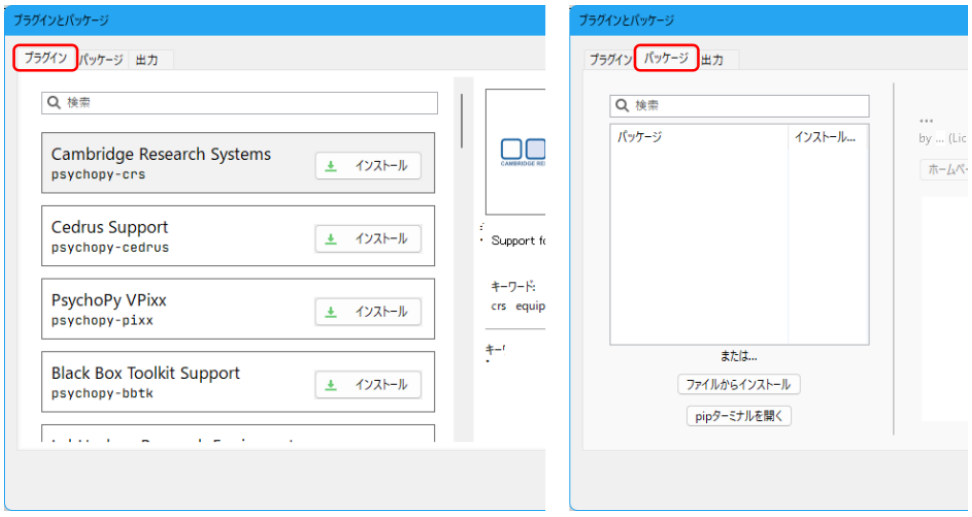


図 13.2　左: プラグインの管理とインストール 右: パッケージの管理とインストール

プラグイン/パッケージマネージャーダイアログには「プラグイン」、「パッケージ」、「出力」のタブがあります (図 13.2)。「プラグイン」タブには、PsychoPy がサポートするプラグインがリストアップされています。プラグインは Python のパッケージの一種ですが、PsychoPy で利用するために特別な仕様に従って作られています。主に Builder に新しいコンポーネントを追加したり、PsychoPy がサポートするハードウェアを使用でき

るようにしたりするために使用します。2025 年 4 月現在のプラグインを示します。

- Camblidge Research Systems (psychopy-crs)
- Cedrus Support (psychopy-cedrus)
- PsychoPy VPixx (psychopy-pixx)
- Black Box Toolkit Support (psychopy-bbtk)
- LabHackers Research Equipment (psychopy-labhackers)
- ioLabs Systems Button Box (psychopy-iolabs)
- Brain Products GmbH Device (psychopy-brainproducts)
- Gamma Scientific Inc. Support (psychopy-gammasci)
- Photo Research Inc. Support (psychopy-photoresearch)
- Labjack Support (psychopy-labjack)
- Konica Minolta Support (psychopy-minolta)
- MRI Emulator (psychopy-mri-emurator)
- Qmix Syringe Pump Support (psychopy-qmix)
- LabeoTech device support (psychopy-labeotech)
- Audio playback support using the SoundDevicew library (psychopy-sounddevice)
- Audio playback support using the Pyo library (psychopy-pyo)
- Emotive EEG (psychopy-emotiv)
- Advanced Vision Science (psychopy-visionscience)
- Tobii Eyetracker Support (psychopy-eyetracker-tobii)
- Pupil Labs Eyetracker Support (psychopy-eyetracker-pupil-labs)
- GazePoint Eyetracker Support (psychopy-eyetracker-gazepoint)
- EyeLogic Eyetracker Support (psychopy-eyetracker-eyelogic)
- Whisper Speech-to-Text (psychopy-whisper)
- SR Resarch Eyetracker Support (psychopy-eyetracker-sr-research)
- PsychoPy Legacy (psychopy-legacy)
- PsychoPy Monkeys (psychopy-monkeys)
- PsychoPy BIDS (psychopy-bids)

ボタンボックスやアイトラッカーなどの特殊なハードウェアをサポートするためのパッケージが数多くありますが、注目しておきたいのが PsychoPy Legacy と PsychoPy Vision Science です。これらは従来のバージョンの Builder では標準で組み込まれていて 2024 では削除されたコンポーネントを含んでいます。また、オーディオ関係のプラグインのうち、Audio playback support using the SoundDevicew library と Audio playback support using the Pyo library はそれぞれ以前のバージョンで標準のサウンドバックエンドであったことあるsounddevice と pyo をサポートするためのものです。以前のバージョンで作成した実験を実行する際には必要になるかもしれません。

「プラグイン」タブは、プラグインの仕様ではない Python のパッケージをインストールするときに使います。現時点でインストールされている追加パッケージの一覧 (PsychoPy のインストール後に何も追加していなければ 図 13.1 右のように空欄) が示されており、その下に「ファイルからインストール」「pip ターミナルを開く」というボタンがあります。追加したいパッケージが zip や tar.gz などのアーカイブファイルとして提供されている場合は、「ファイルからインストール」を選択してそのファイルを選択するとインストールできます。pip で web 上からインストールできるパッケージであれば、「pip ターミナルを開く」をクリックして、pip のコマンドを入力すればインストールすることができます。

プラグインおよび追加パッケージは、PsychoPy のユーザー設定フォルダ (「 12.7: テンプレートにしてみよう」参照) の packages フォルダ内に保存されます。ですから、PC に他の Python がインストールされていても、そちらには影響を与えません。

13.4 予約語

13.4.1 Python の予約語 (Python 3.10)

Python インタプリタを起動して keyword を import すると、keyword.kwlist というリストに Python 予約語の一覧が格納されます。以下に Python3.10 の予約語を示します。これらの語は Builder において **[名前]** や変数名として使用することはできません。

> False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield,

__builtin__という内部モジュールに含まれる以下の語は Python の予約語ではありませんが、予約語と同様に扱われます (つまり **[名前]** や変数名として使用することはできません)。

> ArithmeticError, AssertionError, AttributeError, BaseException, BlockingIOError, BrokenPipeError, BufferError, BytesWarning, ChildProcessError, ConnectionAbortedError, ConnectionError, ConnectionRefusedError, ConnectionResetError, DeprecationWarning, EOFError, Ellipsis, EnvironmentError, Exception, False, FileExistsError, FileNotFoundError, FloatingPointError, FutureWarning, GeneratorExit, IOError, ImportError, ImportWarning, IndentationError, IndexError, InterruptedError, IsADirectoryError, KeyError, KeyboardInterrupt, LookupError, MemoryError, ModuleNotFoundError, NameError, None, NotADirectoryError, NotImplemented, NotImplementedError, OSError, OverflowError, PendingDeprecationWarning, PermissionError, ProcessLookupError, RecursionError, ReferenceError, ResourceWarning, RuntimeError, RuntimeWarning, StopAsyncIteration, StopIteration, SyntaxError, SyntaxWarning, SystemError, SystemExit, TabError, TimeoutError, True, TypeError, UnboundLocalError, UnicodeDecodeError, UnicodeEncodeError, UnicodeError, UnicodeTranslateError, UnicodeWarning, UserWarning, ValueError, Warning, WindowsError, ZeroDivisionError, _, __build_class__, __debug__, __doc__, __import__, __loader__, __name__, __package__, __spec__, abs, all, anyascii, bin, bool, bytearray, bytes, callable, chr,

classmethod, compile, complex, copyright, credits, delattr, dict, dir, divmod, enumerate, eval, exec, exit, filter, float, format, frozenset, getattr, globals, hasattr, hash, help, hex, id, input, int, isinstance, issubclass, iter, len, license, list, locals, map, max, memoryview, min, next, object, oct, open, ord, pow, print, property, quit, range, repr, reversed, round, set, setattr, slice, sorted, staticmethod, str, sum, super, tuple, type, vars, zip

13.4.2 PsychoPy の予約語 (2024.2.5)

以下の語は PsychoPy で利用するモジュール名のため、Builder において **[名前]** や変数名として使用することができません。

core, data, event, gui, microphone, misc, os, psychopy, sound, visual

以下の語は PsychoPy で予約されているため、Builder において **[名前]** や変数名として使用することができません。これらの他に__で始まる名前が数多く登録されていますが、本文で述べた通り__で始まる名前はそもそも使用するべきではないので省略しています。

clear, copy, fromkeys, get, items, keys, pop, popitem, self, setdefault, update, values

以下の語は Builder で予約されているため、Builder において **[名前]** や変数名として使用することができません。

KeyResponse, _thisDir, buttons, component, continueRoutine, currentLoop, dlg, endExpNow, expInfo, expName, filename, frameDur, frameN, globalClock, keyboard, level, logFile, paramName, routineTimer, t, theseKeys, thisComponent, thisExp, win, x, y

以下の語は numpy および numpy.random から import されるので Builder において **[名前]** や変数名として使用することはできません。

asarray, average, randchoice, cos, deg2rad, linspace, log, log10, normal, np, pi, rad2deg, randint, random, shuffle, sin, sqrt, std, tan

13.4.3 Builder の内部変数 (2024.2.5)

Builder で使用されている内部変数の概要を示します。

予約語	概要
KeyResponse	予約されています。
_thisDir	Builder を実行するときにカレントフォルダを psyexp ファイルがある場所に移動するために使用されます。
buttons	最後に取得したマウスのボタンの状態を示すリストが格納されています。
component	予約されています。
continueRoutine	実行中のルーチンを継続するか否かを示す真偽値が格納されています。→ 第 7 章

次のページに続く

表 13.1 – 前のページからの続き

予約語	概要
currentLoop	Loop の種類で staircase、interleaved staircases を選択したときに使用されます。→ 第 3 章
dlg	expInfo ダイアログを作成するために使用します。
endExpNow	ESC キーによる実験の中断を有効にしているときに、この変数を利用して ESC キー以外のキーで実験を終了できます。
expInfo	expInfo ダイアログの項目と値が辞書オブジェクトとして格納されています。→ 第 4 章 、第 5 章
expName	実験設定ダイアログの **[実験の名前]** に入力した実験名が格納されています。
filename	各種実験記録ファイルやログファイルのファイル名を生成するために利用されます。
frameDur	フレームレートの実測値を保持しています。計測に失敗した場合は 1/60sec にセットされます。
frameN	現在のフレーム番号を格納しています。→ 第 5 章
globalClock	実験開始からの経過時間を計測するための psychopy.core.Clock のインスタンスが格納されています。→ 第 12 章
keyboard	キーボード用のライブラリの読み込みのために使用されています。
level	Loop の種類で staircase、interleaved staircases を選択したときに使用されます。→ 第 3 章
logFile	ログファイルを作成するための psychopy.logging.LogFile のインスタンスが格納されています。
paramName	Loop の種類で interleaved staircases を選択したときに使用されます。→ 第 3 章
routineTimer	ルーチン終了までの残り時間を計測するための psychopy.core.CountdownTimer のインスタンスが格納されています。
t	ルーチンが開始してからの経過時間を格納しています。→ 第 5 章
theseKeys	最後に取得したキーの状態を示すリストを格納しています。→ 第 7 章
thisComponent	現在処理中のコンポーネントに対応するインスタンスが格納されています。
thisExp	フローの制御ややデータの保存に関与する psychopy.data.ExperimentHandler のインスタンスが格納されています。
win	刺激提示スクリーン本体である psychopy.visual.Window のインスタンスを格納しています。
x	最後に取得したマウスカーソルの X 座標を格納しています。Mouse コンポーネントの **[マウスの状態を保存]** の設定によって単独の値であったりリストであったりします。

次のページに続く

表 13.1 – 前のページからの続き

予約語	概要
y	最後に取得したマウスカーソルの Y 座標を格納しています。Mouse コンポーネントの **[マウスの状態を保存]** の設定によって単独の値であったりリストであったりします。

以上に加えて、以下の語は正常なルーチンの実行に必須の変数名と一致するので Builder において **[名前]** や変数名として使用することはできません。

trialComponents (trial はルーチン名)	当該ルーチンでフレーム毎に処理する必要があるコンポーネントのインスタンスを並べたリストが格納されています。→ 第 9 章
trialClock (trial はルーチン名)	当該ルーチンが開始されてからの経過時間を計測する psychopy.core.Clock のインスタンスが格納されています。→ 第 8 章

13.5 ログファイル

PsychoPy Builder で実験を実行すると、拡張子 .log のログファイルが作成されます。実験が十分な精度で実行されているかどうかを確認したい場合や、どうも意図している通りに刺激が提示されないといった問題が生じている時に、このログファイルから情報が得られる場合があります。ただし、Builder の実験がどのようなコードにコンパイルされて実行されるのかある程度知識がないと対策をとることまでは難しいかも知れません。

ログには以下のレベルがあります。実験設定ダイアログでログの出力レベルを選択すると、選択されたレベル以上のログがログファイルに出力されます。例えば warning を選択すると、waring と error のレベルのログが出力されます。info を選択すると、info、exp、data、warning、error のレベルのログが出力されます。

1. error
2. warning
3. data
4. exp
5. info
6. debug

以下に debug を選択した場合のログファイルの先頭部分の例を示します。非常に長い行は中略してあります。各行の最初の数値が実験開始からの経過時間 (秒)、続いてログのレベルが示されています。レベルに続いてその時刻に生じたイベントの内容が書かれています。

```
3.0528  WARNING  Movie2 stim could not be imported and won't be available
7.8399  INFO  Loaded monitor calibration from ['2015_06_02 16:25']
8.9837  EXP  Created window1 = Window(allowGUI=False, allowStencil=False, ... (略)
8.9838  EXP  window1: recordFrameIntervals = False
9.1467  EXP  window1: recordFrameIntervals = True
```

(次のページに続く)

(前のページからの続き)

```
9.3343  DEBUG  Screen (0) actual frame rate measured at 58.77
9.3344  EXP  window1: recordFrameIntervals = False
9.9161  EXP  Created text = TextStim(alignHoriz='center', alignVert= ... (略)
9.9274  EXP  Created stimulus = Polygon(autoDraw=False, autoLog=True, ... (略)
9.9286  EXP  <method-wrapper '__getattribute__' of attributeSetter object ... (略)
(中略)
13.8202  EXP  Created sequence: fullRandom, trialTypes=4, nReps=25, seed=None
13.8229  EXP  New trial (rep=0, index=0): {u'correct_ans': u'slash', ... (略)
13.8499  EXP  text: autoDraw = False
13.8499  EXP  stimulus: fillColor = u'green (named)'
13.8499  EXP  stimulus: pos = array([-400.,    0.])
13.8499  EXP  stimulus: lineColor = u'green (named)'
13.8499  EXP  cross1: autoDraw = True
13.8499  EXP  cross2: autoDraw = True
14.8756  EXP  stimulus: autoDraw = True
15.4022  DATA  Keypress: slash
15.4382  EXP  New trial (rep=0, index=1): {u'correct_ans': u'slash', ... (略)
15.4778  EXP  stimulus: autoDraw = False
15.4778  EXP  cross1: autoDraw = False
15.4778  EXP  cross2: autoDraw = False
15.4778  EXP  stimulus: fillColor = u'green (named)'
15.4778  EXP  stimulus: pos = array([-400.,    0.])
15.4778  EXP  stimulus: lineColor = u'green (named)'
15.4778  EXP  cross1: autoDraw = True
15.4778  EXP  cross2: autoDraw = True
16.4711  EXP  stimulus: autoDraw = True
17.0442  DATA  Keypress: slash
```

もしこの実験をログレベル exp で実行していたら、exp 以上のレベルのみが出力されるので、以下のような出力になります。上の例の 2 行目の INFO と 6 行目の DEBUG が抜けている点にご注意ください。

```
3.0528  WARNING  Movie2 stim could not be imported and won't be available
8.9837  EXP  Created window1 = Window(allowGUI=False, allowStencil=False, ... (略)
8.9838  EXP  window1: recordFrameIntervals = False
9.1467  EXP  window1: recordFrameIntervals = True
9.3344  EXP  window1: recordFrameIntervals = False
9.9161  EXP  Created text = TextStim(alignHoriz='center', alignVert= ... (略)
9.9274  EXP  Created stimulus = Polygon(autoDraw=False, autoLog=True, ... (略)
9.9286  EXP  <method-wrapper '__getattribute__' of attributeSetter object ... (略)
(中略)
13.8202  EXP  Created sequence: fullRandom, trialTypes=4, nReps=25, seed=None
13.8229  EXP  New trial (rep=0, index=0): {u'correct_ans': u'slash', ... (略)
```

(次のページに続く)

(前のページからの続き)

```
13.8499  EXP  text: autoDraw = False
13.8499  EXP  stimulus: fillColor = u'green (named)'
13.8499  EXP  stimulus: pos = array([-400.,    0.])
13.8499  EXP  stimulus: lineColor = u'green (named)'
13.8499  EXP  cross1: autoDraw = True
13.8499  EXP  cross2: autoDraw = True
14.8756  EXP  stimulus: autoDraw = True
15.4022  DATA Keypress: slash
15.4382  EXP  New trial (rep=0, index=1): {u'correct_ans': u'slash', ... (略)
15.4778  EXP  stimulus: autoDraw = False
15.4778  EXP  cross1: autoDraw = False
15.4778  EXP  cross2: autoDraw = False
15.4778  EXP  stimulus: fillColor = u'green (named)'
15.4778  EXP  stimulus: pos = array([-400.,    0.])
15.4778  EXP  stimulus: lineColor = u'green (named)'
15.4778  EXP  cross1: autoDraw = True
15.4778  EXP  cross2: autoDraw = True
16.4711  EXP  stimulus: autoDraw = True
17.0442  DATA Keypress: slash
```

ログレベルが warning なら、以下のような出力になります。

```
3.0528  WARNING  Movie2 stim could not be imported and won't be available
```

info や debug のレベルのログは Builder の新機能の開発などの際に便利な内容が多く、おそらく一般ユーザーが必要とすることは少ないでしょう。exp のレベルは刺激オブジェクトの作成やパラメーターの変更、ループの実行状況などに関する情報が出力されます。data のレベルは実験記録ファイルへのデータ出力に関する情報が出力されます。exp のログの時刻を確認したら、もしかすると刺激提示のタイミングなどに関するトラブルの情報が得られる可能性があります。

warning のレベルでは、致命的ではないかも知れないけれども問題が生じていることが報告されます。上の例では MovieStim2 が利用できないという問題が報告されています。今実行している実験の中で動画刺激を使用していなければ問題ありませんが、今後動画刺激を使いたいと思った時には利用できるようにインストールの問題などを解決しなければいけないことを示しています。

warning のレベルで最も注意すべきは、以下のように last frame was XX ms というログが出力されている場合です。このようなログが出力されている場合は、フレームの描画に問題があって一定のスピードで描画ができていません。アニメーションする刺激の描画がカクカクしてしまったり、刺激の出現、消去のタイミングがずれてしまっている可能性があります。常駐プログラムの停止、グラフィックデバイスドライバの更新や、高性能グラフィックボードの追加などの対策が必要となる可能性があります。

```
16.5660  WARNING  t of last frame was 30.22ms (=1/33)
16.5974  WARNING  t of last frame was 31.41ms (=1/31)
16.6285  WARNING  t of last frame was 31.09ms (=1/32)
```

13.6 PsychoPy 設定ダイアログ

PsychoPy 設定ダイアログの各項目について簡単にまとめました。

13.6.1 一般

[単位] 実験の設定ダイアログで **[単位]** を「PsychoPy の設定を用いる」に設定した場合、ここで選択した単位が使用されます。

[フルスクリーン] チェックしておくと標準でフルスクリーンウィンドウで刺激を描画しようとします。実験の設定ダイアログで「フルスクリーンウィンドウ」のチェックを外している場合は、実験の設定ダイアログが優先されます。

[GUI を使用] チェックしておくと標準でマウス等を有効にします。

[パス] 独自の Python モジュールを使用したい場合、モジュールが置かれているパスをここに列挙しておくと import できます。

[frac オーディオ圧縮] frac を利用したい場合、ここへ frac へのパスを設定します。

[実験終了キー] 実験を終了させるキーを ESC 以外にしたい時にここに指定します。pyglet のキー名でなければいけません。

[実験終了キーのモディファイア] 「Shift を押しながら ESC」のようにモディファイアキーを押しながら終了するようにしたい時にここに指定します。

[ガンマエラーの処理] ガンマ補正が出来なかった時に終了するか警告するかを指定します。

[スタートアッププラグイン] PsychoPy 起動時に追加で読み込むプラグインを指定します。

[スタートアッププラグイン] 音声文字変換で Google Cloud API を使用する際に、Google Cloud API キーが格納された JSON ファイルをここに指定します。

13.6.2 アプリケーション

[起動時にチップを表示] チェックしておくと PsychoPy Builder/Coder を起動したときに「今日のチップ」を表示します。

[標準で開くウィンドウ] PsychoPy を起動したときに開くウィンドウを指定します。

[設定の初期化] 設
定を初期化したいときは、ここにチェックをして PsychoPy を再起動してください。

[設定の自動保存] ウ
ィンドウを閉じるときに、未保存の設定を自動的に保存します。

[デバッグモード]
PsychoPy のデバッグ用機能を有効化します。PsychoPy そのもののデバッグであって、ユーザーが作成した実験スクリプトのデバッグではありません。

[ロケール]
PsychoPy のメニュー等の表示に使う言語を選択します。「システムの言語設定」にしておくと、OS の言語の設定を利用します。

[エラーダイアログ]
PsychoPy の動作中にエラーが生じた際にダイアログを表示します。バグの報告をする際はこのダイアログの情報を伝えると開発チームによる問題の把握に役立ちます。

[テーマ] 起
動時に適用するテーマを指定します。旧バージョンの見た目に近づけたい場合は Classic を選択するとよいでしょう。

[スプラッシュを表示] 起
動時にスプラッシュウィンドウを表示するか否かを指定します。

Builder 関連

[前回の実験を開く] チ
ェックしておくと、前回開いていた実験を自動的に開きます。

[Code コンポーネントの言語]
Code コンポーネントの標準のコードタイプを指定します。JavaScript への自動変換に問題がある場合や、JavaScript 用入力欄が不要な場合などに設定を変更してください。

[名前空間の整理]
コードのコンパイルに関連するオプションです。通常は変更する必要はないでしょう。

[コンポーネントフォルダ] パ
スを列挙したリストを指定します。パス内に含まれる拡張子.py のファイルを Builder のコンポーネントとして読み込みます。

[コンポーネント表示制限]
Builder において Python によるローカル実験、PsychoJS によるオンライン実験それぞれに対応したコンポーネントのみを表示したい場合にここで設定します。

[表示しないコンポーネント] 特
定のハードウェア用コンポーネントなど、使用しないコンポーネントの名前をここに書いておくと、コンポーネントペインに表示されなくなります。ロケールが日本語に設定されていて、プロパティ設定ダイアログのタイトルバーに「Foo コンポーネント」と表示されているコンポーネントを非表示にしたい場合は FooComponent と書く必要があります。

[長いコンポーネント名を省略] コンポーネントをルーチンに配置する際、**[名前]** に長い文字列を指定すると、名前を表示するスペースを確保するためにタイムラインの表示幅が狭くなってしまいます。この項目をチェックしておくと、**[名前]** が長いときに一部を省略して表示します。

[デモのディレクトリ]
Builder のデモが展開されているディレクトリを指定します。Builder ウィンドウのメニューの「デモ」メニューから「デモを展開...」すると自動的に設定されます。展開されたデモを手作業で別の場所に移したりしない限り、通常は変更する必要はありません。

[データ保存フォルダ]
Builder の実験を実行したときに、記録ファイルやログファイルが保存されるフォルダ名を指定します。

[Builder のレイアウト]
Builder の各ペインの配置を指定します。

[常に readme を表示]
psyexp ファイルと同一のフォルダに readme.txt というファイルが存在している場合、この項目をチェックしておくと readme.txt の内容が表示されます。

[お気に入りの最大登録数] コンポーネントペインの「お気に入り」に登録できるコンポーネント数の上限を指定します。

[Routine を閉じるときの確認]
Routine のタブを閉じるときに確認ダイアログを表示するかどうかを指定します。

Coder 関連

[読み込み専用で開く] 実験を実施する際に、不用意にコードが変更されてしまうのを防ぎたいときにチェックします。

[出力用フォント] 出力パネルの表示に使用するフォント名を指定します。複数のフォントを列挙すると、最初に読み込むことができたフォントを使用します。

[コード用フォント]
コードの表示に使用するフォント名を指定します。複数のフォントを列挙すると、最初に読み込むことができたフォントを使用します。

[コード用フォントサイズ] フォントサイズを 6 から 24 の整数で指定します。

[出力用フォントサイズ] フォントサイズを 6 から 24 の整数で指定します。

[行間スペース] 行間を整数で指定します。

[文字数ガイドの位置] 1
行の長さの目安となる縦線を引く位置を文字数で指定します。

[ソースアシスタントを表示] チェックすると Coder 起動時にソースアシスタントを表示します。変数に格納されているクラスのヘルプなどが表示されます。

[出力パネルを表示] チェックを外して PsychoPy を再起動すると、Coder ウィンドウの下部に出力パネルが表示されません。

[自動補完] 入力時にコード補完の候補と関数呼び出しのヒントを表示するか否かを指定します。

[前回のファイルを開く] 前回終了時に開いていたファイルを自動的に開きます。

[優先するシェル] Coder ウィンドウ下部のシェルパネルで優先するシェルを指定します。

13.6.3 Pilot モード

Pilot モードに関する設定を行います。「2.10:Pilot モードのウィンドウサイズを設定しよう」も参照してください。

[ウィンドウモードを強制] Pilot モード時に強制的にウィンドウモードで実行します。実験設定ダイアログの **[フルスクリーンウィンドウ]** の設定は無視されます。

[ウィンドサイズを強制] Pilot モード時に強制的にウィンドウをここで指定したサイズにします。実験設定ダイアログの **[ウィンドウの大きさ (pix) $]** は無視されます。

[Pilot モードのログレベル (ファイル)] Pilot モード時にログファイルに出力されるログのレベルを指定します。

[Pilot モードのログレベル (コンソール/アプリ)] Pilot モード時にコンソールやアプリに出力されるログのレベルを指定します。

[Pilot モードインジケーターの表示] Pilot モード時にインジケーター (画面を囲むオレンジ色の枠＋左下のメッセージ) を表示するか否かを指定します。

[Rush モード無効化] Pilot モード時に Rush モードを無効にするか否かを指定します。

13.6.4 キー設定

ショートカットキーを編集することができます。個々の項目については省略します。

13.6.5 ハードウェア

[オーディオドライバ] 音
声刺激の再生に使用するドライバを列挙します。最初にロードできたものを利用します。

[オーディオデバイス] 音
声刺激の再生に使用するデバイスを指定します。PC に複数のオーディオデバイスがある場合、適切なものを選択しないと音声が再生されないので注意してください。

[パラレルポート] パ
ラレルポートのアドレスを列挙します。ここへアドレスを記入しておかないと ParallelOut コンポーネントでアドレスを選択できません。

[Qmix の設定]
Qmix pump(シリンジ制御ライブラリ) の設定を指定します。

13.6.6 ネットワーク

[プロキシ] ネ
ットワークアクセスにプロキシが必要な場合、ここへアドレスとポートを記入します。

[プロキシの自動構成] 自
動的にプロキシを構成しようとします。自動構成が利用可能な場合は「プロキシ」の設定に優先します。

[利用統計の送信を許可]
PsychoPy 開発チームへ利用統計を送信することを許可します。開発の参考にするため、可能な限りチェックしておいてください。

[更新の確認] 起
動時に新しいバージョンの PsychoPy が公開されていないか確認します。

[タイムアウト] ネ
ットワーク接続のタイムアウト時間を指定します。

13.7 実験中に日本語文字入力をさせたい方のために：「オンライン実験」をローカル環境でで実行する

題目の通り、実験の手続き上、フリーテキストで日本語文字入力をさせたい方向けの情報です。今後のバージョンアップで Textbox コンポーネントや Form コンポーネントでの日本語入力がローカル実験でも使用できるようになった場合は不要になる内容なので、附録として収録しておきます。

準備作業

- このデモのためのフォルダを作成して、その中に demo_textinput.psyexp という名前で新しい実験を保存する。
- 実験設定ダイアログの「スクリーン」タブの **[マウスカーソルを表示]** をチェックし、**[単位]** が height であることを確認する。

- trial ルーチン

- TextBox コンポーネント (「反応」カテゴリにある) をひとつ配置し、以下のように設定する。
  - 「基本」タブの **[開始]** を 0.5、 **[終了]** を空欄にする。 **[編集可能]** をチェックし、 **[文字列]** を空欄にする。
  - 「レイアウト」タブの **[オーバーフロー]** を スクロール にする。
  - 「外観」タブの **[塗りつぶしの色]** を darkgray に、 **[枠線の色]** を white にする。
  - 「書式」タブの **[行揃え]** を 左上 にする。
- Button コンポーネントをひとつ配置し、以下のように設定する。
  - 「基本」タブの **[開始]** を 0.5、 **[終了]** を空欄にし、 **[ボタンのテキスト]** を OK にする。
  - 「レイアウト」タブの **[サイズ [w, h] $]** を (0.1, 0.05) に、 **[位置 [x, y] $]** を (0.3, -0.4) にする。
  - 「書式」タブの **[文字の高さ $]** を 0.03 にする。

作業ができたら、Builder のウィンドウ上部のリボンを見てください。「ブラウザ」というカテゴリがあり、図 13.3 に示す「ブラウザで動作確認」ボタンがあるはずです。もしこのボタンが緑色なら Pilot モードになっていないので、Pilot モードに切り替えてください。このボタンをクリックすると、自動的に web ブラウザが開いて、ブラウザの中で実験が動作します。バージョン 2024.2.5 の時点で、Pilot モードの諸設定はブラウザ上で実行する場合無効になるので、Pilot モードを示すオレンジ色の枠線とメッセージも表示されませんし、ウィンドウモードの強制もされません。ブラウザで実行すると、Textbox に文字を入力する際に日本語入力に切り替えても自然に入力、漢字変換できることを確認してください。

図 13.3　実験をローカルブラウザ上で動作確認する

このように、実験を「オンライン実験」の「ローカルブラウザ上での動作確認」として実行することで、事実上ローカル環境で実行することができます。これを利用すると、TextBox コンポーネントや Form コンポーネントのフリーテキスト入力で日本語を問題なく入力できます。

この方法のデメリットは、オンライン実験に対応しているコンポーネントしか使用できないことです。オンライン実験に対応しているコンポーネントを確認するには、Builder のコンポーネントペインの一番上にあるフィルターボタン (図 13.4) を使用します。もし表示されていなければ、コンポーネントペインを一番上までスクロールしてください。「お気に入り」カテゴリの上にあるはずです。フィルターボタンをクリックすると、図 13.4 の左下あたりに示されている小さなダイアログが開きます。ここで「PsychoJS (online)」をクリックすると、オンライン実験に対応しているコンポーネントのみがコンポーネントペインに表示されるようになります (PsychoPy 設定ダイアログの **[コンポーネント表示制限]** と連動しています)。ちなみに「PsychoPy (local)」はローカル実験に対応しているもののみ、「Both」はオンラインとローカルの両方に対応しているもののみに

表示を制限します。「Any」にするとすべてのコンポーネントが表示されます。文字入力をおこないたい実験で他に使いたいコンポーネントがオンライン実験に対応しているか確認してください。

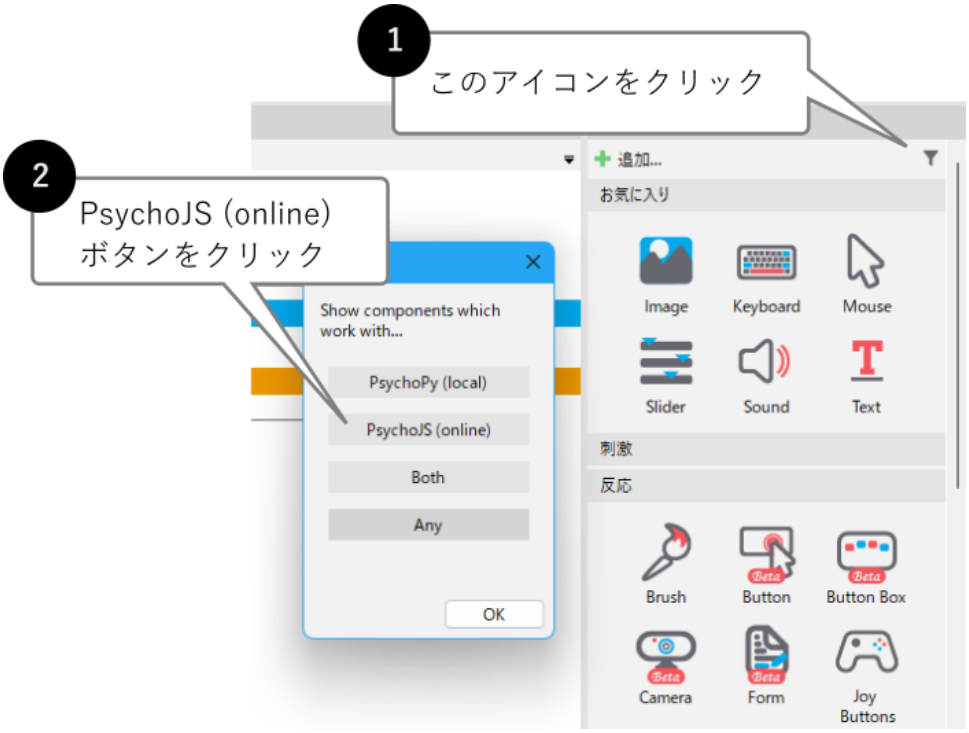


図 13.4　表示するコンポーネントをオンライン実験に対応するもののみに制限する

あともうひとつ、デメリットというほどの事ではないかも知れませんが、この方法で実験をおこなうと、実験終了時にデータが「ダウンロード」されるので、データはブラウザの標準のダウンロードフォルダに出力されます。「実験に使用するファイルや記録されたデータファイルはすべて psyexp ファイルがあるフォルダにまとめる」という Builder の設計思想から外れた挙動となるので注意してください。実験の時間精度に関しては、最近の web ブラウザはその中でリアルタイムな操作が求められるアクションゲームを実行できるような性能を持っており、一般的な心理実験で問題になることはほとんどないでしょう。

13.8 ちょっとしたデータの整理

Builder の弱点はいろいろありますが、そのひとつに「実験記録ファイルのフォーマットが扱いづらい」というものがあります。各コンポーネントの開始、終了時刻がデフォルトで出力されるようになってからやたらと列数が多いこと、列名が長くて Excel などで表示すると冒頭部しか表示されないこと (slider.started, slider.stopped, slider.response, slider.rt,... と冒頭部が同じ列が続くときは特に)、ループが複数ある実験では空白のセルがやたらと多くなることなど、挙げればきりがありません。

データ処理は個人の好みが強く出るものだと思いますし、実験内容によってどのように処理するのが望ましいか大きく異なるので、「このように整理するとよい」という例を示すのは難しいですが、ここでは Excel にせよ R にせよ、他のソフトウェアへ引き継ぐことを前提に必要なデータを抜き出す例をひとつ挙げます。

まず、教示や練習試行など、分析の対象としない出力がたくさんあって、どこかのループに分析対象となる出力がある実験データから、分析対象部分だけを取り出してくるパターンです。slider という名前の Slider コンポーネントで反応を測定していて、刺激画像を表すパラメータ stim_image と、slider による評定値 (slider.response) と反応時間 (slider.rt) が分析対象とします。参加者を区別するために participant の情報も必

要でしょう。psyexp ファイルと同じフォルダに data と言うフォルダが出来ていて、そこに CSV 形式の実験記録ファイルが蓄積されているとします。この状況で、以下のコードを psyexp ファイルと同じフォルダに作成して PsychoPy Coder で実行すると、全参加者の反応をまとめたものが alldata.csv というファイルが作成されます。

```
# coding: utf-8
# #で始まる行はコメントです。 1 行目の # coding: はスクリプトの文字コードを表します。
# コード内に非 ASCII 文字がある場合は文字コードを指定する必要があります。
# psychopy の Coder は文字コードとして UTF-8 を使うので UTF-8 と宣言しています。
import pandas as pd
import glob

# colums に抜き出す列名をカンマ区切りで並べます。
# target_column が空欄の行は出力しません。分析の対象としたい列、
# たとえば参加者の反応が記録された列を指定するといいでしょう。
columns = ['participant','stim_image', 'slider.response', 'slider.rt']
target_column = 'slider.response'

# 実行フォルダ内にある data フォルダに含まれる拡張子.csv ファイルを読み込んで
# columns に挙げられた列のデータを alldata に追加していきます。
alldata = []

# glob は引数に合致するファイルのパスを順番に取り出す関数で、for と組み合わせると
# 取り出されたファイルを順番に処理していくことができます。
for datafile in glob.glob('data/*.csv'):
    # pandas の read_csv() で csv ファイルを開きます。
    df = pd.read_csv(datafile)
    # target_column が na でない (=空白でない) 行を抜き出して alldata に追加します。
    alldata.append(df.loc[~df[target_column].isna(), columns])

# alldata に追加されたデータを concat() でひとつに結合して、
# to_csv() で CSV ファイルに出力します。
pd.concat(alldata).to_csv('alldata.csv', index=False)
```

コメント (コード内の # から行末までの部分) を多めに入れておいたので、コメントを読んでいただければおおよその動作はわかると思います。columns と target_column を編集すれば他の実験にも使用できるでしょう。

実験内に 2 つ以上のループがあって、それぞれのループでの反応を出力しないといけない場合、どういった形でデータを整頓したいかによって話がかわってきますが、ループごとに別々のデータファイルになっていいなら、上記コード冒頭の import 文以降の処理全体を for 文で囲んで、columns、target_column と出力ファイル名を変更しながら繰り返せば手っ取り早いです。実験記録ファイルの読み込みに pandas というパッケージを使っているのですが、pandas を使うと条件別に平均を計算したりといった処理が簡単におこなえるので、興味がある人は pandas の使い方を勉強してください。