PsychoPy Builder で作る心理学実験

リリース *2.1*

十河宏行

12 月 18, 2018

目次

はじめに

2014 年 9 月に「PsychoPy Builder で作る心理学実験」の初版を公開してから約 1 年が過ぎました。その間に PsychoPy はさらにバージョンアップを重ね、初版の記述が不適切となってしまった点がいくつも出てきました。最大の変化はメニューやダイアログの日本語化ですが、それよりもむしろ「○○はバグがあって機能しない」とか「○○は実装されていない」といった記述が当てはまらなくなった点が放置されているのが問題と考えて、日本語環境での多くのバグが修正された 1.82.02 のリリースを機に改訂することにしました。十分な時間が確保できず、改訂の結果、作業の流れが不自然になってしまった部分もありますが、現行の 1.82.02 から見て明らか不適切な記述は排除することが出来たと考えています。主な変更点は以下の通りです。9 章の条件式によるコンポーネントの開始、終了の解説など、ご意見をいただいている点はいろいろあるのですが、時間的な事情により今回は加筆を見送りました。

- 1.79.01 および 1.80.00 に固有のバグの話題を削除しました。(全般)
- メニューやダイアログの日本語化に対応し、スクリーンショットを更新しました。(全般)
- 設定ウィザードとロケールについての解説を追加しました。(1 章)
- degFlat、degFlatPos、ブレンドモードに関する解説を追加しました。(2 章)
- 時間計測の精度について補足しました。(3 章)
- Loop の「使用する行」と「試行を繰り返す」に関する解説を追加しました。(4 章)
- RatingScale コンポーネントの解説とループに属さないルーチンの保存に関する記述を更新しました。複数行にわたる文字列の入力について補足しました。(6 章)
- Mouse コンポーネントの解説を更新し、マウスカーソルの位置を設定する方法の解説を追加しました。(8 章)
- Movie コンポーネントのバックエンドについての解説を追加しました。(9 章)
- 予約語一覧を更新し、ログファイルおよび PsychoPy 設定ダイアログについての解説を追加しました。(付録)

本書の実験で使用している画像ファイルや実験ファイルは、筆者の web ページ (http://www.s12600.net/psy/python/ppb/) でダウンロードできますのでご利用ください。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。

2015 年 9 月 25 日 十河 宏行

2016 年 8 月に PsychoPy 1.84.0 がリリースされました。Builder の操作について、以下の 2 点が大きく変化しましたので対応する部分の記述をアップデートしました。

- Static コンポーネントが標準で配置されなくなりました。(2 章など)
- ルーチン名が変更できるようになりました。(3 章)

あと、HTML で表示したときにマイナス記号が非常に判別しにくかったため、HTML 表示用のフォントを変更しました。引き続き、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。

2016 年 11 月 09 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記録するとか、数十種類の刺激を無作為な順番で 10 回ずつ提示するといった作業が求められることがありますが、こういった作業を手作業で行うのは困難です。PsychoPy を使うと、PC を用いて画像刺激や音声刺激を提示したり、反応時間を計測したり、刺激の提示順序を自動的に制御したりすることができます。

PsychoPy では、python というプログラミング言語を使って PC に指示を出します。本来 PC に指示を出すには機械語と呼ばれる言語を使わないといけないのですが、機械語は一見ただの数値の羅列で普通の人が読んだり書いたりすることはできません。プログラミング言語は、人間の言語 (たいてい英語なのですが) にちょっと似た雰囲気で読み書きできる言語で、インタプリタやコンパイラと呼ばれるソフトウェアを使って機械語に翻訳することができます。プログラミング言語や機械語で書かれた PC への指示をプログラムと呼びますが、python のプログラムはスクリプトと呼ばれることもあります。スクリプトって何？という疑問へのお答えはちょっと難しい話になるので **コンパイラとインタプリタ、そしてスクリプト** を参照してください。

本題に戻りまして、PsychoPy は python 用のライブラリをまとめたパッケージです。ライブラリとは、PC にいろいろな指示を出すうえでよく使われる手続きなどをまとめたものです。例えば、皆さんが使っている PC のソフトウェアでは「OK」とか「キャンセル」とか書かれたボタンが表示されて、その上にマウスカーソルを動かしてマウスのボタンをクリックするとボタンに応じた動作が行われるでしょう。この時、ソフトウェアを作る人は「マウスカーソルがボタンの上に乗っているか」、「マウスのボタンは押されているか」、「ボタンが押されたらどのような動作を実行するか」といった事を PC に処理させるプログラムを書かないといけません。しかし、これらの処理はどのようなソフトウェアのボタンでも共通しているので、この共通する動作をまとめて「ボタンに関する処理」という名前で呼び出せるようにしておけば、ソフトウェアのプログラムを書くときに「ボタンに関する処理を行う」と書くだけでまとめられた処理を全て PC に行わせることが出来ます。「ボタンに関する処理」の他にもたくさんの共通機能をまとめたものライブラリであり、ライブラリを用いることによって定番の処理を書くことに煩わされずに効率よく自分のプログラムを書くことが出来るのです。python では、ライブラリの個々のパーツをモジュールと呼び、モジュールの集合体をパッケージと呼んでいます。ライブラリ、モジュール、パッケージ、と耳慣れない用語が続きますが、とりあえずこれらは同じような意味だと思っておいていただいて問題ないと思います。

PsychoPy は心理学実験で使われる定番の処理、すなわち画面に刺激を描いたり、音声を鳴らしたり、反応時間を記録したりするための処理などをまとめた python のパッケージです。python のパッケージですから、利用するためには python のプログラムを書けなければいけません。python 以外の言語でプログラムを書いた経験がある人が python に乗り換えるのはそれほど難しくありませんが、プログラムを書いた経験がほとんどない人にはなかなか厳しいハードルです。そこで、PsychoPy にはウィンドウ内に刺激提示や反応計測を行うためのアイコンを並べることによってプログラムを書かずに実験を作成するアプリケーションが用意されています。このアプリケーションを PsychoPy Builder と呼びます。PsychoPy Builder はアイコンを並べて作成した実験を python のスクリプトに変換してくれます。自分で書いた python のプログラムを実行する場合は PsychoPy Coder と呼ばれるアプリケーションを使用します。以下、PsychoPy Builder のことを単に Builder、PsychoPy Coder のことを Coder と表記します。

本書では、Builder を使って心理学実験を作成する方法を解説します。非常に複雑な手続きの実験を行うには Coder を使って直接プログラムを書く必要があるかも知れませんが、工夫をすればさまざまな実験を Builder で行うことが出来ます。

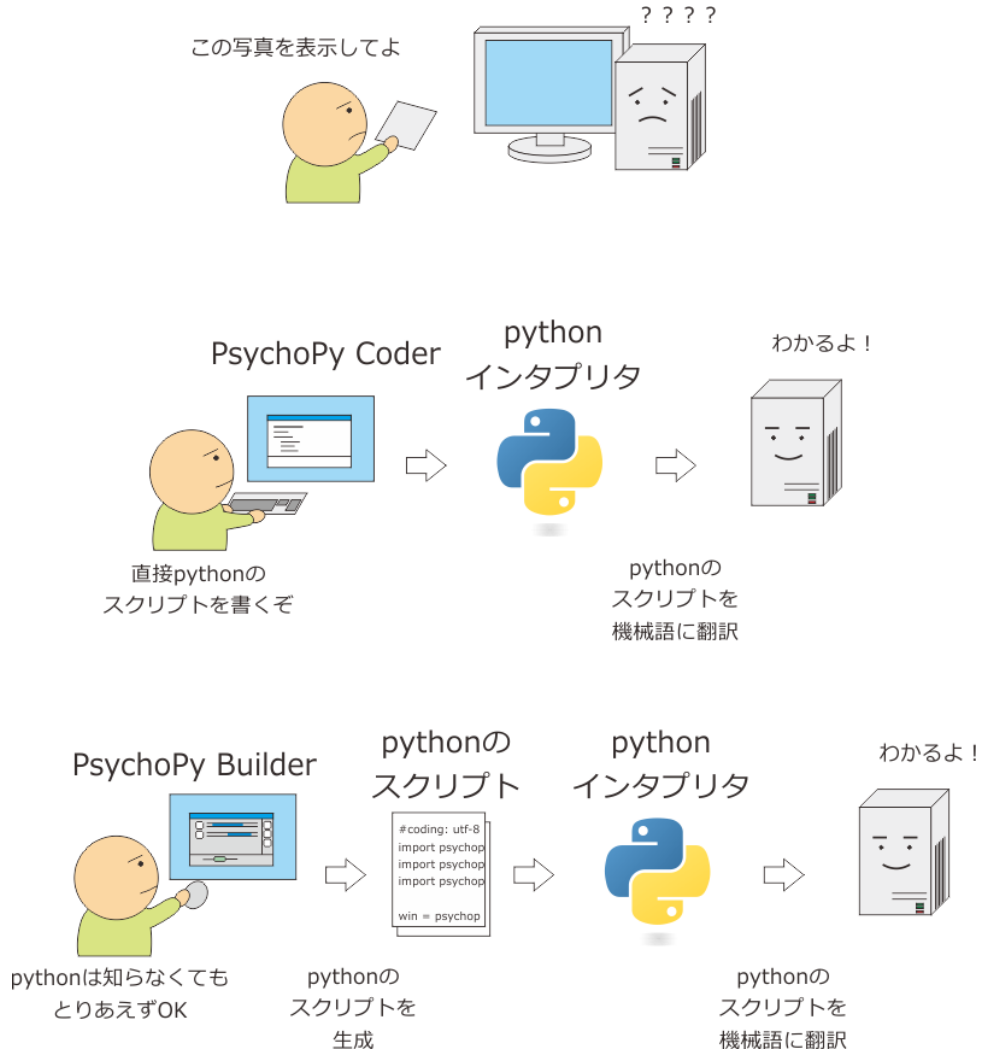


図 1.1　PsychoPy は python という言語を用いて PC に指示を出すためのソフトウェアです。python のスクリプトを直接書いて使う PsychoPy Coder と、グラフィカルなインターフェースで利用できる PsychoPy Builder があります。

1.2 PsychoPy をインストールしよう

PsychoPy をインストールする方法は複数ありますが、初心者向けで最も簡単なのは Standalone インストーラーを用いる方法です。PsychoPy を動かすには数多くの python パッケージが必要で、本来であれば PsychoPy をインストールする前に python 本体とそれらのパッケージのインストーラーをすべて web 上からダウンロードするなどして集めてひとつひとつインストールしていかなければいけません。python に慣れていない人にとってこれは大変な作業です。Standalone インストーラーは、これひとつで python 本体と PsychoPy の動作に必要なパッケージをまとめて一度にインストールしてくれるという優れものです。Standalone インストーラーは SourceForge という web サイトでダウンロードすることができます。URL は以下の通りです。

- http://sourceforge.net/projects/psychpy/files/PsychoPy/

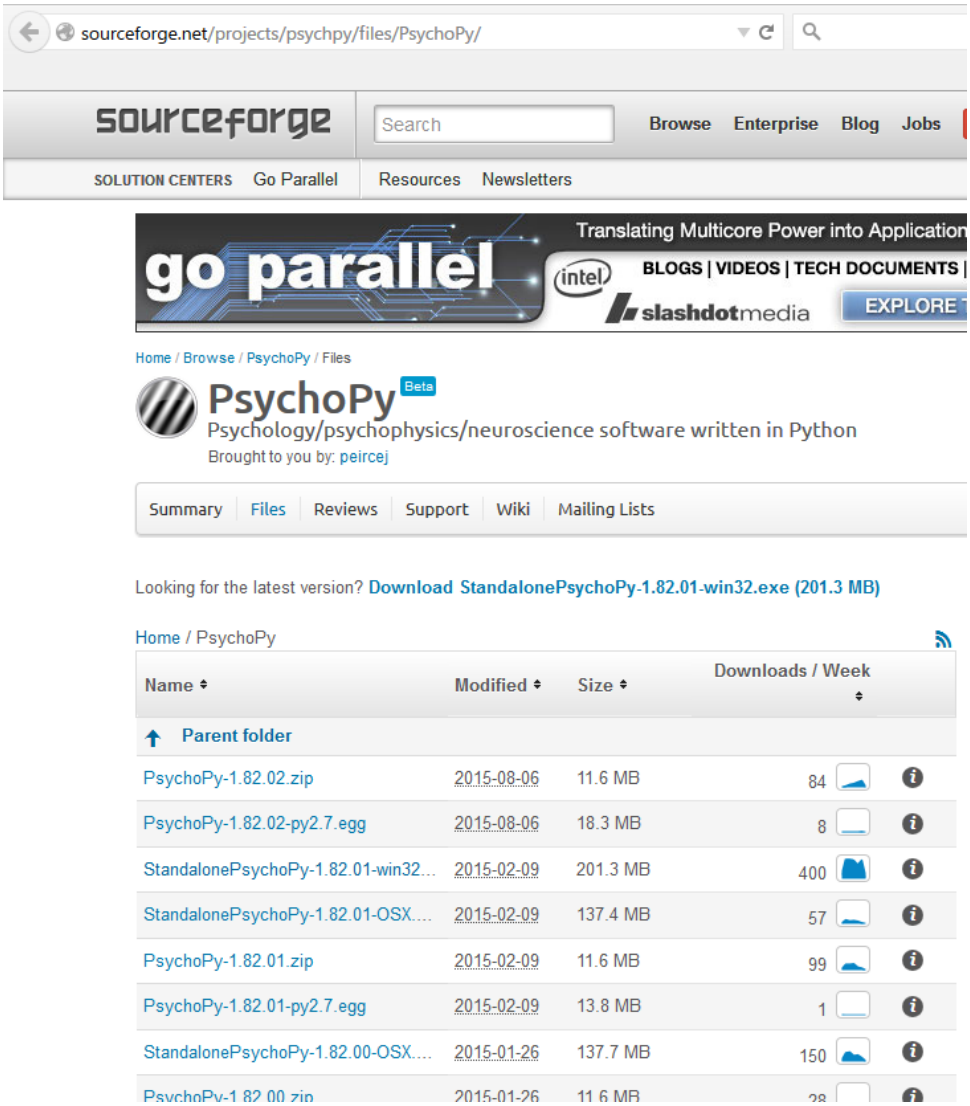


図 1.2　PsychoPy のダウンロード画面。本書ではバージョン 1.82.02 を使用しています。

図 1.2 はダウンロードページを開いた様子です。たくさんのファイルが並んでいますが、StandalonePsychoPy という名前のファイルが Standalone インストーラーです。後に続く数値 (例えば 1.82.01) がバージョンを示し

ていて、後に続く文字列が対応する OS を示しています。Windows を使用している方は win32、MacOS X を使用している方は OSX と書かれているファイルをダウンロードしてください。基本的には新しいバージョンのインストーラーをダウンロードすればいいのですが、最新バージョンはしばしば未解決のバグが含まれていますので判断が難しいところです。

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。インストールには管理者の権限が必要ですので注意してください。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。PsychoPy2 という項目が出来ているはずです (図 1.3)。このアイコンをクリックすると PsychoPy が起動します。

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 PsychoPy2 というグループができています。この中に PsychoPy2 というアイコンがありますのでこれを選択して起動します。

MacOS X 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。これでインストールは終了です。Ubuntu を使用している方はパッケージを利用してインストールできます。詳しくは *Linux* **で** *PsychoPy* **を使う** をご覧ください。PsychoPy のパッケージが用意されていない Linux 系 OS を利用している方や、すでに Windows 上で python.org 版の python を利用している等の理由で Windows の Standalone 版 PsychoPy を使いたくない方は、*Standalone* **インストーラーを使用しないインストール** *(***上級***)* を参照してください。

最初に PsychoPy を起動する時には、図 1.4 のようなスプラッシュ (タイトルロゴ) が表示された後に、「PsychoPy2 設定ウィザード」と上に書かれたダイアログが表示されます。OK を選択すると、PsychoPy が実行する PC に合わせた設定を自動的に行うためのプログラムが起動されます。正確な設定を行うために、OK を選択する前にダイアログに書いてあるようにメールソフトや web ブラウザ、(DropBox や Google Drive などの) ファイル同期サービスなどを終了させておきましょう。OK を選択した後、設定を行うために 10 秒ほど画面全体が灰色になったりちらちらと図形が画面に描かれたりしますので待ってください。

設定ウィザードが終了すると、図 1.5 のように結果の概要を示したダイアログが表示されます。動作テストの結果、基準値を下回るなどの問題があった項目が赤色で表示されています。ダイアログの OK ボタンをクリッ

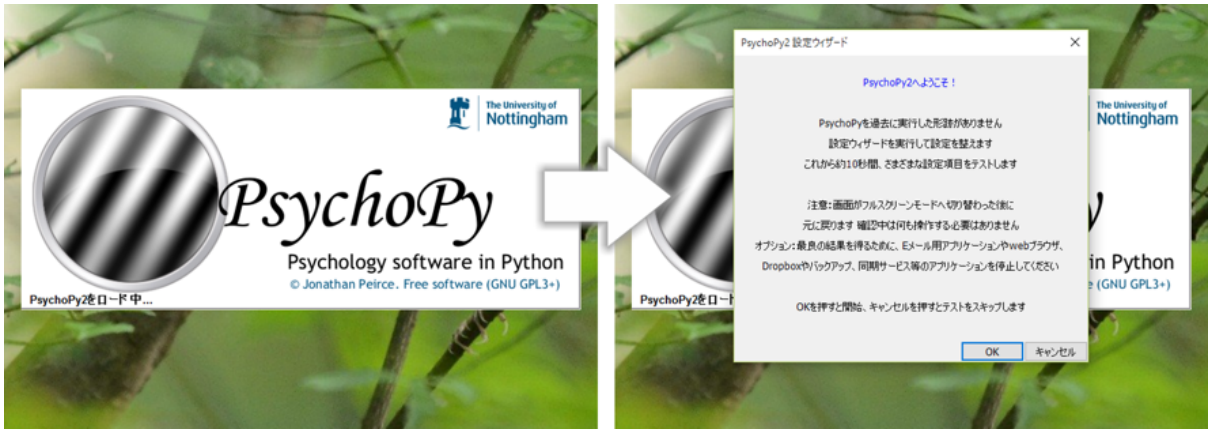


図 1.4　PsychoPy を起動すると左のようなスプラッシュ（タイトルロゴ）が表示されます。初回実行時のみ、右の「PsychoPy2 設定ウィザード」が表示されます。

クすると、ブラウザが開いて結果の詳細を見ることができます。図 1.5 では DropBox が動作していると警告されているほか、テスト中に描画タイミングが (標準偏差で)1.5 ミリ秒程度ばらついていたことや、スピーカーの出力に 0.2 秒ほど要したなどが指摘されています。

これらの指摘の意味がわかる人は各自で判断していただければよいと思いますが、意味が分からない人はこのような警告が表示されると不安になると思います。一般にどのくらいの値であれば問題ないという基準を示すのは難しいのですが、多くの実験では 1.5 ミリ秒程度の描画タイミングのばらつきは問題にならないはずです。スピーカーの出力が 0.2 秒遅れるのは、視覚刺激と聴覚刺激のずれが問題になる実験をしている人にとってはまずいですが、聴覚刺激を使わない人には問題にならないでしょう。「どのような実験をするか具体的な計画があるわけではないけど、とりあえず PsychoPy Builder の使い方を学んでみたい」という人は、あまり警告を気にせず使ってみるのが良いと思います。具体的な実験計画が出来てきたら、先行研究の手続きや結果をよく読んで、どの程度の精度が必要なのか、自分が使っている PC はその精度を達成できるのかを判断しましょう。

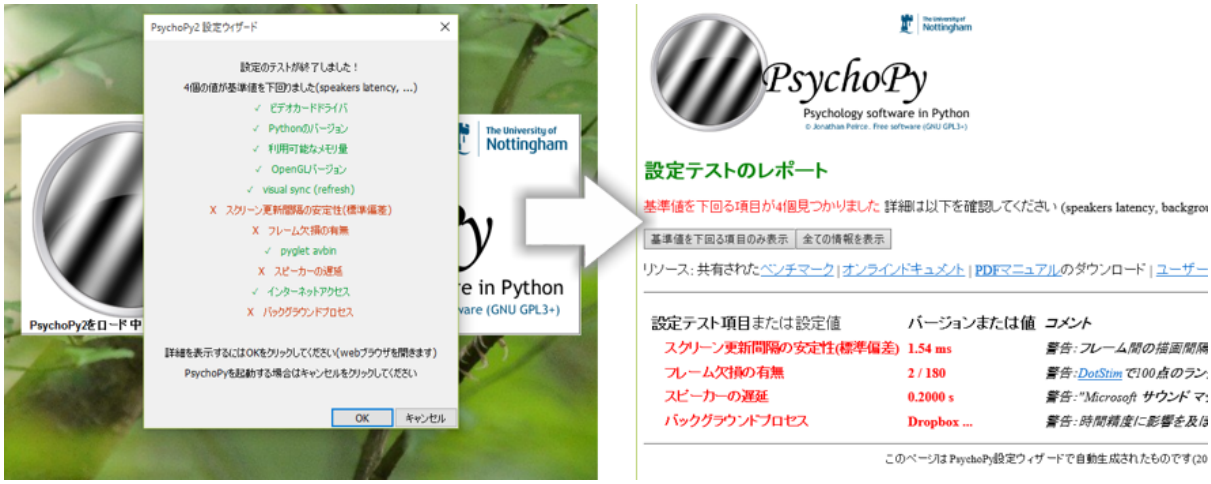


図 1.5　設定ウィザードが終了すると、概要を示したダイアログが表示されます。OK をクリックすると詳細を確認できます。

さて、設定ウィザードを閉じると、図 1.6 に示す二つのウィンドウが画面に出現します。図 1.6 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。もう一方のウィンドウ内が二分割されているウィンドウが Coder です。本書では主に Builder を使用します。

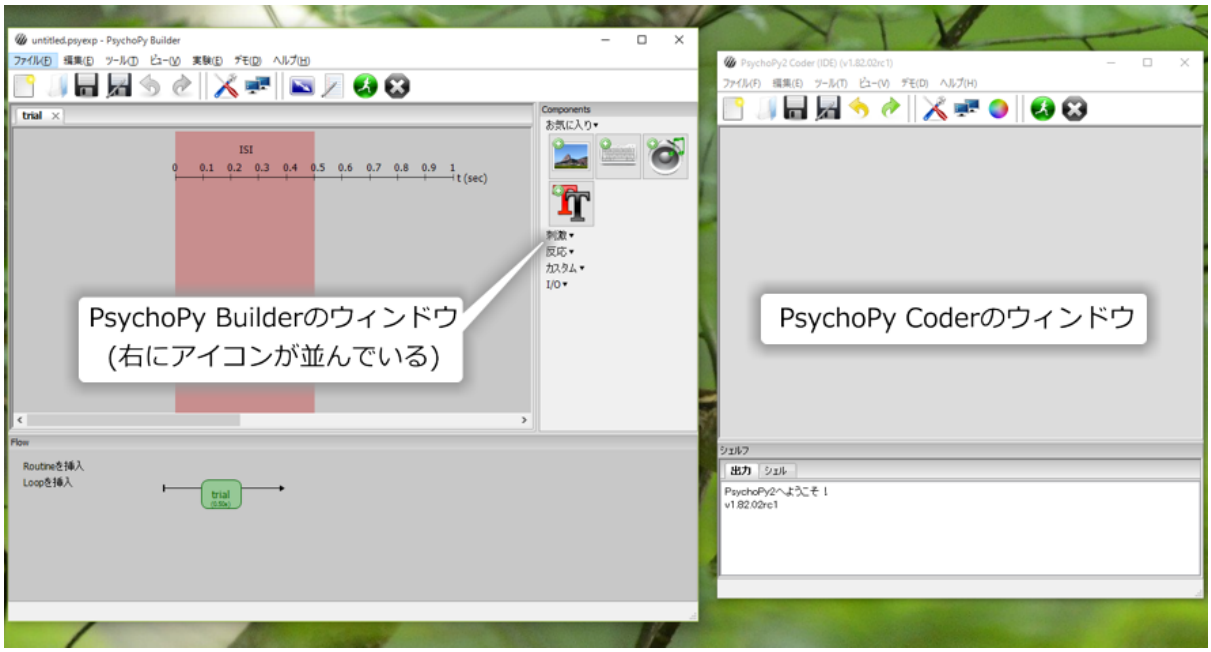


図 1.6　PsychoPy Builder(左) と Coder(右)。Builder はウィンドウ内が三分割されていて、右側にアイコンが並んでいます。Coder はウィンドウ内が上下に分割されています。

Builder の画面を閉じてしまったりして Coder だけしか表示されていない状態になった場合は、Coder ウィンドウ上部のメニューの「ビュー」から「Builder を開く」を選択すると Builder を開くことが出来ます。逆に Builder から Coder を開きたい場合には Builder ウィンドウのメニューの「ビュー」から「Coder を開く」を選択します (図 1.7)。

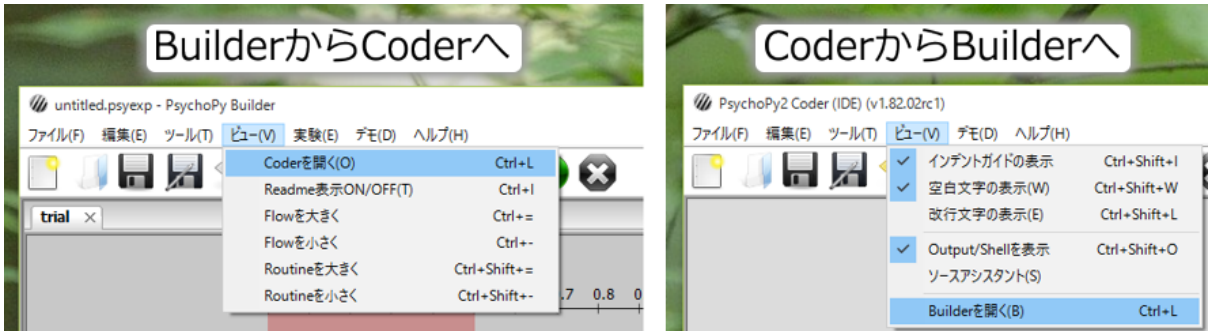


図 1.7　PsychoPy Builder と Coder のどちらか一方しか画面上に表示されていない場合は、メニューの「ビュー」から「Coder を開く」、「Builder を開く」を選択すると表示させることが出来ます。

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、万一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。 **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。

インストールしている PsychoPy より新しいバージョンの PsychoPy が公開されると、PsychoPy を起動してしばらくした後に、図 1.8 のようにアップデート通知のダイアログが表示されます。アップデートを行うと作成した実験が実行できなくなったり動作が変わったりすることがありますので、不用意にアップデートしないことをお勧めします。アップデートによる変更が小さい場合 (図の左のダイアログ) は、「このバージョンをスキップする」というボタンを押すことによってさらに新しいバージョンが公開されるまで通知ダイアログを表示させないようにすることが出来ます。

アップデートの内容を確認して、アップデートを行うことに決めた場合は、インストール済みのファイルを変更できる権限を持つユーザーで PsychoPy を起動する必要があります。詳しくは *PsychoPy* **のバージョンアップ** をご覧ください。

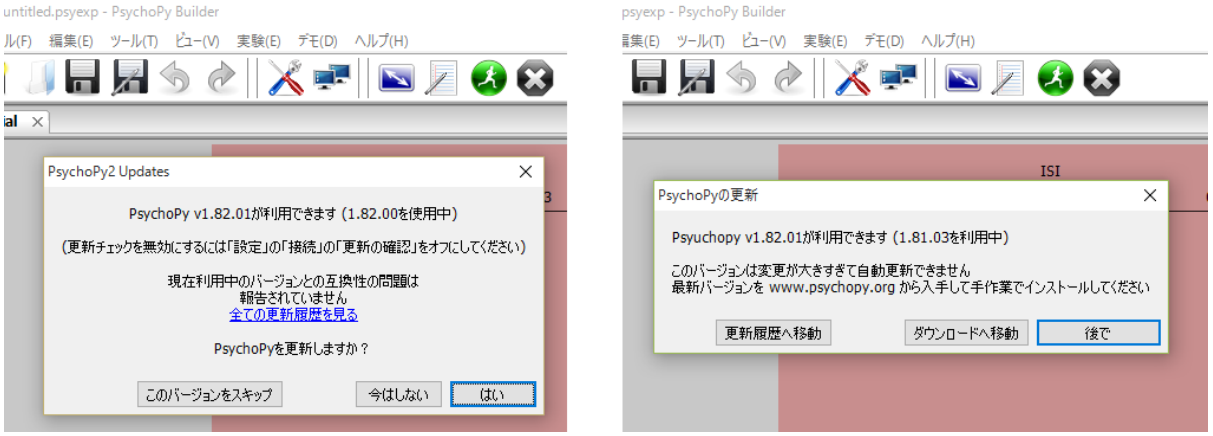


図 1.8　アップデート通知のダイアログ。通常は左のダイアログが表示されますが、アップデートによる変更が大きい場合は右のダイアログが表示されます。

さて、これで Builder が使用できる状態になりました。さっそく解説を始めたいところですが、その前に二つ確認してください。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出来ます。このテキストでも 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 を利用することも出来ます。LibreOffice は以下の URL でダウンロードすることが出来ます。

- 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 1.9 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。

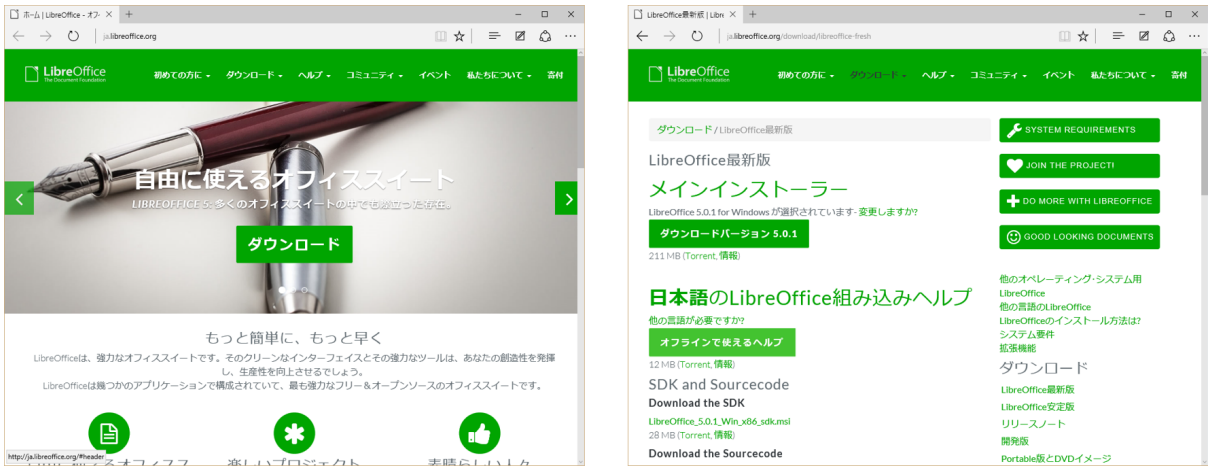


図 1.9　LibreOffice の日本語版 web ページ (左)。「LibreOffice をダウンロード」をクリックするとダウンロードページ (右) へ移動するのでメインインストールパッケージとヘルプパッケージをダウンロードします。

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 1.10)。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。Windows10 の場合は、エクスプローラーのウィンドウの上部の「表示」タブをクリックすると「ファイル名拡張子」という項目がありますので、そこにチェックをつけてください。Windows8.1 でも同様の手順で設定できます。

以上の作業が終了したら、次の章へ進んで実際に Builder を使ってみましょう。

チェックリスト

- PsychoPy を起動できる。
- Coder のウィンドウから Builder のウィンドウを開くことが出来る。
- Builder のウィンドウから Coder のウィンドウを開くことが出来る。
- 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 1.10　Windows10 で拡張子が表示されていない時の表示方法。

1.3 この章のトピックス

1.3.1 コンパイラとインタプリタ、そしてスクリプト

プログラミング言語で書かれたプログラムを実行する際に、毎回プログラミング言語を機械語に翻訳しながら実行する方法と、事前に機械語に翻訳したプログラムを作成しておいて、実行時には翻訳済みのプログラムを実行する方法があります。前者の毎回翻訳を行う翻訳プログラムをインタプリタと呼び、後者の事前に翻訳済みプログラムを作成する翻訳プログラムをコンパイラと呼びます。

インタプリタの利点は、プログラムを書いたら直ちに PC に実行させることが出来る点です。少し書いては書き直すといった試行錯誤をする時にはとても楽です。その代り、実行する度に PC は翻訳作業を行いますので、実行速度がやや遅いという欠点があります。コンパイラは逆に、プログラムを書いたらコンパイラを使って翻訳する作業を毎回行う必要があります。また、人間が書いたプログラムのファイルの他に翻訳後のプログラムのファイルが出来るのでファイルの整理が面倒です。その代り、実行する時にはすでに翻訳済みなので高速に実行することが出来ます。

多くのプログラミング言語では、インタプリタとコンパイラのどちらの方法が用いられるかが決まっています。例えば C 言語という言語ではコンパイラを使って事前に翻訳済みプログラムを作成することがほとんどです。一方、python は一般にインタプリタを使って実行します。

インタプリタを使う言語のように、人間が書いたプログラムを（コンパイルなどの作業なしに）直接実行できる言語をスクリプト言語と呼ぶことがあります。そして、スクリプト言語で書かれたプログラムをスクリプトと呼びます。この意味では python はスクリプト言語であり、python のプログラムはスクリプトです。

1.3.2 Linux で PsychoPy を使う

Linux で PsychoPy を使うことを検討している方向けの発展的な話題です。使用している Linux のディストリビューションが Debian や Ubuntu であれば、PsychoPy のパッケージが用意されているので、通常のパッケー

ジと同じ手順でインストールすることが出来ます。例えば apt コマンドを使用する場合は、sudo を利用できるユーザーでログインして端末から以下のように入力します。

```
sudo apt-get install psychopy
```

PsychoPy のパッケージが用意されていないディストリビューションで利用する場合は、*Standalone* **インストーラーを使用しないインストール** *(上級)* で述べる方法で python のパッケージとしてインストールすることが出来ます。

1.3.3 Standalone インストーラーを使用しないインストール (上級)

すでに python を利用している人向けの発展的な話題です。Standalone インストーラーはとても便利なのですが、python の実行環境を丸ごとインストールするので、python をインストール済みの PC に Standalone インストーラーで PsychoPy をインストールすると python の実行環境が PC の中に二重に存在することになってしまいます。このような事態を避けたい場合は、インストール済みの python へパッケージとして PsychoPy を導入することも出来ます。

例えばパッケージの管理に pip を使用しているのであれば、以下のコマンドを実行すれば PsychoPy をパッケージとしてインストールすることが出来ます。

```
pip install psychopy
```

ただし、実行に必要なパッケージが一部自動的にはインストールされませんので、PsychoPy 公式ページの Dependency(下記 URL) を確認しながら足りないパッケージをひとつひとつインストールしていく必要があります。

- http://www.psychopy.org/installation.html

必須パッケージのリストは随時更新されます。2015 年 9 月の時点では以下のパッケージが挙げられています。

共通 python 本体 (2.6 または 2.7), avbin, setuptools, numpy, scipy, pyglet (1.1.4), wxpython (2.8), PIL (Python Imaging Library), matplotlib, lxml, openpyxl, pyo

Windows pywin32, winioport, inpout32, inpoutx64

Linux pyparallel

PsychoPy の機能をすべて利用するためにはここに挙げられていないパッケージも必要であり、実行時のエラーメッセージを確認しながらパッケージを導入するなどしなければいけません。

1.3.4 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 1.11 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」というタブに「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。

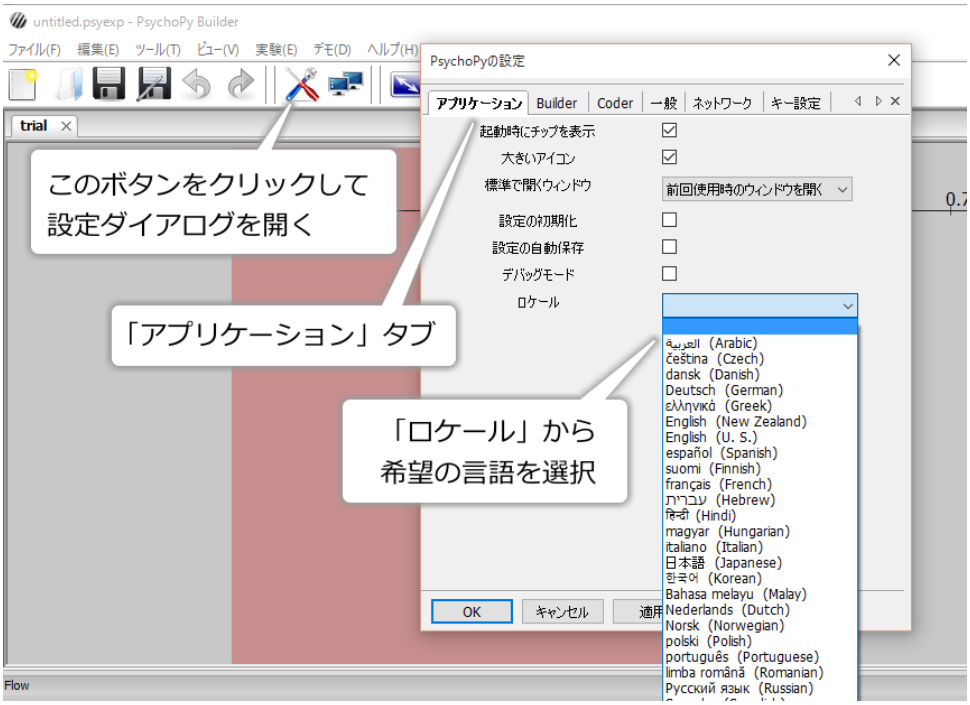


図 1.11　ロケールの設定。

1.3.5 PsychoPy のバージョンアップ

本文中で述べたとおり、PsychoPy の新しいバージョンが公開されると起動時にアップデート通知ダイアログ (図 1 7) が表示されます。このダイアログで「はい」ボタンをクリックすると、図 1.12 のダイアログが表示されます。「自動更新」を選択すると、PsychoPy はインターネット上から最新バージョンを自動的にダウンロードしてアップデートを試みます。しかし、筆者の経験上、「自動更新」は失敗することがあるので次の「以下の Zip ファイルを使用」を利用することをお勧めします。「以下の Zip ファイルを使用」では、ユーザーが自分で PsychoPy の配布サイトから最新版の zip ファイルをダウンロードしてアップデートを行います。Standalone インストーラーではなく、拡張子が zip のアップデートを選択するのがポイントです。Browse ボタンを押してダウンロードした zip ファイルを選択すると、アップデートが行われます。古いバージョンの zip ファイルをダウンロードして利用することによって、以前のバージョンに戻すことも可能です。ただし、バージョン間でアップデート不能な大きな変更がなかった場合に限ります。

どちらの方法でアップデートを行うにせよ、注意すべき点がひとつあります。Windows で Standalone インストーラーを使った場合や、Ubuntu のパッケージを使ってインストールした場合は、管理者の権限があるユーザーでなければアップデートを行えません。Windows の場合は PsychoPy のアイコンを右クリックして「管

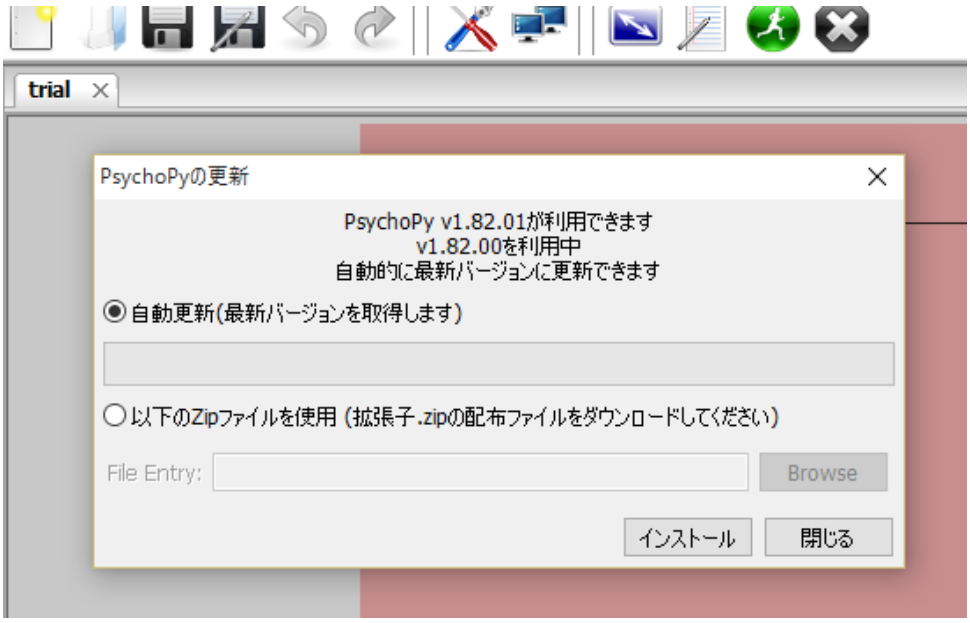


図 1.12　PsychoPy のアップデートのダイアログ。

理者として実行」して PsychoPy を起動してください。Ubuntu の場合はターミナルを起動して sudo psychopy と入力すると管理者の権限で PsychoPy を起動できます。

図 1.8 右に示したダイアログが表示された場合は、この方法でのアップデートが困難なほど大きな変更が行われます。多くの場合、PsychoPy 以外のパッケージもバージョンアップが必要だったり、さらに追加のパッケージが必要だったりしますので、配布サイトへアクセスして Standalone 版インストーラーをダウンロードしてインストールするのがお勧めです。

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

なにはともあれ、まずは刺激を 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と呼びます。痛みのペイン (pain) ではなく、窓枠にはめる一枚一枚のガラスのペイン (pane) です。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。

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」、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。

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、四角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」、「高度」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。

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」という作業が出来ました。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

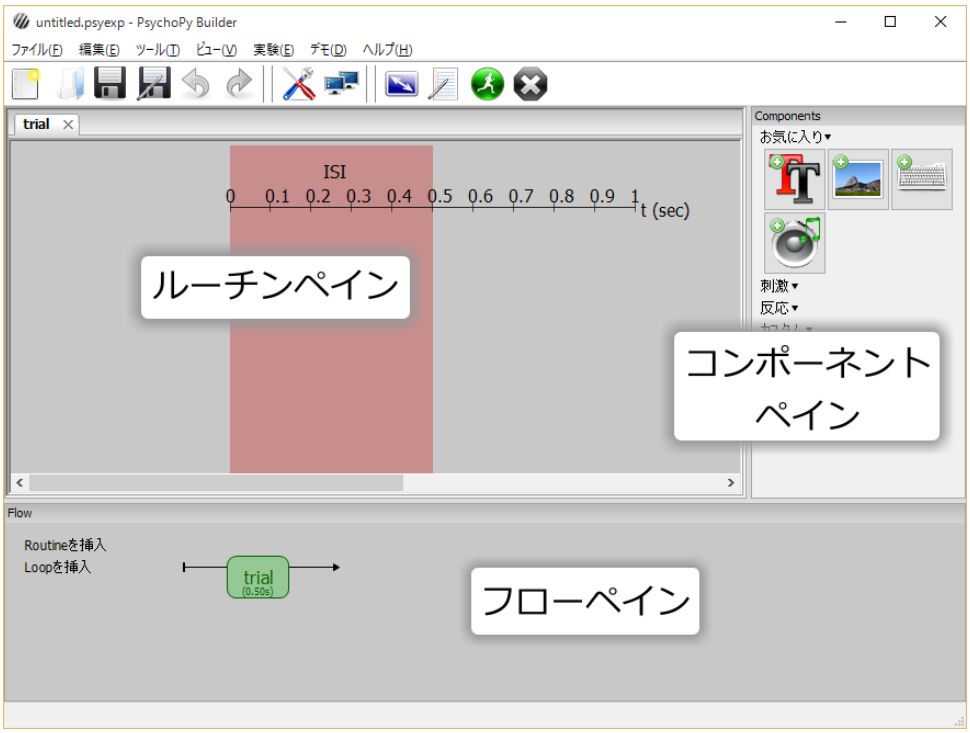


図 2.1　Builder の 3 つのペイン。コンポーネントペインから刺激等を選んでルーチンペインに配置し、フローペインでルーチンの実行順序を指定します。PsychoPy 1.84.0 以降を使用している場合はルーチンペインの赤い領域がありません。

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、四角が描かれたアイコン) をクリックします。

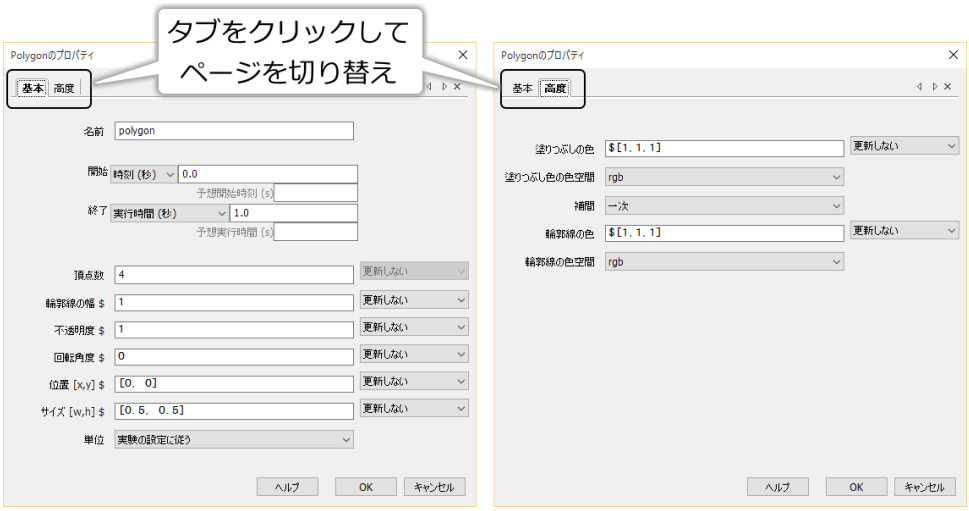


図 2.3　Polygon コンポーネントのプロパティを設定するダイアログ。左上のタブをクリックすることでページを切り替えられます。

くことによって刺激を作成します。

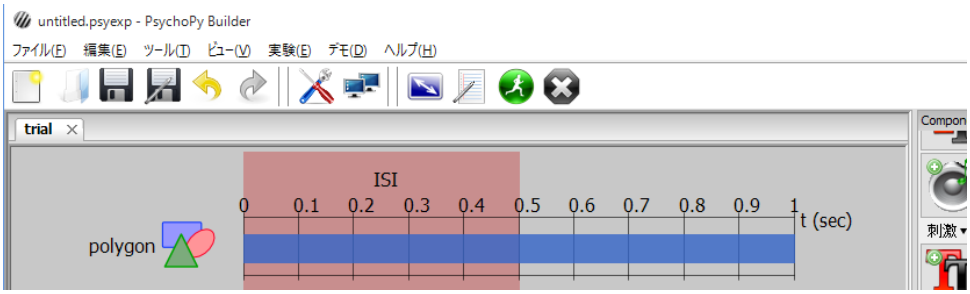


図 2.4　コンポーネントをルーチンに配置した状態。

では、この Polygon コンポーネントをひとつ配置することによって、どのような刺激が表示されるのかを見てみましょう。刺激を見てみるためには、実験を保存して実行する必要があります。実験はウィンドウ上部に並んでいるボタンのうち、緑色の円の中で人が走っているボタンをクリックすると実行できます。以後、このボタンを実験実行ボタンと呼びます (図 2.5)。実験を作り始めてまだ一回も実験を保存していないので、ここで実験を保存するダイアログが表示されます。ここでは exp01.psyexp という名前で保存しておきます (図 2.6 上)。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、それらについてはこの章の「Builder が作成するファイルを確認しよう」をご覧ください。

実験を保存すると、続いて画面上に先ほど保存したファイル名がタイトルについた小さなダイアログが表示されます (図 2.6 下)。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。

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

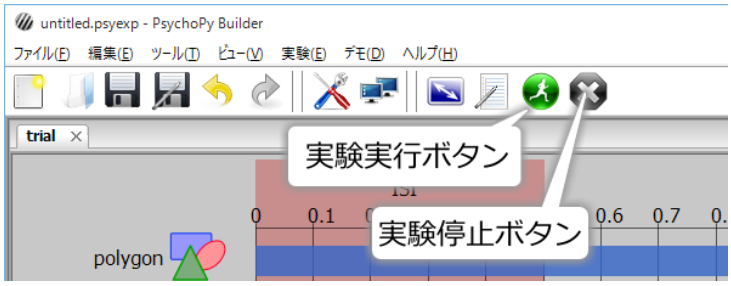


図 2.5　実験実行ボタンをクリックして実験を実行します。実験が開始されると実験停止ボタンが赤色になります。赤色状態の実験停止ボタンを押すと実験を強制終了出来ます。

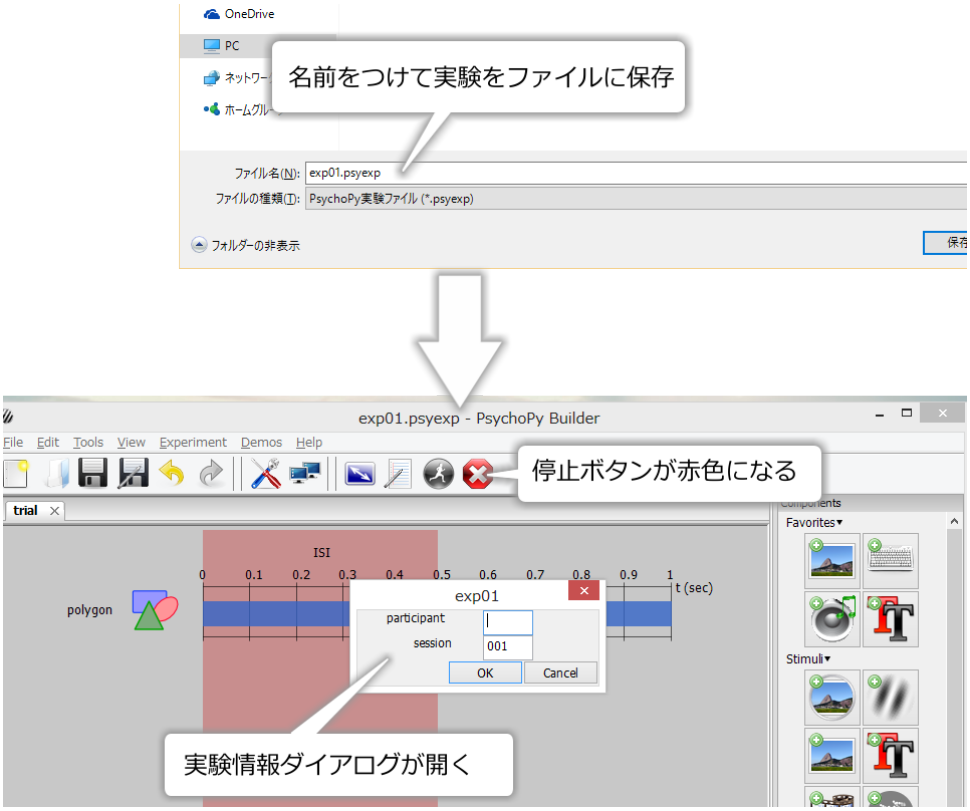


図 2.6　実験を保存した後、少し待つと実験情報ダイアログが表示されます。

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、とりあえず何も入力せずに OK をクリックしてください。すると画面全体が灰色一色になり、少し間をおいて白い長方形が表示されます (図 2.7 上)。その 1 秒後に灰色の画面から元の Builder のウィンドウが表示されている画面に戻り、図 2.7 下のようなダイアログが表示されます。これは「ログ」と呼ばれるもので、実験実行状況に生じた問題が簡単に示されています。赤色の文字がエラー、緑色の文字が警告です。このエラーについては後の章で少しずつ触れます。とりあえず 図 2.7 上のような長方形が表示されたら、実験は無事に実行されたと考えてください。これで私たちが作った「最初の実験」が無事終了しました。

さて、実際の実験ではもっといろいろな図形や写真を表示したりしないといけませんし、実験参加者の反応を計測することも出来ないといけません。それらの方法を詳しく解説する前に、基本的な操作を二つ紹介しておきましょう。まず、ルーチンペインに配置している Polygon コンポーネントのアイコンにマウスカーソルを動

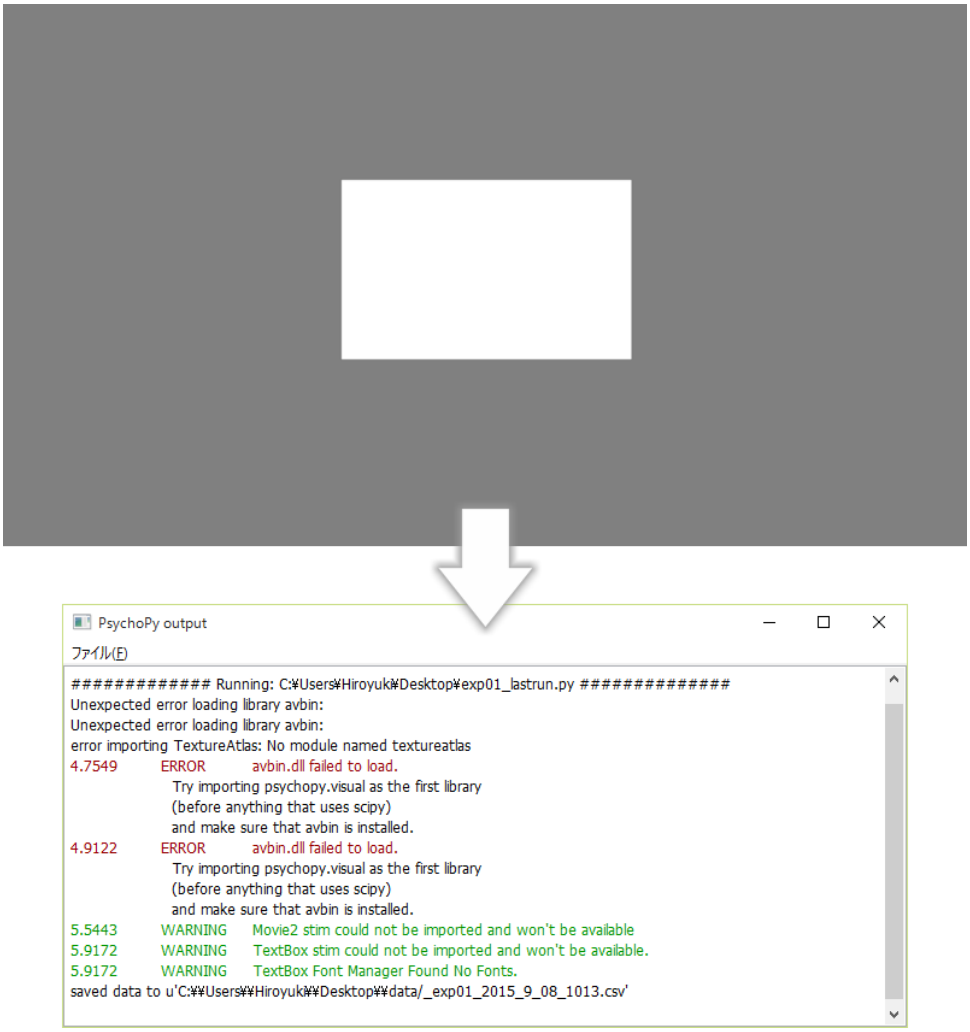


図 2.7　実行すると上図のように灰色の画面に 1 秒間白い長方形が表示されます。終了すると下図のようなログが表示されます。

かして左ボタンをクリックしてみてください。すると先ほどルーチンペインに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 (図 2.8)。ここでダイアログの **塗りつぶしの色** という項目にキーボードを使って black と入力してダイアログ右下の OK をクリックし、もう一度実験を実行してみましょう。すると、今度は黒色に塗りつぶされた長方形が表示されます。このようにプロパティを編集することによって、刺激の色や大きさを変更することが出来ます。配置済みのコンポーネントのプロパティ設定ダイアログを開く操作を覚えておきましょう。なお、本書ではこれ以後、プロパティ名を **塗りつぶしの色** のように太字のフォントで表記します。

もうひとつの基本操作は、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 2.9 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るはずです。

なお、PsychoPy 1.78.00 から 1.83.04 の PsyhcoPy を使用している場合はルーチンペインに赤い領域があるはずですが、これは Static コンポーネントというコンポーネントの一種です。これらのバージョンの PsyhcoPy

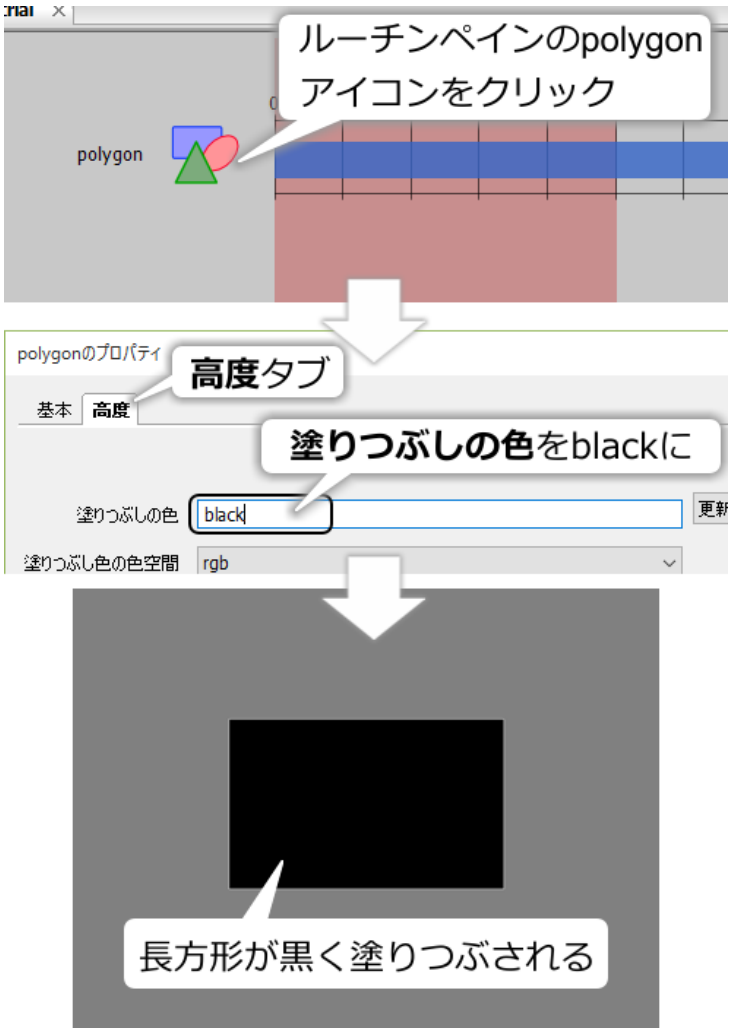


図 2.8　ルーチンペインのコンポーネントのアイコンをクリックすると、すでにルーチンに配置したコンポーネントのプロパティ設定ダイアログが開きます。 **塗りつぶしの色** という項目を選択してキーボードから black と入力して実験実行ボタンを押して実験を開始すると、黒色に塗りつぶされた長方形が表示されます。

では Builder で新しい実験を作成した時に自動的に Static コンポーネントが配置されます。しかし、本書では第 9 章まで Static コンポーネントを使用しませんので、削除しておきましょう。赤い領域上にマウスカーソルを移動させて右クリックすると、Polygon コンポーネントを削除した時と同様にポップアップメニューが出現するので、削除を選択してください。

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出来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

- ルーチンペインにコンポーネントを配置できる。
- 作成した実験を実行できる。

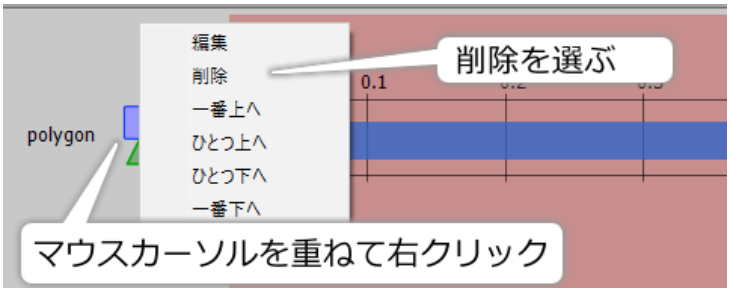


図 2.9　ルーチンペインのコンポーネントのアイコン上でマウスを右クリックするとメニューが表示されます。削除を選択するとコンポーネントを削除できます。

- コンポーネントのプロパティ編集ダイアログを開くことができる。
- コンポーネントをルーチンから削除することができる。
- 実験を保存したファイルの拡張子を答えられる。

2.2 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ください。Static コンポーネントはこの章では使用しないので、まだ削除していない人は削除しておきましょう。

Polygon コンポーネントのプロパティ設定ダイアログを表示すると、下の方に **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、プロパティ設定ダイアログの下の方にある **単位** という項目で指定します。 表 2.1 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 28.6cm のモニターを例として示したのが 図 2.10 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。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。

少し複雑なのが norm と height で、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

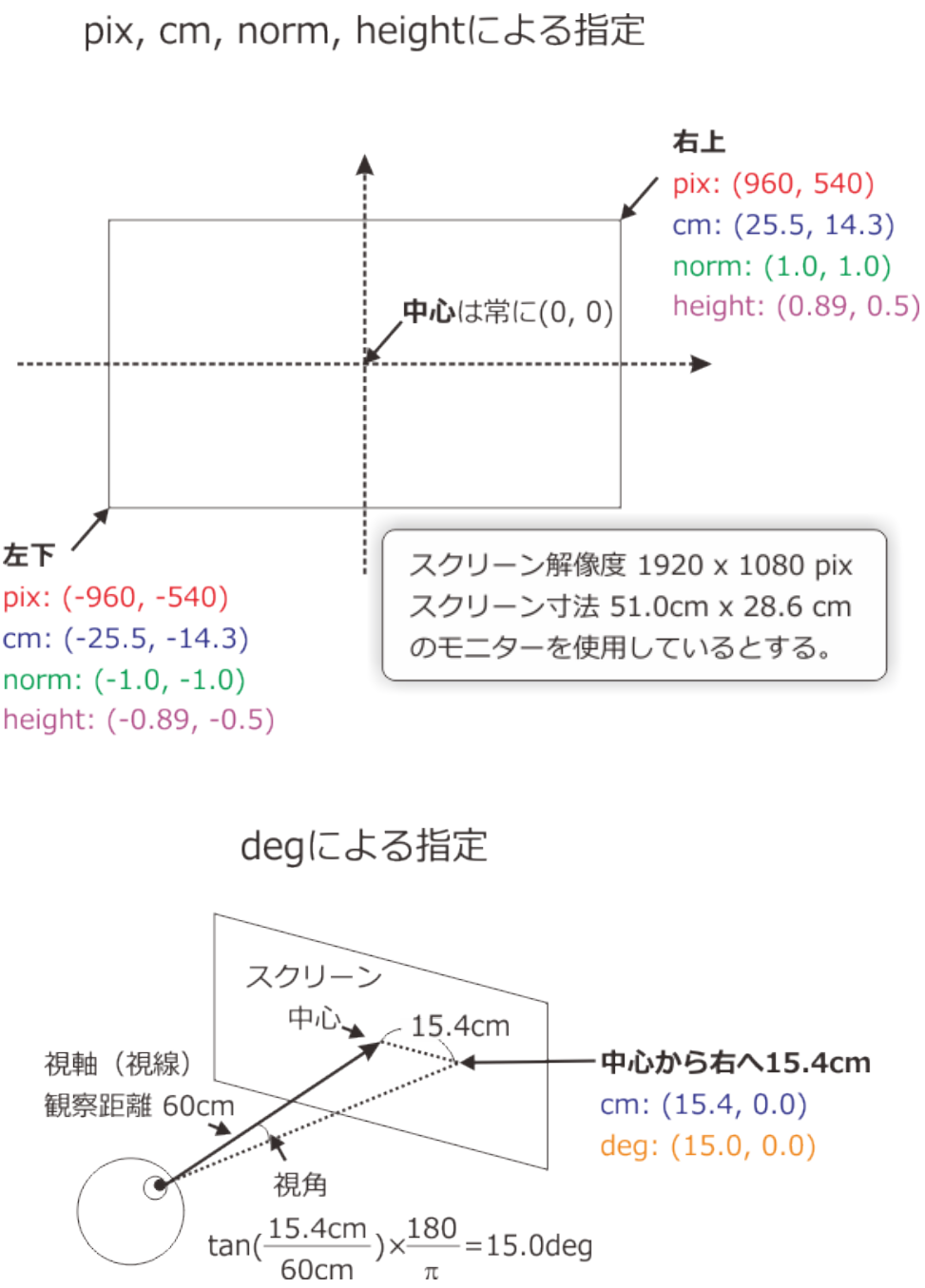


図 2.10　PsychoPy で使用できる単位。

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 (後述)。一方、height で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 2.10 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 2.10 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 **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 *PsychoPy* **における視角の計算について** をご覧ください。

表 2.1　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。実験設定ダイアログから指定する必要があります。
degFlat	deg と同様ですが、deg よりも正確に計算します (*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 (*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」を選んだ場合は 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 2.11 上のようにプロパティ設定ダイアログの **単位** を pix にしてください。ここで **単位** を間違える

と正しく表示されませんので、必ず忘れずに pix に設定してください。 **単位** の設定間違いは非常によくある「うっかりミス」です。そして、 **サイズ [w, h] $** の値を [200, 100] にしてみましょう。数値の両脇の角括弧やカンマを忘れずに入力してください。 **サイズ [w, h] $** の w と h はそれぞれ width と height ですから、[200, 100] と入力すれば、幅 200pix、高さ 100pix 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 2.11 下のようにスクリーン中央に横幅が高さのほぼ 2 倍の長方形が表示されるはずです。

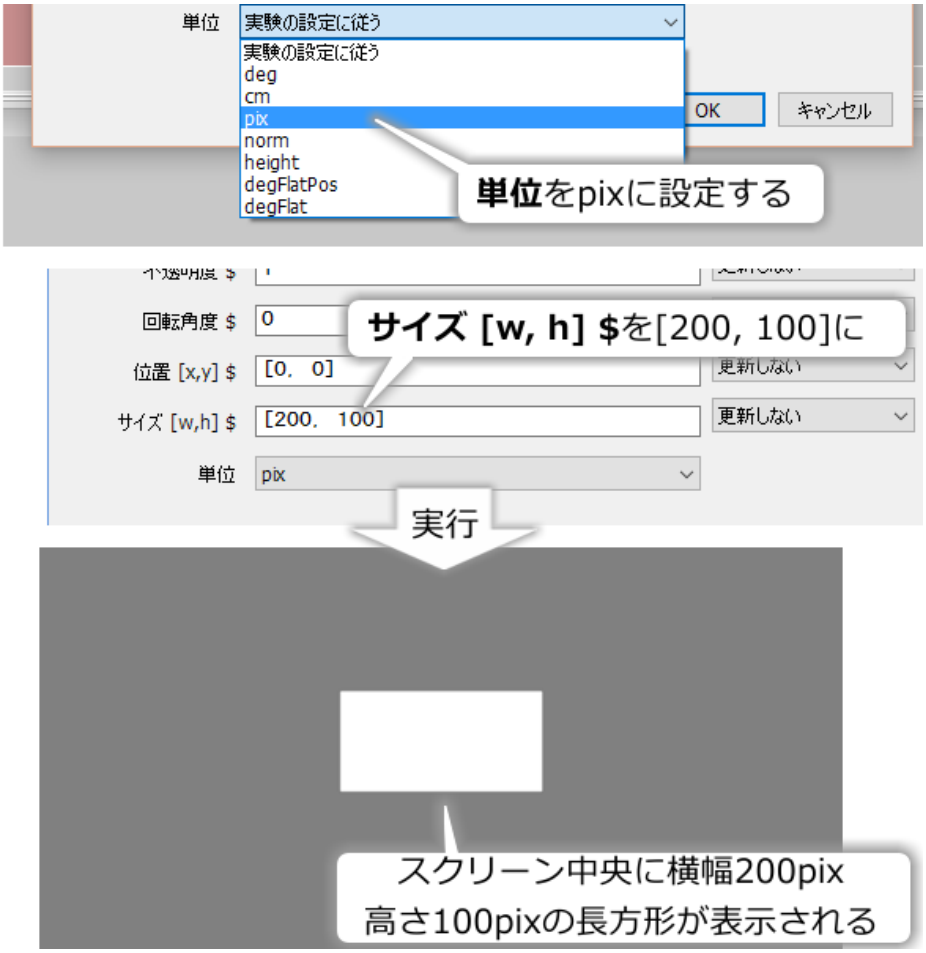


図 2.11　サイズの設定。Polygon コンポーネントを横幅 200pix、高さ 100pix に設定しています。

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 *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 2.12)。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、うまく並べられずに隙間が出来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「高度」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

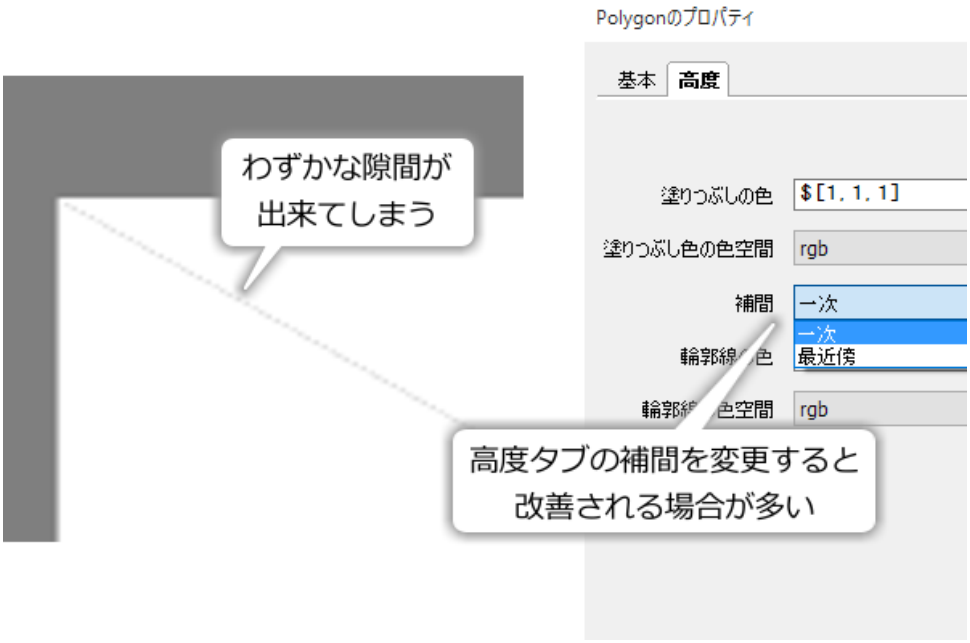


図 2.12　Polygon コンポーネントで長方形を表示すると細い線が見える事があります。「高度」タブの **補間** を変更すると多くの場合問題が解消されます。

ロパティ設定ダイアログを開いて、 図 2.13 上のように **位置 [x, y] $** に [100, 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、[100, 0] はスクリーン中央から右へ 100pix 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 2.13 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。

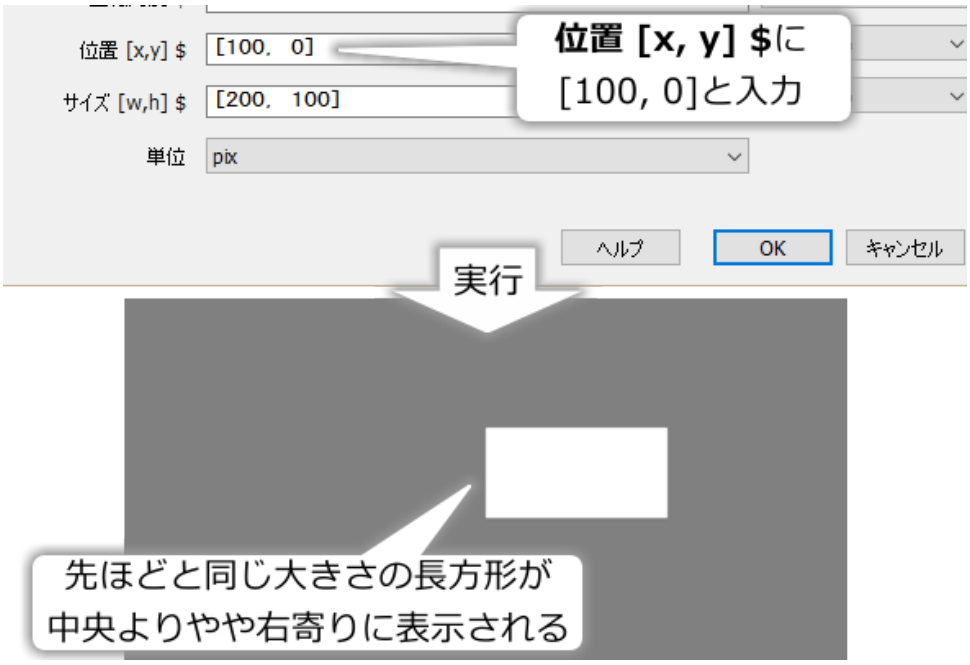
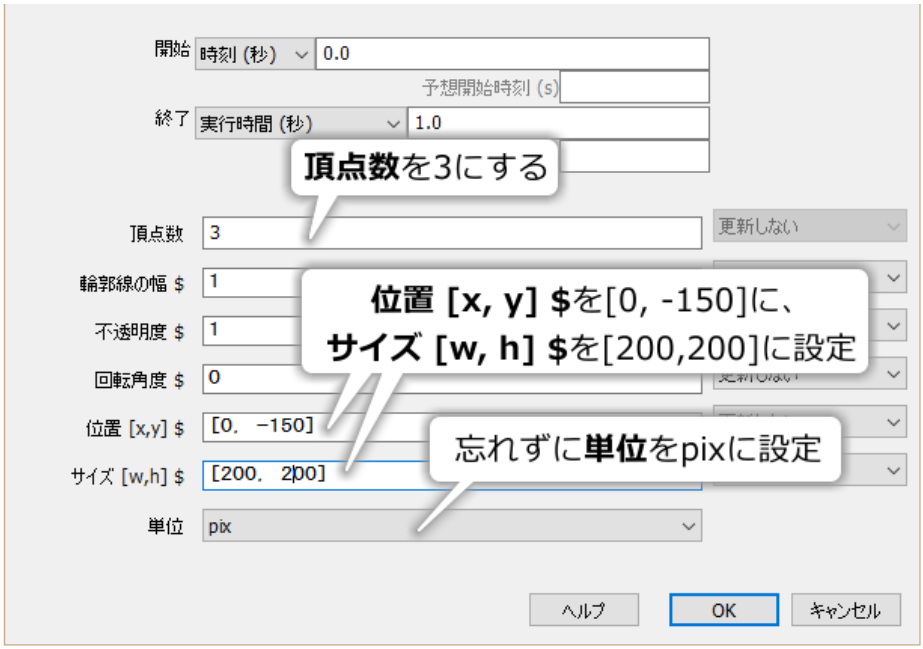


図 2.13　**位置 [x, y] $** に [100, 0] を設定すると長方形が右寄りに表示されます。

でも、 図 2.13 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、本当に 100pix 右に寄っているのかどうかの判断は困難です。それに、いったい **位置 [x, y] $** は長方形のどの部分を (100, 0) の位置に合わせているのでしょうか。これらの点を確認するために、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。



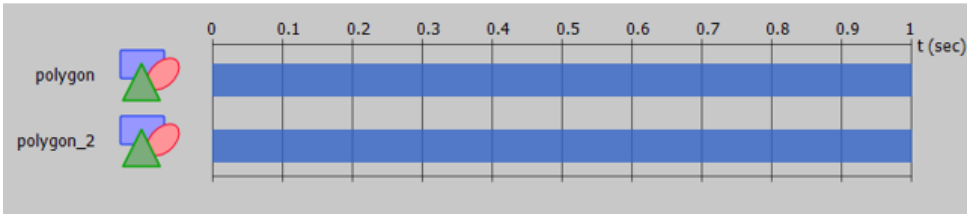


図 2.14　Polygon コンポーネントを追加し、頂点数を 3 にして位置と大きさを設定します。コンポーネントが追加されると下の図のようにルーチンペイン上に複数のアイコンが並びます。

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。 図 2.14 のように **頂点数** に 3 と入力してください。続いて **位置 [x, y] $** と **サイズ [w, h] $** も 図 2.14 と同じ値を入力してください。そして、忘れずに **単位** を pix にしましょう。入力が出来たら実験を実行してください。 図 2.14 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 2.14 の右です。 **位置 [x, y] $** は図形の左右端の中点、上下端の中点に対応しています。長方形は幅が 200pix で中心は右へ 100pix 移動していますので、長方形の左端の X 座標は 0pix です。また、長方形の高さが 100pix で上下方向へは移動していませんから、左下の頂点の Y 座標は高さ 100pix の半分に相当する 50pix だけスクリーン中心より下にあります。ですから、長方形の左下の頂点の座標は (0, -50) です。一方、三角形は高さが 200pix ですから、上側の頂点は三角形の中心から 100pix 上

に位置するはずです。三角形の **位置 [x, y] $** は [0, -150] に設定したのですから、上側の頂点の座標は (0, -50) となり、長方形の左下の頂点の座標と一致します。

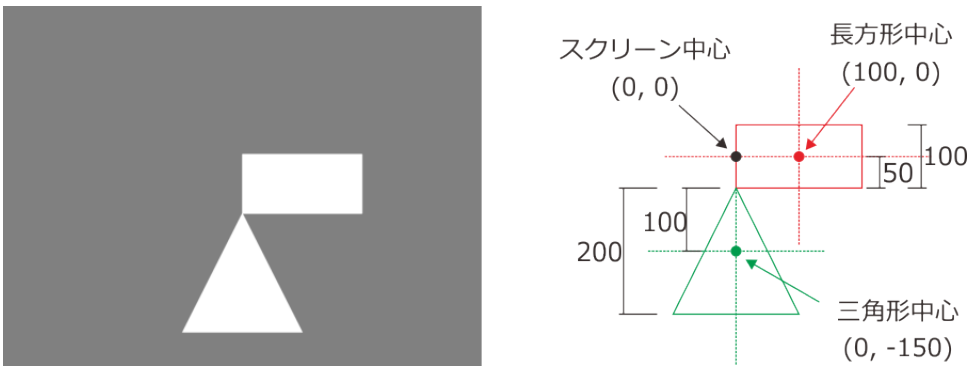


図 2.15　三角形の Polygon を追加した結果。 **位置 [x, y] $** が図形の左右端と上下端の中点に対応していることがわかります。

なお、ここで 図 2.14 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 200 の正三角形の高さは $200/\sqrt{3} \times 2 =$ 約 230.9 ですから、正三角形を表示するには **サイズ [w, h] $** を (230.9, 200) としなければいけません。 **位置 [x, y] $** が指し示す位置が三角形の重心と一致してない事にも注意する必要があります。

なお、「 **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 **頂点数** が 5 以上の時だけは例外的に、 **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。この点も注意してください。

ここでは単位に pix を指定した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 pix、もうひとつのポリゴンは cm 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

- Polygon コンポーネントを用いて多角形を表示できる。
- 多角形の頂点数を変更できる。
- 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。
- pix を単位に指定して、 **位置 [x, y] $** と **サイズ [w, h] $** に適切な値を入力し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出来る。
- 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。

2.3 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出来ます。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 30 度、90 に設定すると 90 度回転します。負の値も指定できますので、-60 を指定すると反時計回りに 60 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。 図 2.16 に縦長の三角形を 30 度、90 度、-60 度回転させた例を示します。

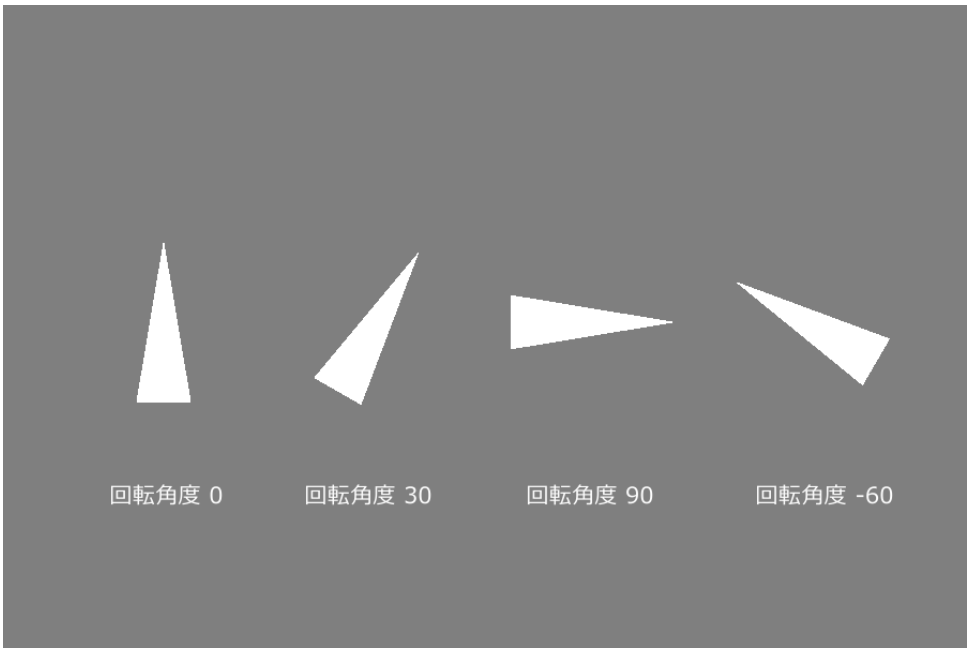


図 2.16　**回転角度 $** の指定による図形の回転。

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、 **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 2.17 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 の使用を検討しましょう。

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 $** 、 **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。

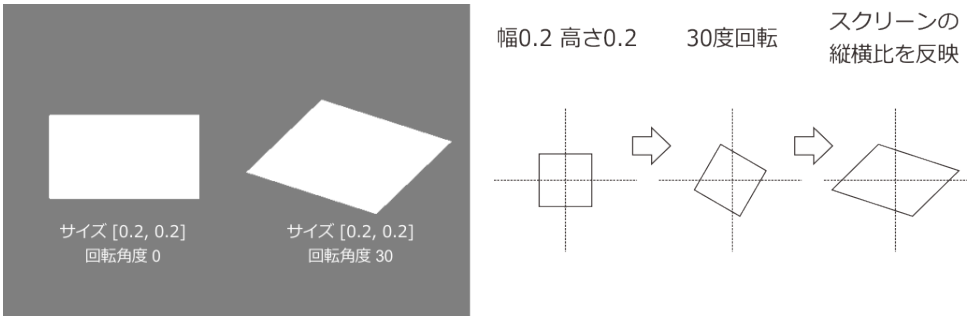


図 2.17　**単位** に norm を指定した場合の回転。回転してからスクリーンの縦横比を反映させるため図形が歪みます。

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出来る。
- 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。

2.4 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 2.18 左のアイコン) 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。すでにおなじみの **位置 [x, y] $** 、 **回転角度 $** 、 **回転角度 $** 、 **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 **文字の高さ $** 、「高度」タブの **反転** 、 **折り返し幅 $** を使ってみましょう。

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 **単位** を pix に設定して、 **文字の高さ $** に 24 や 36、96 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 $** に 24 と 36 を設定した例を 図 2.19 に示します。

文字の高さ $ には初期値として 0.1 があらかじめ入力されていますが、 **単位** を pix にして **文字の高さ $** を 0.1 のままにしていると、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 **単位** に norm など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。非常にありがちなミスなので注意してください。

続いて **反転** と **折り返し幅 $** です。 **反転** 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 2.19)。 **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 **単位** プロパティに従います。 図 2.19 では **折り返し幅 $** を指定しなかった場合、600pix を指定した場合、300pix

図 2.18　Text コンポーネントのアイコンとプロパティ設定ダイアログ (左：「基本」タブ・右：「高度」タブ)

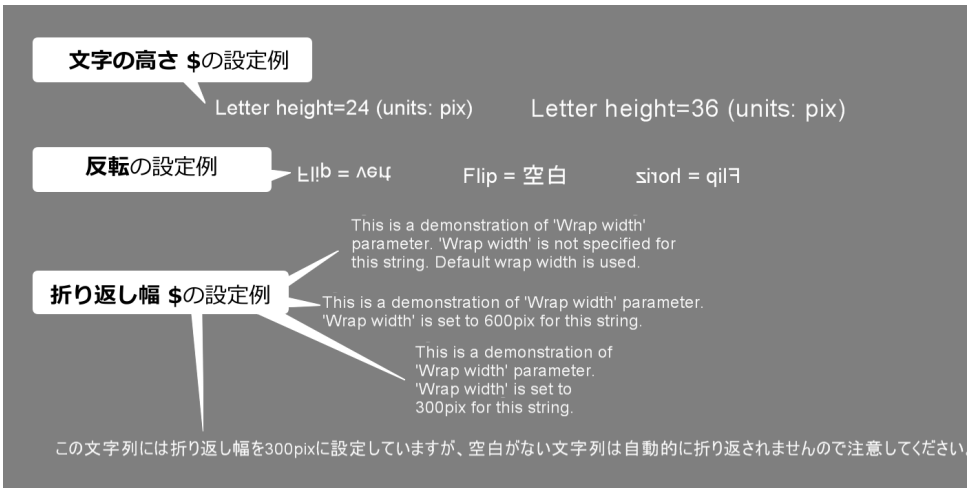


図 2.19　Text コンポーネントにおける文字の高さ、反転、折り返し幅の設定例。

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 2.19 の一番下の日本語の文字列は **折り返し幅 $** に 300pix を指定しているのですが、折り返されずに一行で表示されてしまっています。

Text コンポーネントに慣れたところで、いよいよ色の指定をしてみましょう。色を指定するにはプロパティ設定ダイアログの **色** に値を設定するのですが、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出来ます。図 2.20 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出来たのは、この web/X11 Color name による指定が利用できるからです。図 2.20 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 **色** に色名を入力して実行してみましょう。

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

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 *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

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」と表現することも出来ます。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出来ます。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出来ました。 **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出来ます。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 **色** に

```
$[-1, -1, -1]
```

と記入して実行してみましょう。カンマや角括弧、$記号もこの通りに入力してください。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と思いますので、グラフィックソフトウェアで色を作成して、変換して入力するとよいでしょう。

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 **輪郭線の色** という 2 つのプロパティがあり、それぞれ塗りつぶしと輪郭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輪郭線だけの図形や、輪郭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輪郭線だけ、**輪郭線の色** を None にすると輪郭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出来る。
- 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、紫にすることが出来る。
- **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、青色にすることが出来る。
- 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輪郭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輪郭線がない多角形を描画することができる。

2.5 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
  - **単位** を pix にする
  - **サイズ [w, h] $** を [200, 200] にする
  - **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
  - **単位** を pix にする
  - **文字の高さ $** を 24 にする
  - **文字列** を「PsychoPy Builder による心理学実験」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 2.21 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。

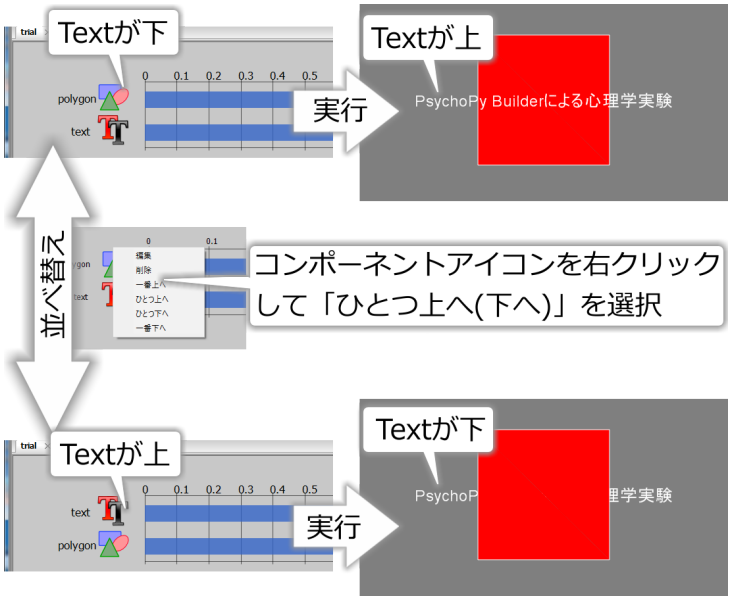


図 2.21　ルーチンペインにおける順序と刺激の重ねあわせの関係。ルーチンペインで上の方に配置されている刺激から順に表示されますので、スクリーン上での刺激の重ね順では上に配置されている刺激ほど下になります。

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 2.22 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、0.5、

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

図 2.22　**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

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出来る。

2.6 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を見比べてみると、どちらのダイアログにも上の方に 図 2.23 のようなプロパティが存在しているのがわかります。これらのプロパティの内、 **名前** 以外のプロパティが刺激の提示開始および終了に関わるプロパティです。 **名前** はコンポーネントに名前をつけるためのプロパティで、次章で詳しく解説する予定ですが、ここでも少しだけその機能に触れてみましょう。

図 2.23　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」の使用法については第 9 章で触れます。

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、二つの正方形を隙間なく並ぶように配置してください。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)
  - **名前** に red と入力
  - **単位** を pix、 **サイズ [w, h] $** を [200, 200]、 **位置 [x, y] $** を [-100, 0] にする
  - **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)
  - **名前** に green と入力
  - **単位** を pix、 **サイズ [w, h] $** を [200, 200]、 **位置 [x, y] $** を [100, 0] にする
  - **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 2.24 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。

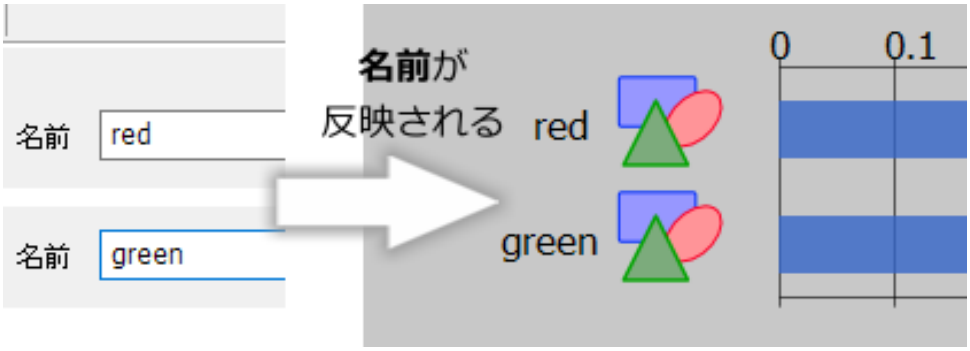


図 2.24　**名前** プロパティに文字列を入力すると、ルーチンペイン上でコンポーネントのアイコンの左側に入力した文字列が表示されます。

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 2.25 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 2.25 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。

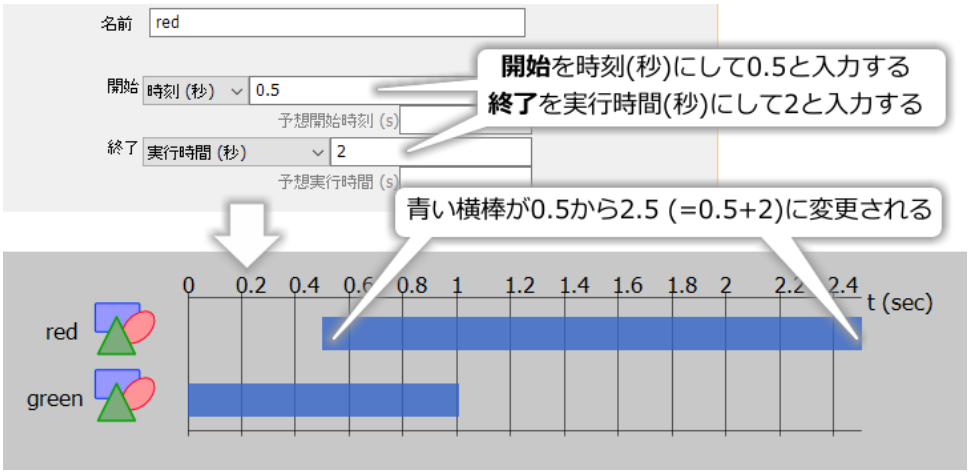


図 2.25　**開始** と **終了** の値を変更すると、ルーチンペイン上で青いバーの長さが変更されます。青いバーは実験実行時に、そのコンポーネントが有効となる (視覚刺激の場合は表示される) 時間帯を示しています。

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)」を選択することが出来ます。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出来ます。 図 2.25 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 2.25 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、 **時刻指定における *frame* について** をご覧ください。 図 2.23 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による指定と密接な関係がありますので、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 9 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 2.26)。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 2.26 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。

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出来る。
- 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出来る。
- 刺激の表示終了時刻を定めずに表示させることが出来る。
- 実行中の実験を強制的に終了させることが出来る。

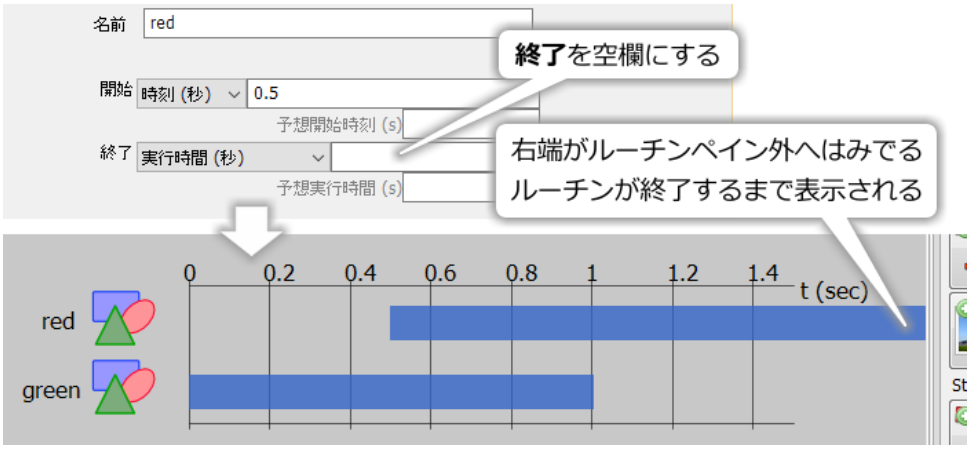


図 2.26　**終了** を空白にしておくと、ルーチンの終了までコンポーネントが有効になります。何らかの方法でルーチンを終了させない限り刺激は表示され続けます。標準設定では ESC キーを押すと強制的に実験を終了させることができます (ルーチンも強制終了されます)。

2.7 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 exp01.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 exp01_lastrun.py というファイルが出来ているはず。もし psyexp ファイルを exp01 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。

exp01_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出来ます。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出来ます。この方法を使って Builder を

使わずに直接実験を編集することも可能です。第 10 章でそういったテクニックも紹介します。

保存した psyexp ファイルは、 図 2.27 の「psyexp ファイルを開く」ボタンをクリックすると Builder で開くことが出来ます。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出来ます。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。

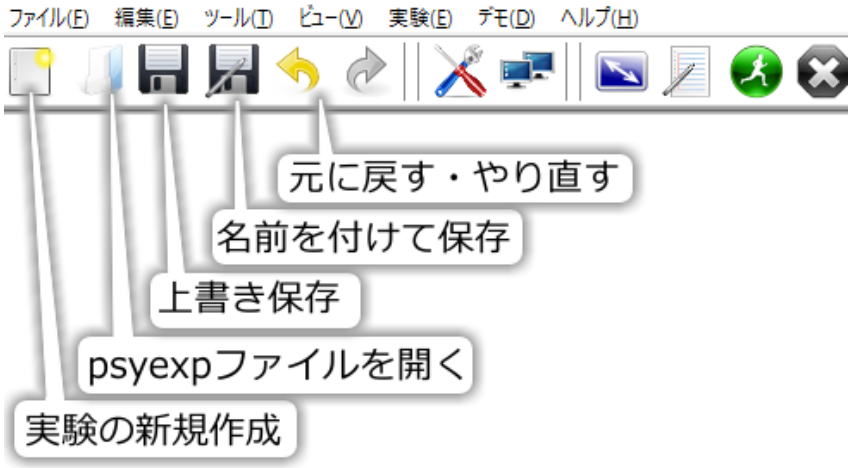


図 2.27　ファイル操作に関するボタンと元に戻す・やり直すボタン。

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出来る。
- data ディレクトリの役割を説明することが出来る。
- 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 で開くことが出来る。
- psyexp ファイルを別の名前で保存することが出来る。

2.8 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 2.28 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。

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

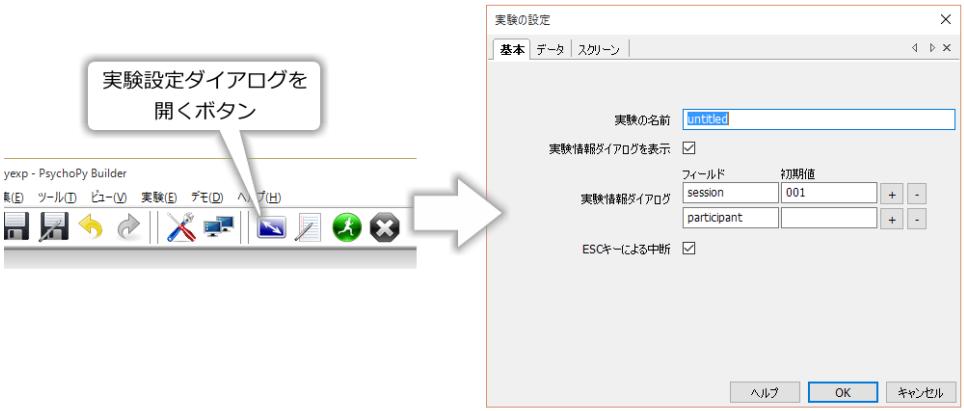


図 2.28　Builder ウィンドウ上部のツールバーのボタンから実験設定ダイアログを開くことが出来ます。

無難です。

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 (図 2.6 下) 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。

ESC キーによる中断　**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出来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 Builder のウィンドウを選択して実験停止ボタン (図 2.5) をクリックすると強制終了することが出来ます。

以上が「基本」ページの項目です。続いて「データ」ページを見ていきましょう。

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。標準で入力されている式で使われているテクニックは第 9 章で解説します。通常の用途では変更する必要はないはずです。

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。

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 (1.82.02 で確認)。

ログの保存　PsychoPy の動作状況をログファイルに記録します。

ログレベル $ 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。標準設定は exp です。通常は変更する必要がありません。

以上で「データ」ページは終了です。最後は「スクリーン」ページです。

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出来ますが、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最後に触れます。

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。

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。

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。

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出来ます。

色空間 背景色の指定に使用する色空間を指定します。

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 2.1 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。PsychoPy 設定ダイアログとは、ツールバー上の 図 2.6 に示すアイコンをクリックして PsychoPy の設定ダイアログを開き、「一般」というページの **単位** を設定することで標準の単位を設定することが出来ます。ただし筆者の個人的な意見としては、この設定に頼ってしまうと実験を作成した PC のとは別の PC で動かそうとしたときに、両 PC で標準に設定している単位が一致していないと正常に動かなくなってしまうので、実験設定ダイアログで個々の実験に対して単位を指定するべきです。

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。

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、**刺激の重ね順と透明度を理**

図 2.29　PsychoPy 設定ダイアログを開いて標準の単位を設定することが出来ます。PsychoPy 設定ダイアログはメニューの「ファイル」から「設定」を選択しても開くことが出来ます。

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 2.30 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。

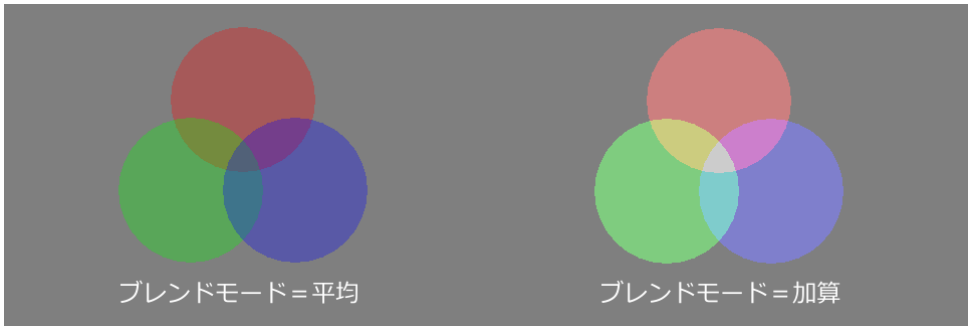


図 2.30　ブレンドモードの比較。

以上で実験設定ダイアログの概要の説明は終了です。

- 「基本」タブの **実験情報ダイアログを表示** をオフにする
- 「スクリーン」タブのに **色** を指定して背景色を変える

の二つは各自で実際に試してみてください。

チェックリスト

- 実験設定ダイアログを開くことが出来る。

- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。

2.9 モニターの設定

長くなりましたが、これ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 2.31 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」からも開くことが出来ます。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、特に違和感を感じ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 2.32)。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 2.32 では 図 2.10 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 2.32 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ねられるので、保存しておきましょう。

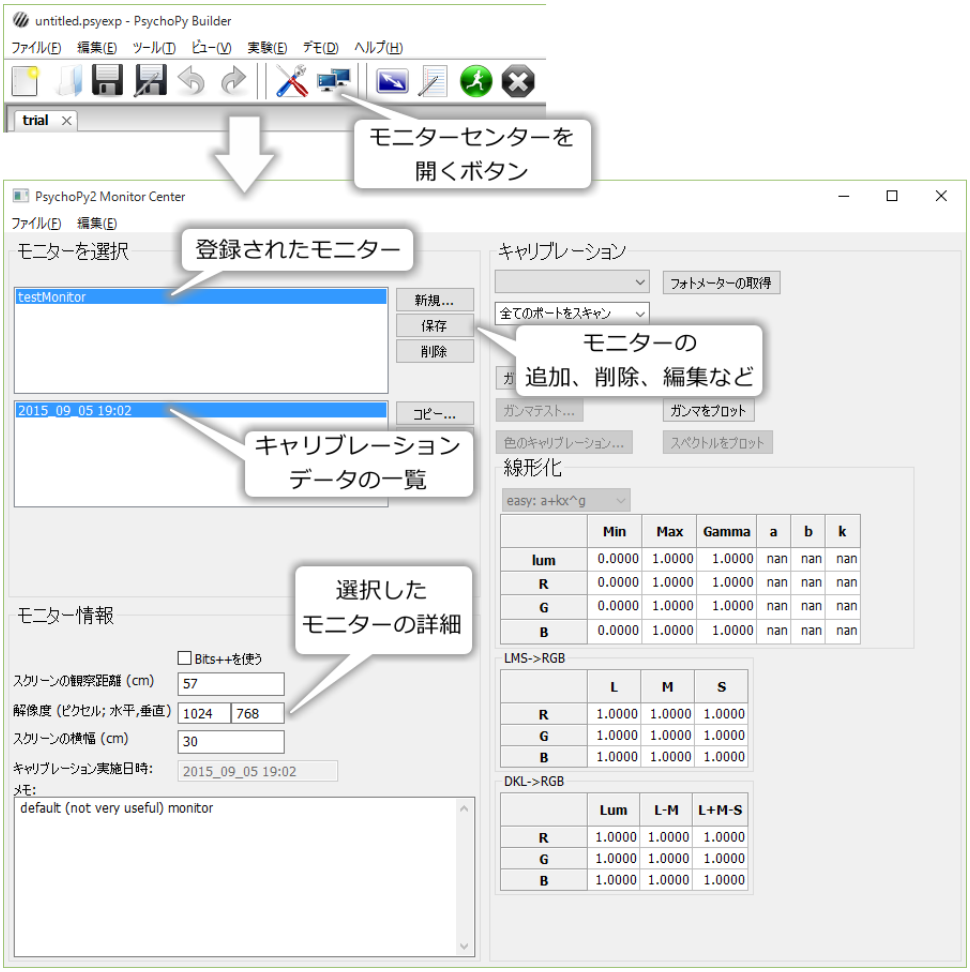


図 2.31　モニターセンターのダイアログ。モニターの登録や削除、設定の変更などが出来ます。右半分はセンサーを用いたキャリブレーション時に使用します。

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]、 **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。

確認が出来たら、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 **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 **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。

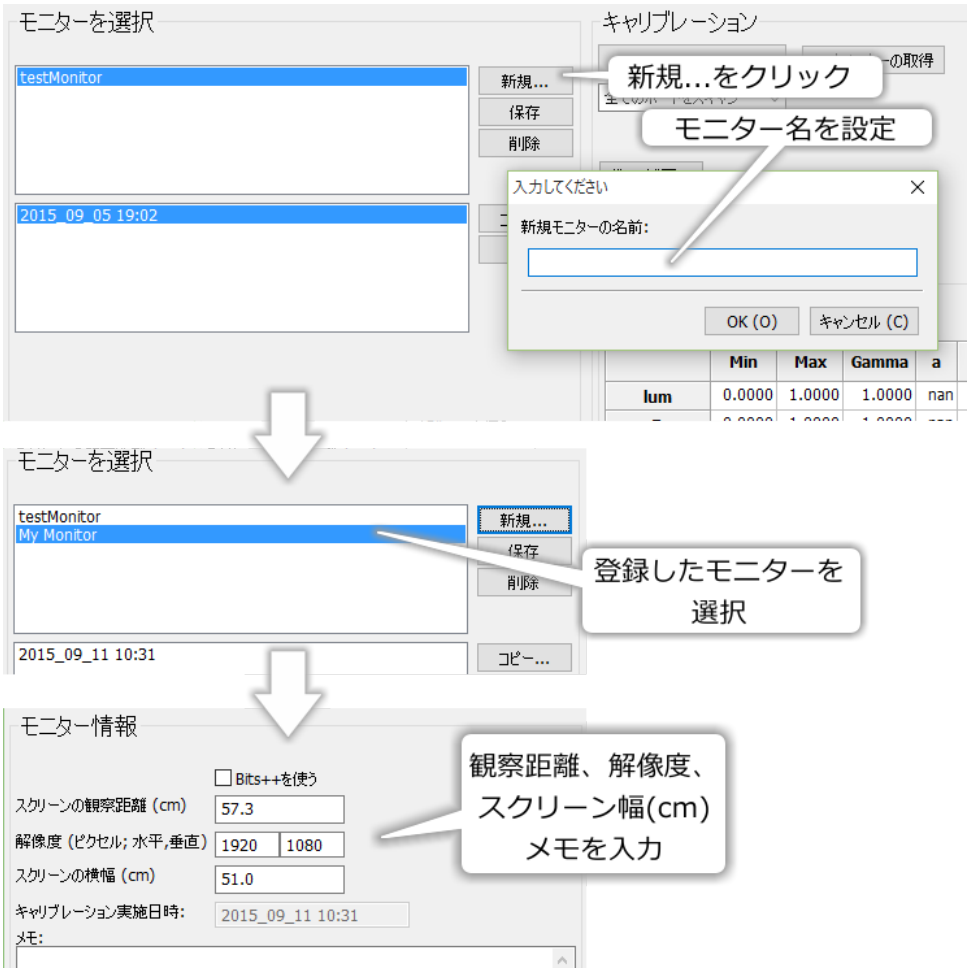


図 2.32　モニターの登録手順。

チェックリスト

- モニターセンターを開くことが出来る。
- モニターセンターに新しいモニターを登録することが出来る。
- 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出来る。

2.10 この章のトピックス

2.10.1 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出来るのですが、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(10π/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 2.33)。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。

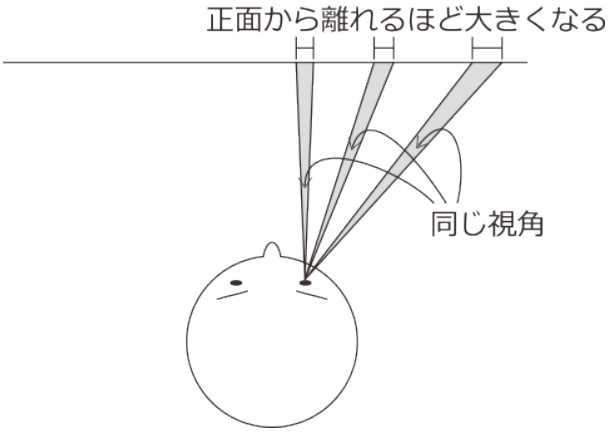


図 2.33　平面モニターを使う場合、視角を一定のまま刺激を端に移動させるとモニター画面上では刺激を大きくする必要があります。

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 2.34 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。

2.10.2 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 Windows10 では 図 2.35 のようにカンマが非常に小

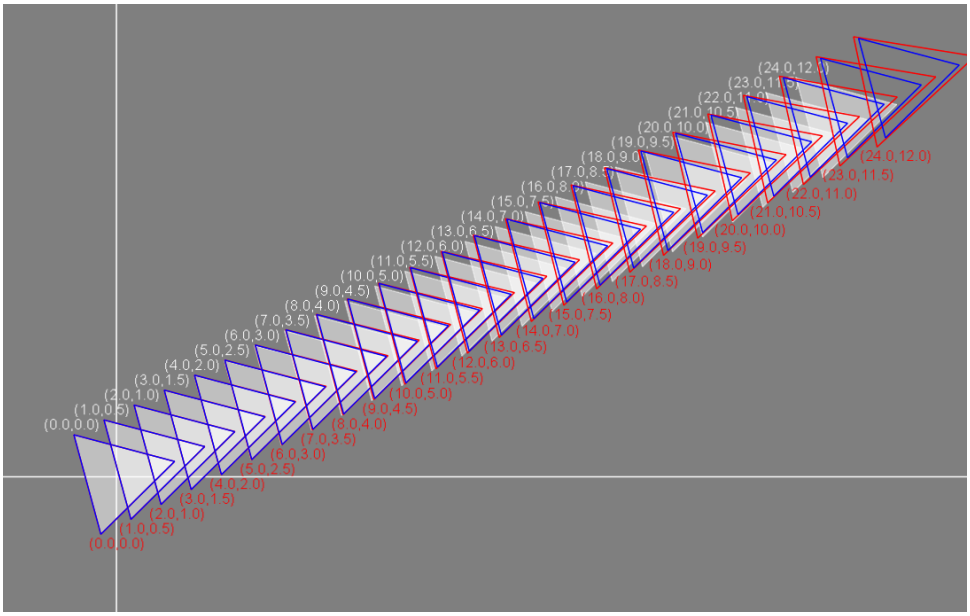


図 2.34　deg(白)、degFlat(赤)、degFlatPos(青) の違い。

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す。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。

なお、この設定ダイアログはツールバーの 図 2.29 に示すボタンをクリックしても表示することが出来ます。プロパティ名についている $ 記号の意味については第 3 章で解説します。

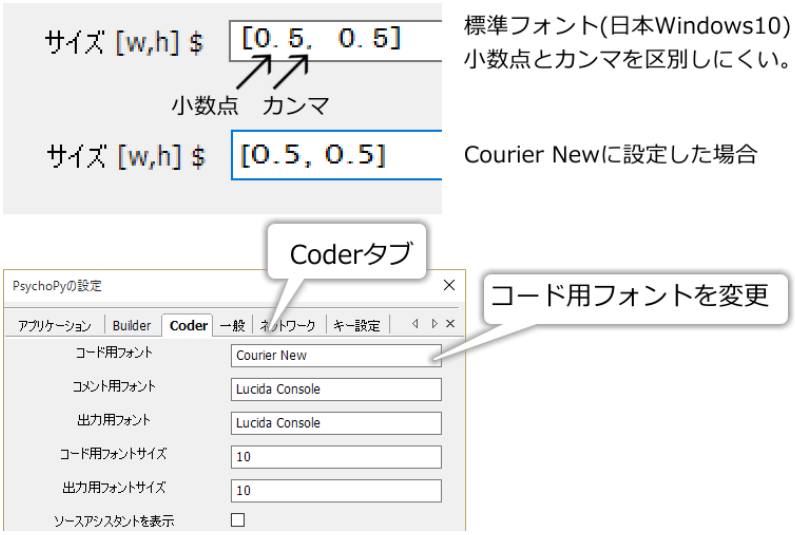


図 2.35　コード用フォントの変更。

2.10.3 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出来ます。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 では「0x」も「#」も両方使用することが出来ます。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出来ます。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。

2.10.4 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出来ます。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=0.0167 秒、すなわち 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、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 2.36)。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。

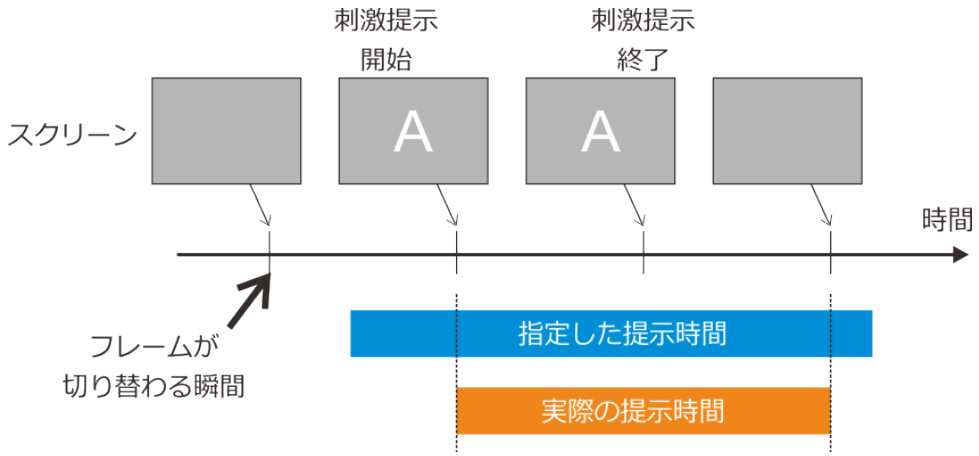


図 2.36　提示時間がフレーム表示時間の倍数になっていなければ指定した時間と実際の提示時間の間にずれが生じます。

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出来ます。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

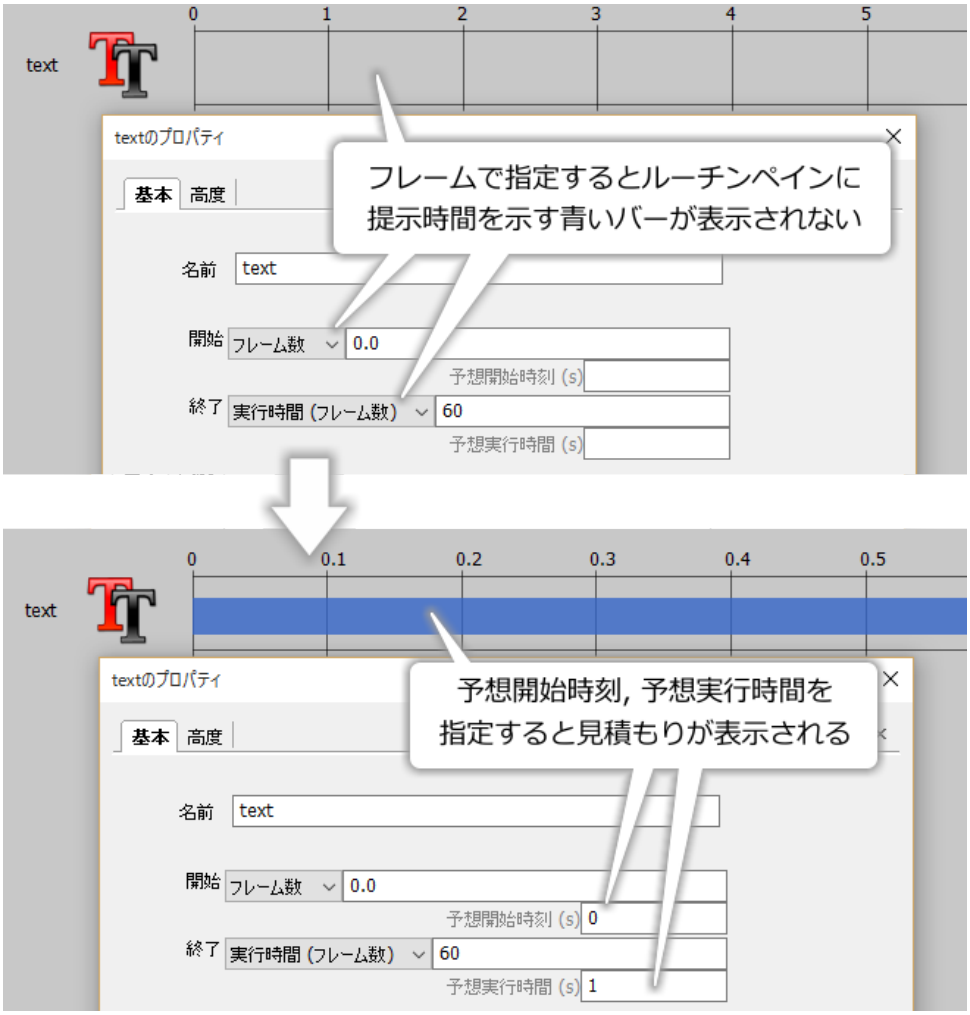


図 2.37　フレームで刺激の提示時間を指定するとルーチンペイン上で提示時間を示す青い棒が表示されません。 **予想開始時刻 (s)** 、 **予想実行時間 (s)** に見積もり値を入力することでルーチンペイン上に提示時間を表示することが出来ます。

れているかを計算することが出来ないのです。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 31)。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出来ますが、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。

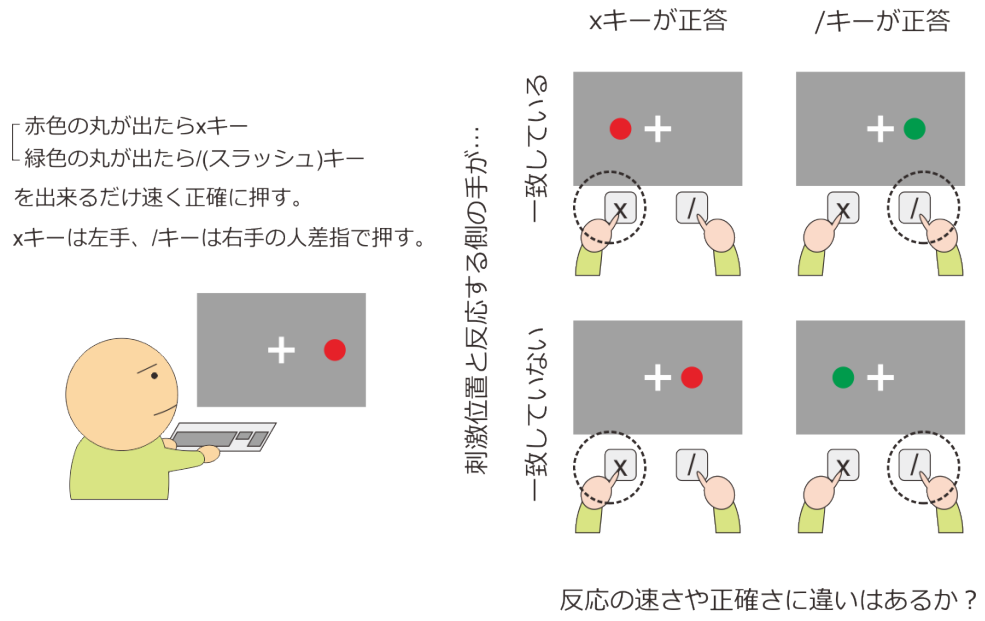


図 3.1　サイモン効果の実験概要。実験参加者は刺激の色に対応するキーを出来るだけ速く正確に押すように求められます。スクリーン上の刺激の位置と反応するキーの位置関係が反応の速さや正確さに及ぼす影響を検討します。

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) のキーを出来るだけ速く、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 50pix。
- 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 400pix。
- x4 刺激の直径は 100pix。
- 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。

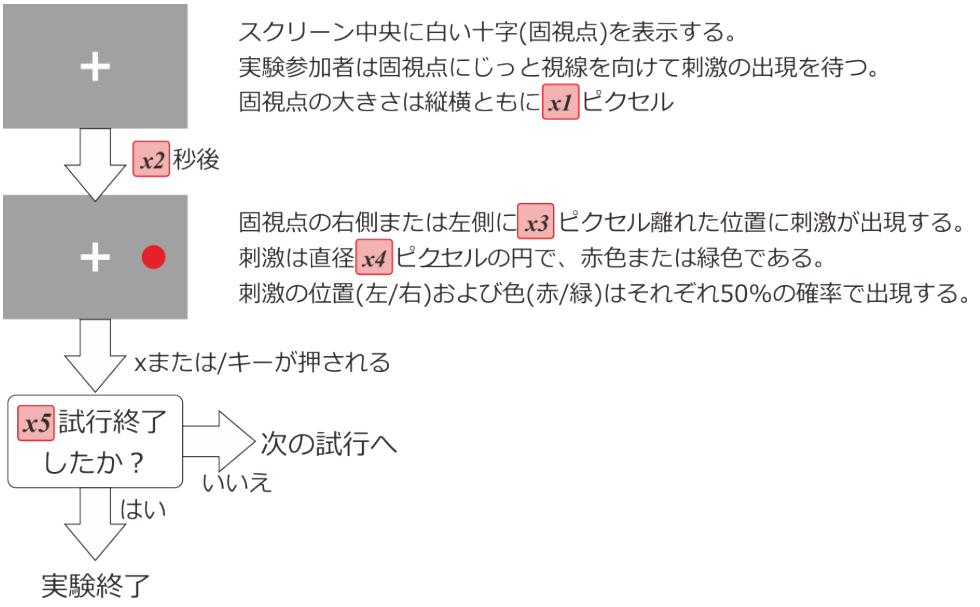


図 3.2　サイモン効果実験の手続き。実際に実験を作成するためには x1 から x5 の値を決定する必要があります。

なお、刺激はスクリーン中央からある程度離れていた方がよいので、解像度 1920 × 1080 などの解像度が高いモニターを使用している場合は、x3 の値を大きくした方がよいです (例えば 800pix)。同様に、解像度が高いモニターでは刺激の直径 (x4) が 100pix では小さすぎて見えにくいかもしれません。この辺りの数値は各自の実行環境に応じて変更してください。以下の解説では、x3 は 400pix、x4 は 100pix であるとします。

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 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 コンポーネントで頂点の数を大きくすることで表示できます。小さい円であれば頂点が 20 ほどあれば十分ですが、大きな円を提示する場合は 40 や 80 といった値にする必要があるでしょう (図 3.3)。頂点数を大きくしすぎると PC が行う表示処理の負担が大きくなるため、むやみに大きな値にするのはお勧めできません。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 [400, 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。

- **名前** に stimulus と入力する。
- **頂点数** に 40 と入力する。
- **位置 [x, y] $** に [400, 0] と入力する。
- **サイズ [w, h] $** に [100, 100] と入力する。
- **塗りつぶしの色** に red、**輪郭線の色** に None と入力する。
- 実験設定ダイアログを開いて **単位** を pix にしておく。

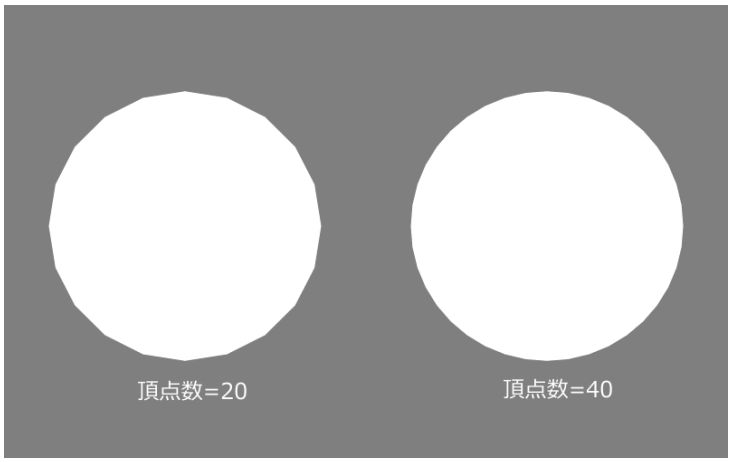


図 3.3　polygon コンポーネントの **頂点数** を増やすと円が描けます。

続いて固視点ですが、十字を表示するには Text コンポーネントを用いて「＋」を一文字だけ表示するのが最も簡単です。しかし、Text コンポーネントの場合 **文字の高さ $** に 50pix を指定しても表示される十字のサイズはスクリーン上で縦横 50pix にはなりません。文字は上下左右に余白があるのですから当然です。より正確に刺激の大きさを指定するためには、Polygon コンポーネントで長方形を二つ重ねて描くなどの工夫が必要です。ここでは幅 5pix、高さ 50pix の長方形と幅 5pix、高さ 50pix の長方形を重ねて十字を描くことにしましょう。新たに Polygon コンポーネントを 2 つルーチンに配置し、以下のように設定してください。

- **名前** にそれぞれ cross1、cross2 と入力する。

- **サイズ [w, h] $** に [5, 50] と入力する。
- cross2 の **回転角度 $** に 90 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 (cross1 および cross2) 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 3.4 上のように 3 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 3.4 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。

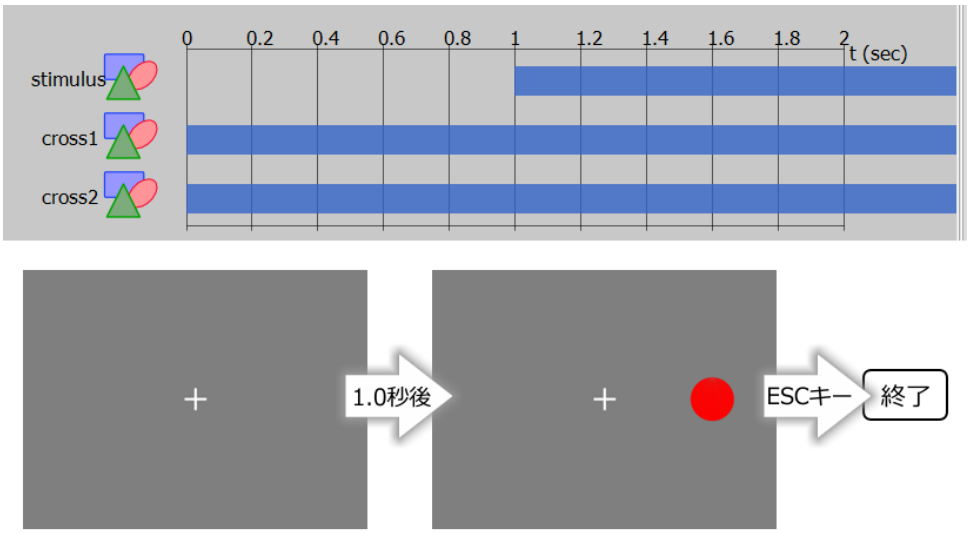


図 3.4　視覚刺激用の Polygon コンポーネントの配置と実行確認。

チェックリスト

- 試行中にキー押しが有効となる時間帯を指定できる。
- Text 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出来る。
- 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出来る。

3.3 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。Keyboard コンポーネントのアイコンは 図 3.5 左に示してあります。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をクリックしましょう。すると 図 3.5 の 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。

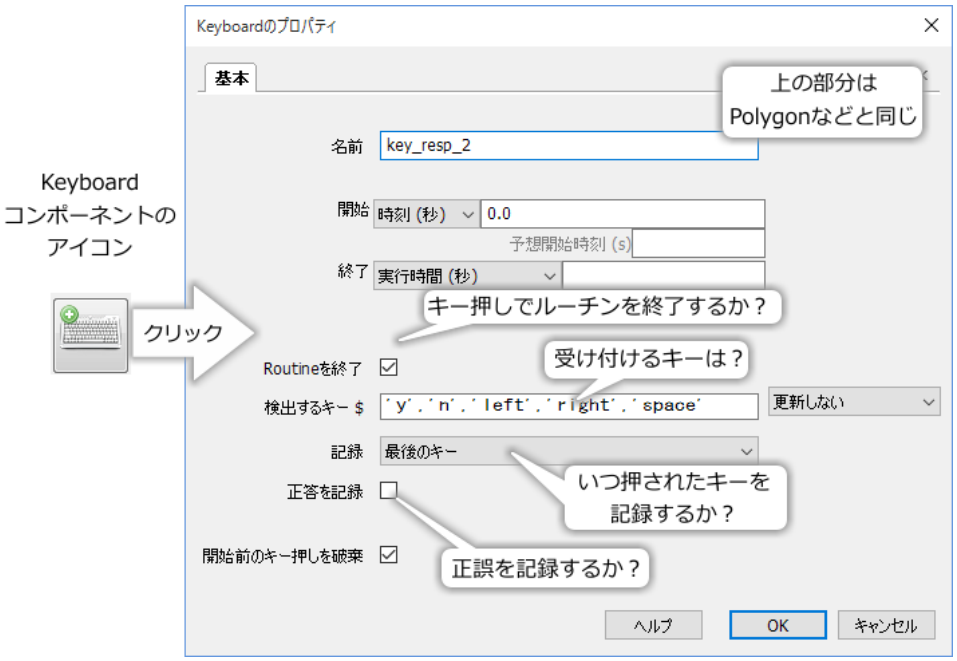


図 3.5　Keyboard コンポーネントのプロパティ設定ダイアログ。上半分は Polygon コンポーネントなどと同一です。

上半分の **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出来ます。

続いて **Routine を終了** ですが、この項目にチェックが入っていれば、キーが押されると同時に他のコンポーネントがまだ有効であっ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出来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。

さらに続いて **検出するキー $** ですが、ここには Builder がキー押しを監視するキーの名前を列挙します。 図 3.5 に示すように初期状態では'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を入力すること、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ぐことができ

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。

問題は自分が使いたいキーのキー名を何と書けばよいのかですが、一般的な日本語キーボードであれば 表 3.1 に示すキー名が使えるはずです。日本語キーボードでは 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、あくまで Shift 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。

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予定のテクニックが必要になりますので、ここでは扱わず **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。

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

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_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
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

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。

- **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。

これらの設定が済めば Keyboard コンポーネントのプロパティ設定ダイアログとルーチンペインは 図 3.6 のようになるはずです。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出来るよ

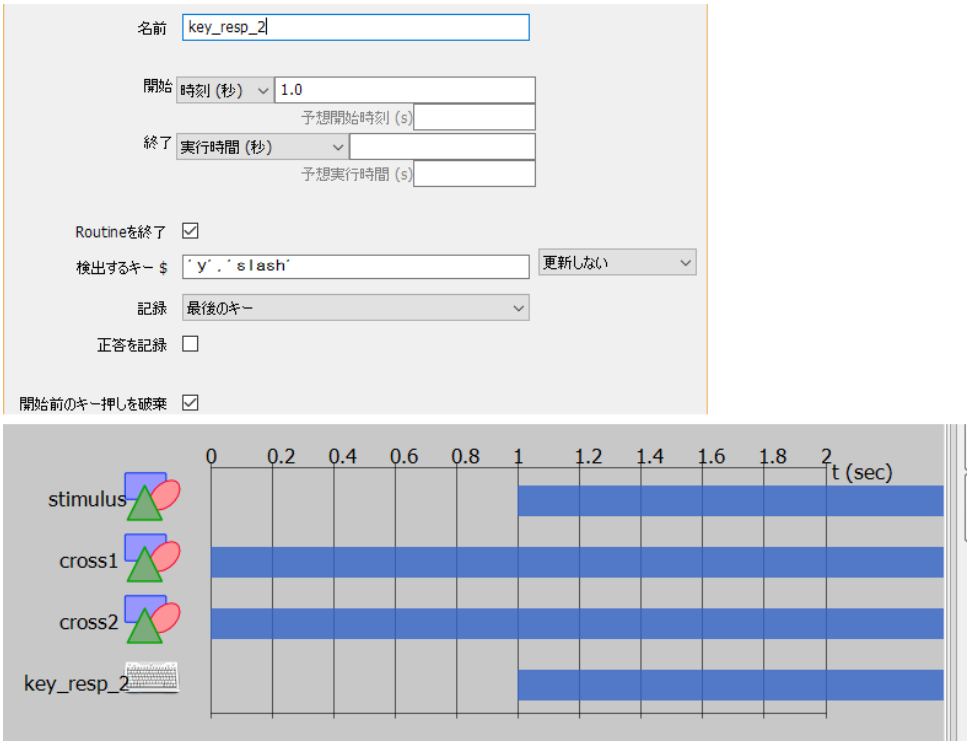


図 3.6　Keyboard コンポーネントの設定例。

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）。

キー押しが検出出来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出来る。
- 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出来る。

3.4 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出来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出来ます。実は Builder には Excel や Calc なしでも条件ファイルを作成する機能があるのですが、条件ファイルの保存形式が特殊なので本書ではお勧めしません。この機能については *Builder* **組み込みの条件ファイル作成機能** をご覧ください。

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 3.7 を参考にしながら具体的に手順を追っていきましょう。

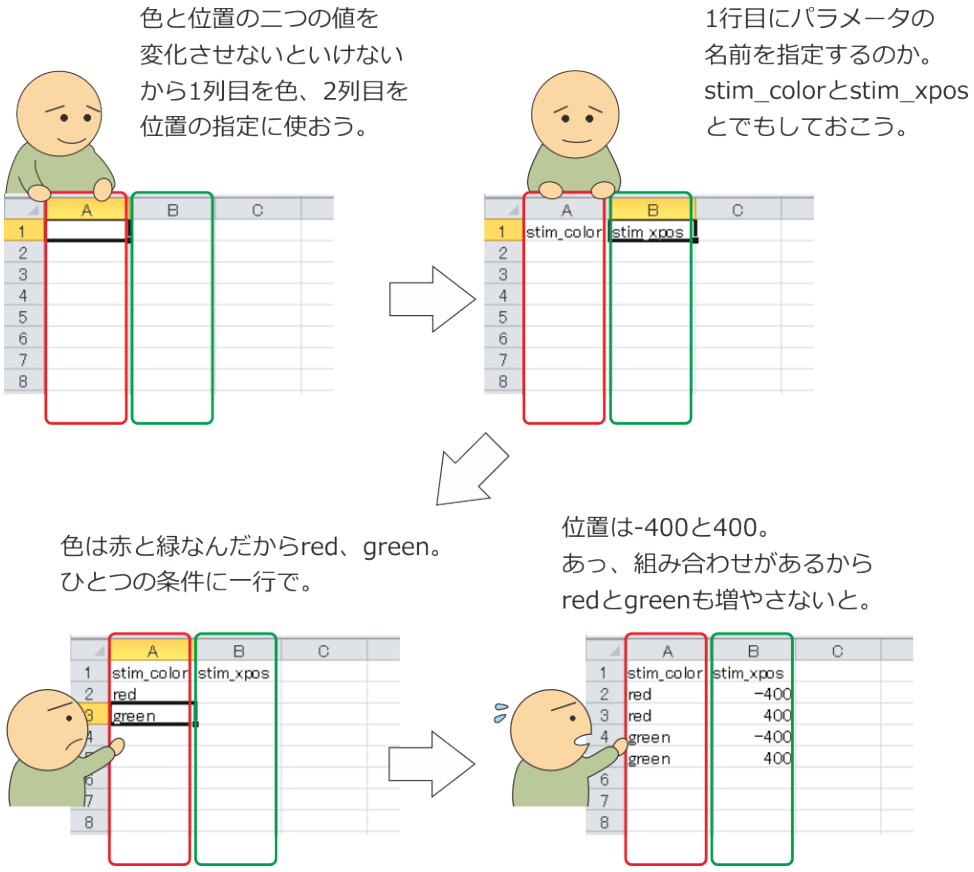


図 3.7　条件ファイルの作成。

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」と断りましたように、異なるものに同じ名前を割り当ててはいけません。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。具体的には以下の条件を満たす必要があります。

- 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。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) 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両者を合わせて「予約語」と呼ぶ。

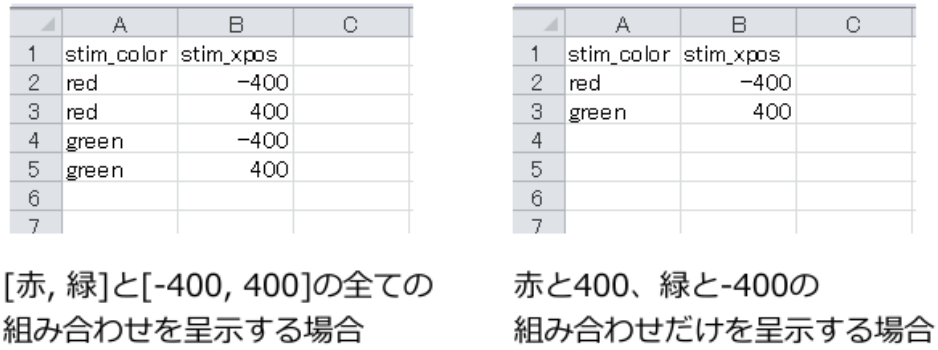
一番目と二番目の条件を満たすのはそれほど難しくありませんが、最後の「予約語を使用してはいけない」という条件は厄介です。予約語は Python や Builder が正常に動作するようにすでに使用している名前のことであり、かなりの数の名前が予約済みなので覚えるのはあまり現実的ではありません。予約語を避けるのに有効なのが、英小文字から始まる接頭語 + アンダーバーという名前を用いるという方法です。例えば自分で決める名前すべて先頭に my_ を付けたり、刺激には s_、反応には r_を付けたりするといったルールを決めておけば、ほぼ確実に予約語を避けることが出来ます。PsychoPy 1.82 の時点でこのルールに抵触するキーワード及び予約語は has_key、key_resp、raw_input の 3 つしかありません。Builder を起動中であれば *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出来ます。また、PsychoPy 1.82.02 で定義されている予約語の一覧を付録に付けておきましたので、そちらも参考にしてください。

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。ややこしいことに$[1, 0, 0]

などの表記は$を付けずに [1, 0, 0] と書かないといけないのですが、その理由は本章の **パラメータを利用して刺激を変化させよう** で説明します。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。

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 400pix に決めているのですから、左は-400、右は 400 です。stim_color と同じように stim_xpos の列に-400、400 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-400、400、-400、400 と書けばよいのです (図 3.8 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 3.8 右のように red と 400、green と-400 がそれぞれ同一の行に並ぶように書きます。



	A	B	C
1	stim_color	stim_xpos	
2	red	-400	
3	red	400	
4	green	-400	
5	green	400	
6			
7			

[赤, 緑]と[-400, 400]の全ての
組み合わせを呈示する場合

	A	B	C
1	stim_color	stim_xpos	
2	red	-400	
3	green	400	
4			
5			
6			
7			

赤と400、緑と-400の
組み合わせだけを呈示する場合

図 3.8　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。

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。保存するファイル名は、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。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 3.9)。

これで条件ファイルの準備が出来ました。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

図 3.9　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。

- 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出来る。
- 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出来る。

3.5 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

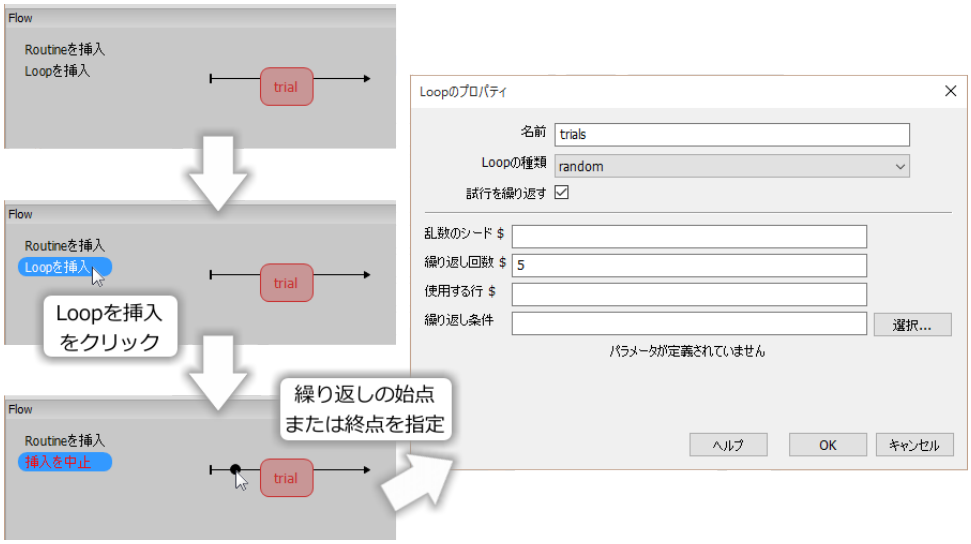


図 3.10　繰り返しの設定。

点として指定することが出来ます。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。

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、本書では指定できる種類の内 sequential、random、fullRandom の三種類を解説します。 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

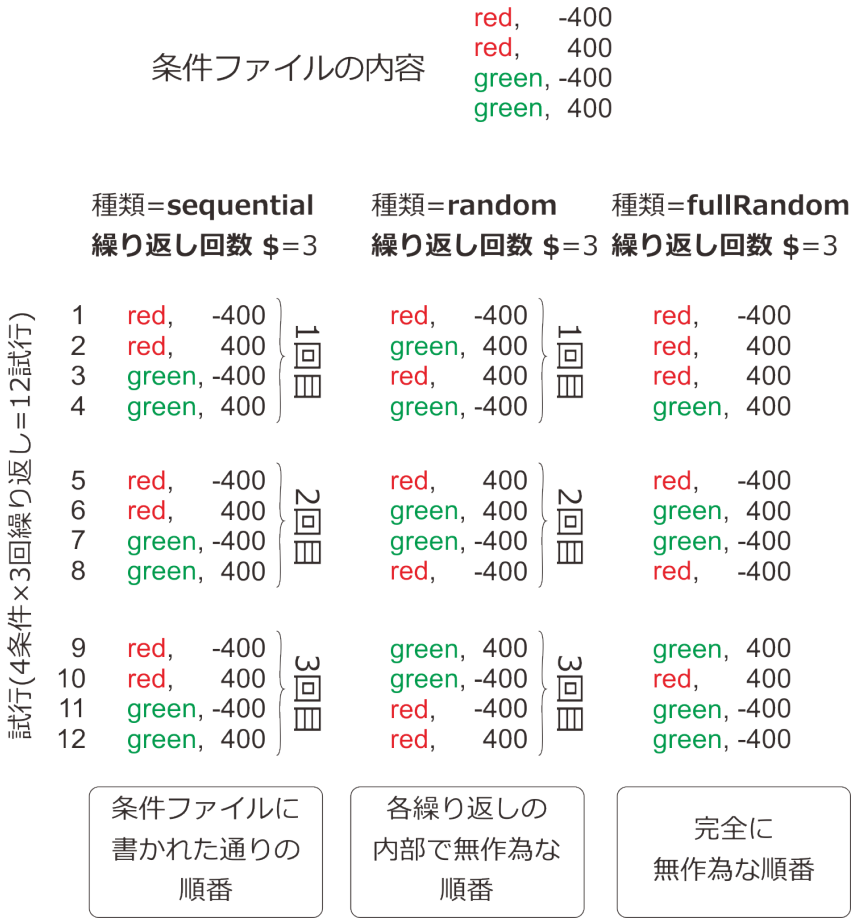


図 3.11　3 種類のループタイプの違い。

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 **無作為化と疑似乱数** をご覧ください。

繰り返し条件 には、条件ファイル名を入力します。右の「選択…」というボタンを押すと通常のアプリケーションのようなファイル選択ダイアログが出現して、目的の条件ファイルを選択すると **繰り返し条件** にファイル名が入力することが出来ます。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。

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 $** については *Loop* **のプロパティ設定ダイアログの 使用する行 *$* 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 Type 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 **繰り返し条件** の項目の右端の「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出来れば、 図 3.12 左のように **繰り返し条件** の上に条件ファイルの概要が表示されます。 図 3.12 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 3.12 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。

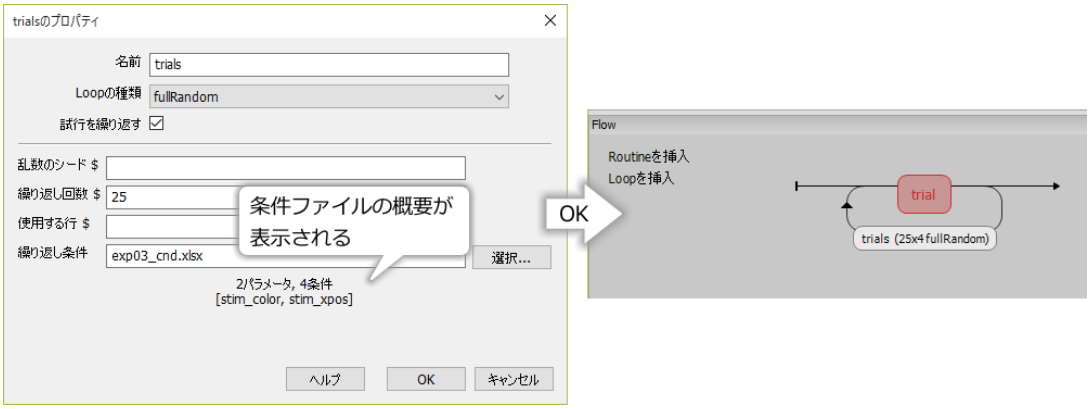


図 3.12　**繰り返し条件** に条件ファイルを指定すると、条件ファイルの概要が表示されます。概要が表示されない場合は条件ファイルの読み込みに失敗していますので条件ファイルの内容を確認しましょう。

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 3.13)。

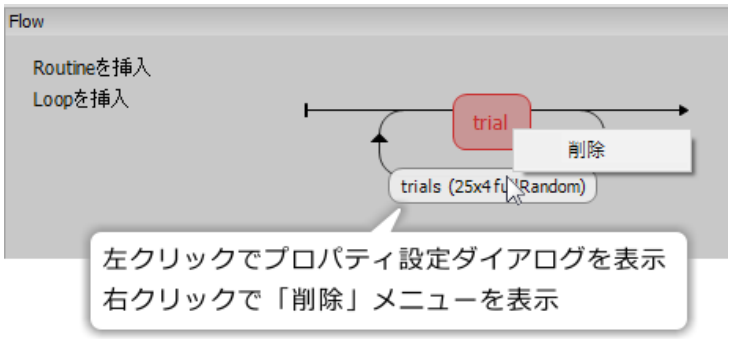


図 3.13　ループの再設定と削除

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 **繰り返し条件** 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。

3.6 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 [400, 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 3.14)。

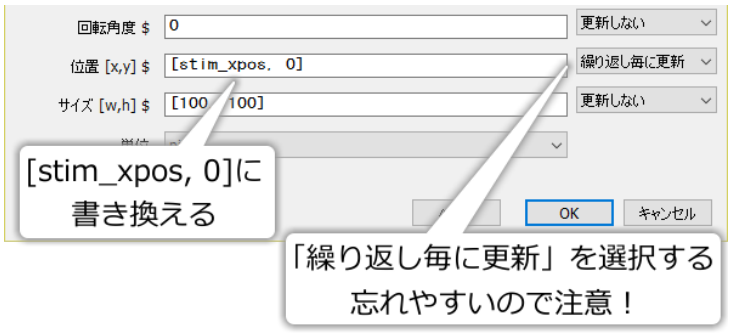


図 3.14　条件ファイルからパラメータを読み込んで自動的にプロパティを更新します。「繰り返し毎に更新」を選択し忘れるミスは非常によくあるので忘れないようにしてください。

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。今回の実験の場合は、先ほど設定した繰り返しによってこのルーチンが実行されるたびに、条件ファイルで定義された条件が無作為化された順序で代入されます。初期値の「更新しない」は文字通り「実験を通じて変化しない」という意味ですので、「更新しない」のままにしておくと代入が行われず「stim_xpos なんて値は知らないよ」というエラーが生じて実験が停止し

ます (図 3.15)。このエラーメッセージはあまり原因を適切に反映していなくてわかりづらいので気を付けてください。「更新しない」のままにするという失敗は、Builder 初心者はもちろん、ある程度慣れてきたユーザーでも時々犯しがちですので注意しましょう。

図 3.14 の変更が終わったら、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、ぜひ一度「更新しない」に戻してエラーが発生することも体験しておいてください。

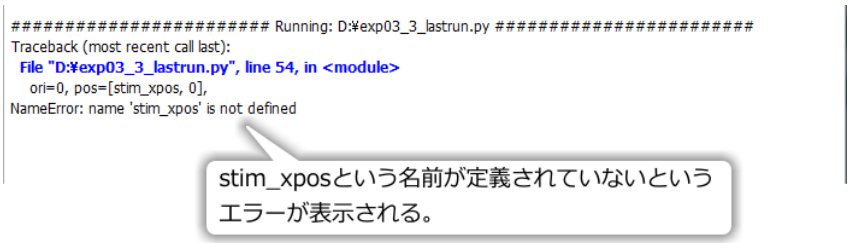


図 3.15　条件ファイルからパラメータを読み込むつもりなのに「更新しない」のまま実行してしまった時に表示されるエラー。

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 T **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 3.16)。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。

ここで、読者の皆さんの中には「ちょっと待てよ、じゃあ 図 3.14 の時はなんで$[stim_xpos, 0] じゃなくて [stim_xpos, 0] でパラメータの値を読み込めたの？」と疑問を持った方もいるかも知れません。確かに 図 3.16 の規則に従うのであれば、 図 3.14 は$[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

図 3.16　$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。

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 3.17)。

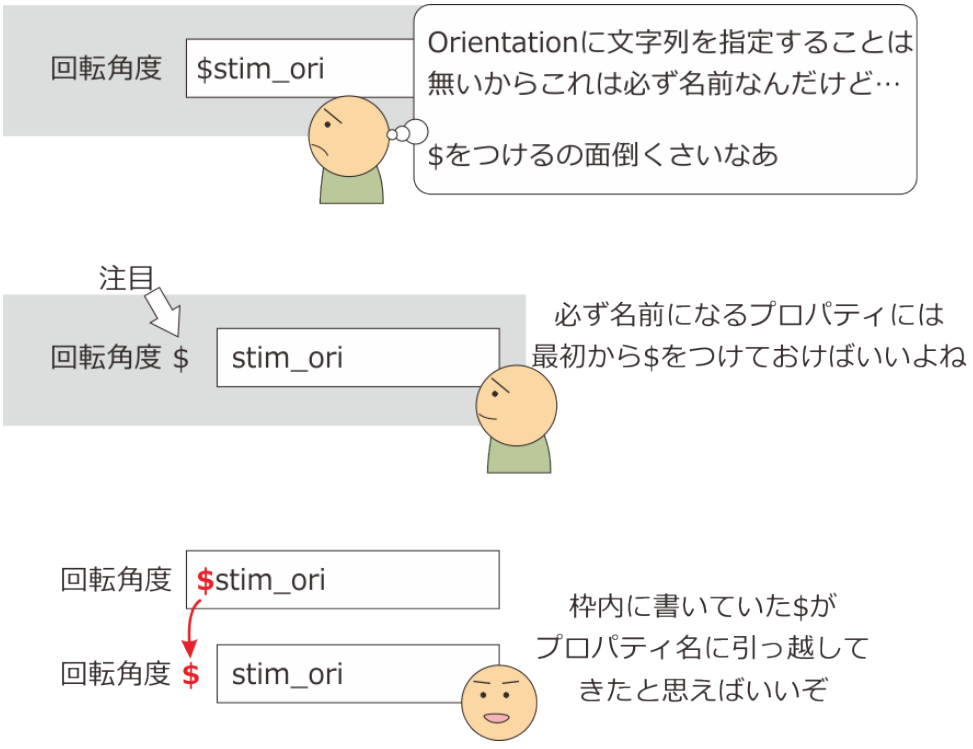


図 3.17　プロパティ名に$が付いているものは、文字列が指定されることがあり得ないプロパティなので、$を入力しなくても名前として解釈されます。

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出来るようになりました。刺激の色は **塗りつぶしの色** と **輪郭線の色** 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 **輪郭線の色** 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 3.18 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。設定が出来たら、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。

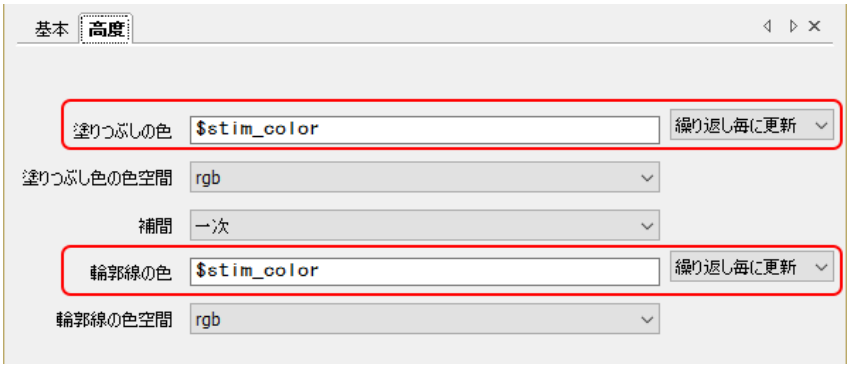


図 3.18　条件ファイルを用いて刺激の色を繰り返し毎に更新する時の設定例。「高度」タブですので注意してください。

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ここまで説明を後回しにしていた問題を取り上げましょう。まず、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$[1.0, 0.0, 0.75] のように$を必ず付けていました。この節の内容を理解した方は、なぜ$が必要であったか想像がついているのではないかと思います。 **塗りつぶしの色** 、 **輪郭線の色** 、 **色** にはいずれも$が付いていないので、$なしで [1.0, 0.0, 0.75] と書かれていると Builder には「[1.0, 0.0, 0.75] という名前の色」に見えてしまうのです。当然こんなヘンテコな名前は web/X11 Color name にはありませんので、Builder は「こんな名前の色は知らない」というエラーを出して止まってしまいます。一方、本章で条件ファイルの作成方法を解説した際に「条件ファイル内において色空間座標値で色を指定する場合は$なしで書かねばならない」と述べましたが、こちらはちょっと事情が複雑です。ごく簡単に説明しますと、Builder は条件ファイルから読み込んだ値に関しては「名前ではありえない」と考えます。ですから、条件ファイル内に書かれた$記号は「名前か、文字列か？」を判断するための記号としてではなく、単なる文字として解釈されます。ですから、条件ファイル内に$付きで$[1.0, 0.0, 0.75] と書かれていると、余計な$が付いていることになって色座標値として解釈されなくなってしまうのです。まあ、この辺りはよくわからなければとりあえず置いておいて、「条件ファイル関連でうまく値が読み込まれない時は$記号に問題があるかも知れない」と疑うことを覚えておいていただければ十分かもしれません。では$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？という点については *$*を含む文字列を提示する に書いておきましたので、興味がある方はご覧ください。

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。

3.7 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 19) で設定を変更してから実験を実行してください。ちょっと大変ですが、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、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)」、「pydat 形式のデータを保存」にチェックを付ける。
- 実験実行ボタンをクリックして実験を開始し、実験情報ダイアログの participant に foo と入力して実行する。

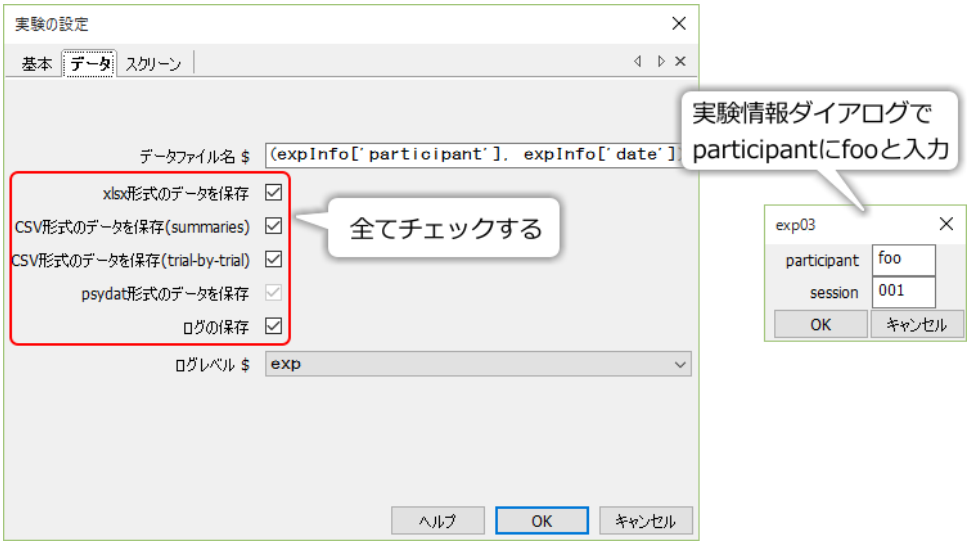


図 3.19　実験設定ダイアログを開いてすべての記録ファイルを保存するようにチェックして実験を最後まで実行してください。実験開始時の実験情報ダイアログの participant に foo と入力してください。

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出来ているはずです。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)。

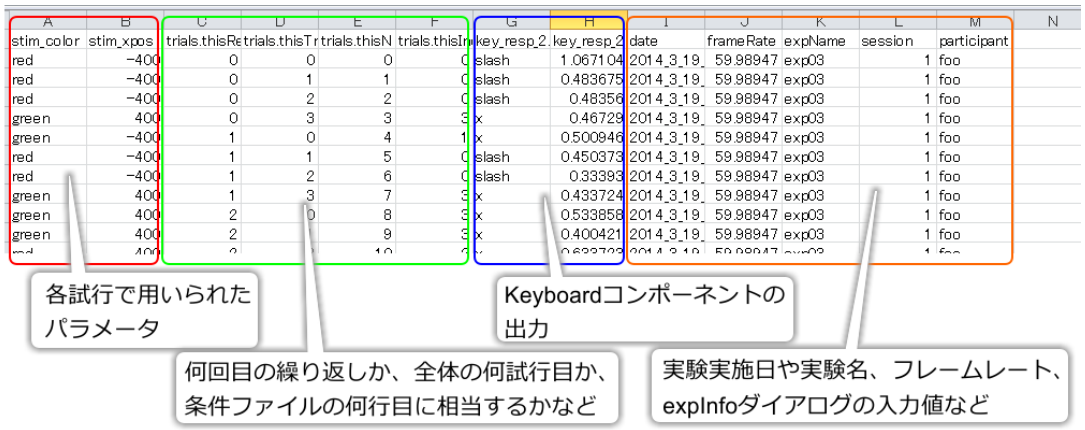
ファイル名	概要
foo_yyyy_mn_dd_hhMM.xlsx	条件毎に結果をまとめた Excel ファイルを保存する。実験設定ダイアログの「xlsx 形式のデータを保存」をチェックすると作成される。
foo_yyyy_mn_dd_hhMM.trials.csv	条件毎に結果をまとめた CSV ファイルを保存する。実験設定ダイアログの「CSV 形式のデータを保存 (summaries)」をチェックすると作成される。
foo_yyyy_mn_dd_hhMM.csv	試行毎に結果をまとめた CSV ファイルを保存する。実験設定ダイアログの「CSV 形式のデータを保存 (trial-by-trial)」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出来ますので、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 3.20)。

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



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 3.20　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。

されています。この情報を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 3.20 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。

Keyboard コンポーネントの出力。　もちろん実験に 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 *PsychoPy* **の時間計測の精度について** *(上級)* をご覧ください。

実験に関する様々な情報。　実験実施日や実験名、実験情報ダイアログで入力した値、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 **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を求めたり、x キーと slash キーが押された回数を条件毎に数えて正答率を計算したりすることが出来ます。

trial-by-trial 記録ファイルは、個々の試行の情報が 1 行にまとめられていて分析をする際にとても便利なのですが、平均反応時間や正答率を計算するという分析は心理学実験において非常によく用いられるので、ソフトウェアが自動的に計算してくれると実験者は楽が出来ます。Builder には、条件毎に平均反応時間と正答率を計算してくれる機能が備わっています。一旦 trial-by-trial 形式のファイルを閉じて、拡張子が xlsx のファイル (foo_yyyy_mn_dd_hhMM.xlsx) を開きましょう。trial-by-trial 記録ファイルに対して並び替えなどの操作をしていたら「変更を保存しますか？」と Excel から聞かれますが、筆者としては変更を保存しないことを強くお勧めします。今回は Builder の使い方を学ぶために作っている実験なので別に保存してもしなくても構わないのですが、学会発表などに用いる可能性がある実験では、まったく変更が加えられていない「オリジナル」な記録ファイルが残っていることが極めて重要です。オリジナルのファイルが残っていれば新たに分析しなおすことが可能ですが、変更を保存してしまうと二度とオリジナルの状態に復元できないかもしれません。大切な記録ファイルは必ず未変更の状態で保存しておいて、変更を加えた場合は別のファイル名にするか別のフォルダに保存すべきです。

foo_yyyy_mn_dd_hhMM.xlsx を開くと、 図 3.21 のように横方向にたくさんの値が並んだシートが表示されるはずです。このファイルは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ので、xlsx 記録ファイルと呼ぶことにしましょう。xslx 記録ファイルでは、シートの 1 行がひとつの条件に対応しています。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た後、n という見出しの列が続きます。この列には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 3.21 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。続いて key_resp_2.keys_raw という見出しの列からずらっと横方向に Keyboard コンポーネントが記録した「押されたキー名」が記録されています。並んでいる列数は各条件に含まれる試行数 (この例では 25 列) に一致します。キー名に続いて、key_resp_2.rt_mean という見出しの列が来ます。この列には各条件の平均反応時間が示されています (単位は秒)。key_resp_2.rt_mean に続く key_resp_2.rt_raw は個々の試行の反応時間です。やはり各条件の試行数だけ反応時間の値が続いた後、key_resp_2.rt_std という列に反応時間の標準偏差が示されています。最後に order という見出しと共に、横方向に並べられたそれぞれの試行が繰り返し全体の中で何試行目であったかを示す値が入っています。というわけで、条件別の平均反応時間はわざわざ計算しなくても Builder の記録ファイルに出力されていることがわかりました。

気を付けておかないといけないのは、xlsx 記録ファイルの作成を指示する実験設定ダイアログの「xlsx 形式のデータを保存」が初期状態ではチェックされていないという点です。「trial-by-trial 記録ファイルだけあれば後は自分で分析するから大丈夫」という人以外は、忘れずに「xlsx 形式のデータを保存」をチェックしておくようにしましょう。なお、実験設定ダイアログには「xlsx 形式のデータを保存」 の下に「CSV 形式のデータを保存 (summaries)」という項目がありますが、この項目にチェックしておくと、xlsx 形式のファイルとほぼ同一の内容が CSV 形式で出力されます。ファイル名は foo_yyyy_mn_dd_hhMM.trials.csv という具合に実験実施時刻の後に.trials という文字列が入ります。この記録ファイルを summaries 記録ファイルと呼ぶことにしましょう。実験後の分析に使用するアプリケーションとの相性などを考慮して、xlsx 記録ファイルか summaries 記録ファイルかどちらかの一方を作成しておけばよいでしょう。一般論として、summaries 記録ファイルはほとんどのファイルで開くことが出来ますが、条件ファイルの値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 が必要ですが、日本語を含んでい

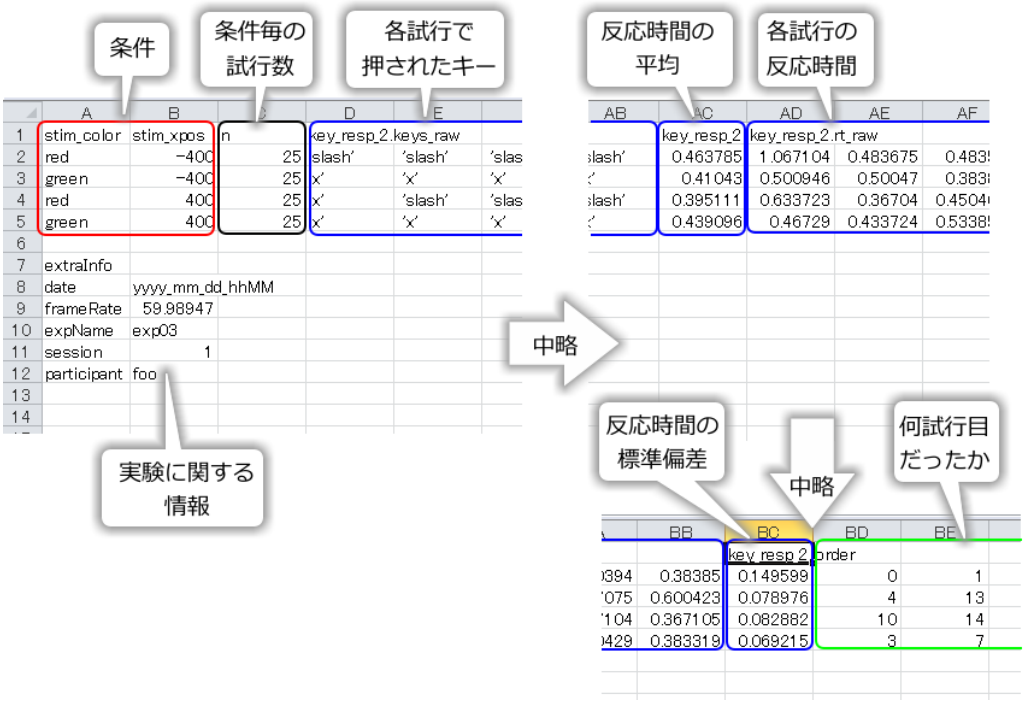


図 3.21　xlsx 記録ファイルの内容。Keybooard コンポーネントの出力が 1 条件 1 行でまとめられています。

ても正常に表示できます。

さて、平均反応時間は xlsx 記録ファイルに出力されていることがわかりましたが、正答率はどうでしょうか。押されたキーのキー名が条件毎に 1 行にまとめられていますので、x キーまたは slash キーが何回押されているか条件別に数えれば正答率を計算することはそれほど難しくありません。ですが、実は正答率の計算も Builder に自動的にさせる方法があるのです。次節ではその方法について解説します。

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出来る。
- 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出来る。
- 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出来る。
- trial-by-trial 記録ファイルと xlsx 記録ファイル、summaries 記録ファイルの違いを説明できる。
- 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出来る。
- 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出来る。
- trial-by-trial 記録ファイル、xlsx 記録ファイル、summaries 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出来る。
- trial-by-trial 記録ファイル、xlsx 記録ファイル、summaries 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出来る。

- trial-by-trial 記録ファイル、xlsx 記録ファイル、summaries 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出来る。
- xlsx 記録ファイルおよび summaries 記録ファイルから各条件の試行数を読み取ることが出来る。
- xlsx 記録ファイルおよび summaries 記録ファイルから反応時間の平均値と標準偏差を読み取ることが出来る。
- 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。

3.8 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 3.22 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** という項目が出現します。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$correct_ans ですか？ 図 3.22 を一番下まで先にご覧になった方はすでにおわかりと思いますが、答えは$付きの$correct_ans です。なぜ$が必要か、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、stim_color が red の行は x、green の行は slash が正答キーです (図 3.23)。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

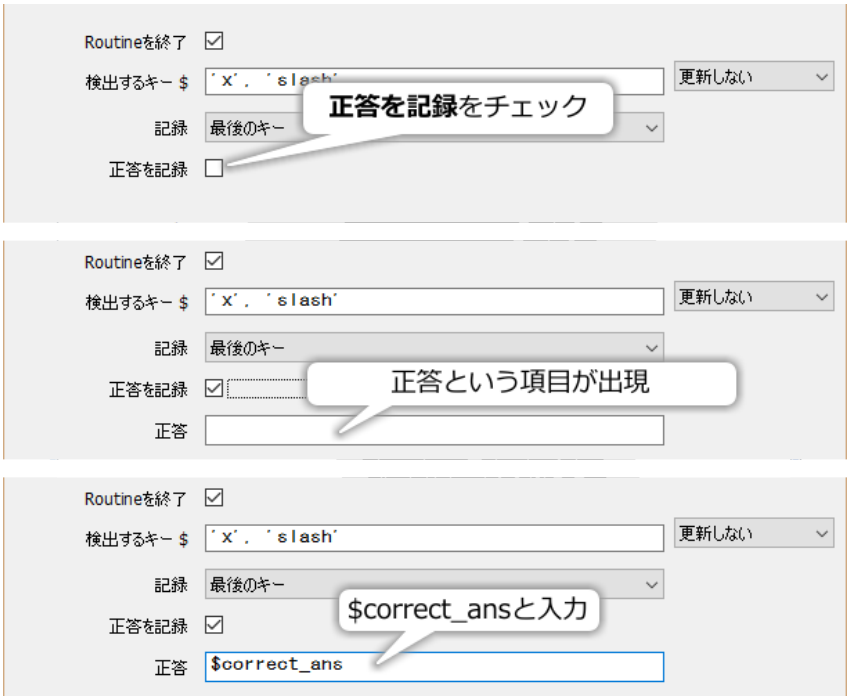


図 3.22　反応の正誤を記録するよう設定する手順。

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	D
1	stim_color	stim_xpos	correct_ans	
2	red	-400	x	
3	green	-400	slash	
4	red	400	x	
5	green	400	slash	

図 3.23　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 3.24)。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。xlsx 記録ファイルと summaries 記録ファイルでは、key_resp_2.corr_mean、key_resp_2.corr_raw、key_resp_2.corr_std という列が追加されます。それぞれ各条件における key_resp_2.corr の値をまとめたものですが、1 が正答、0 が誤答なのですからその平均値は正答率 (0=全問不正解、1=全問正解) そのものです。ですから、key_resp_2.corr_mean を見れば各条件の正答率がわかります。念のため補足しておきますが、key_resp_2 の部分は当然 Keyboard コンポーネントの名前に応じて変化します。

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

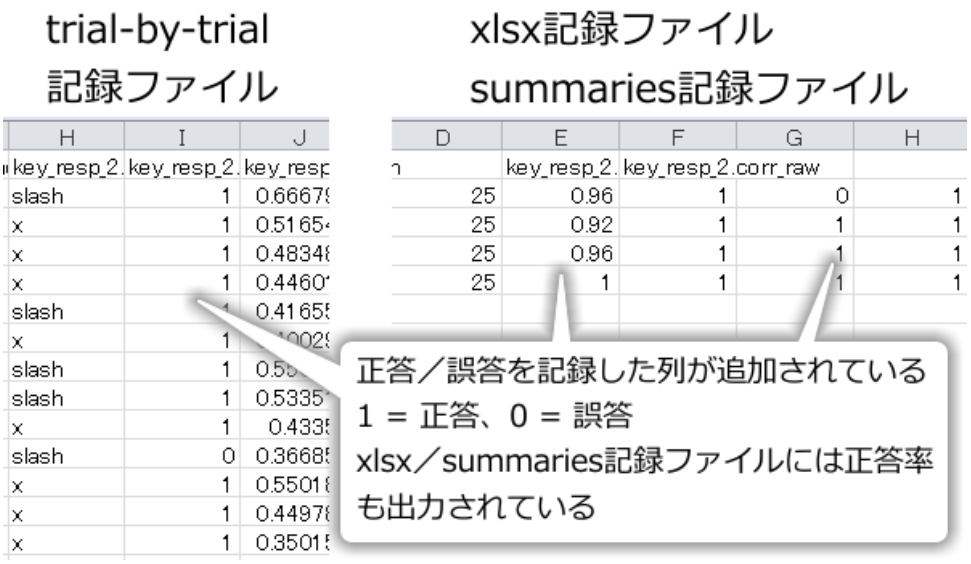


図 3.24　**正答** を設定すると記録ファイルに正答／誤答が記録されます。

るとなるといくつか不都合な点があるからです。例えば、実験実行ボタンを押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出来ないでしょう。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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- xlsx 記録ファイルおよび summaries 記録ファイルから各条件の正答率を読み取ることが出来る。

3.9 教示などを追加しよう

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出来ません。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 3.25 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入力しましょう。名前はパラメータ名に使用出来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。

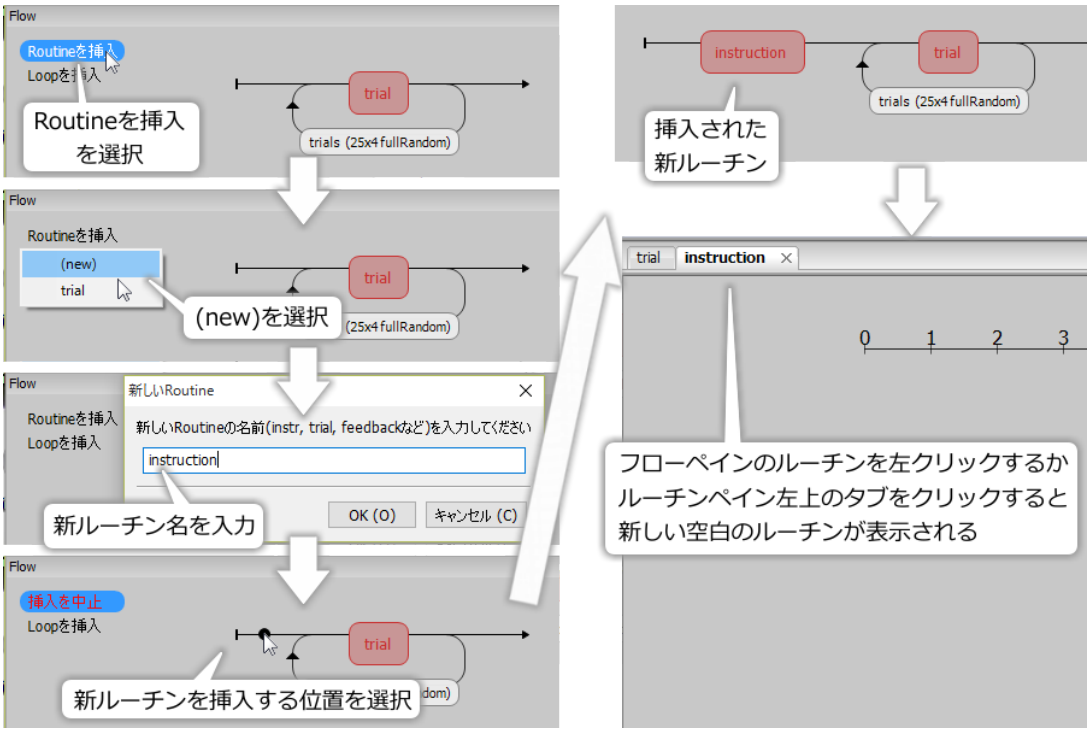


図 3.25　フローペインから新しいルーチンの追加を追加します。

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 3.25 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 3.26)。

- 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「削除」という項目のみのポップアップメニューが表示され、それを選択するとルーチンをフローから削除出来ること。
- 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。

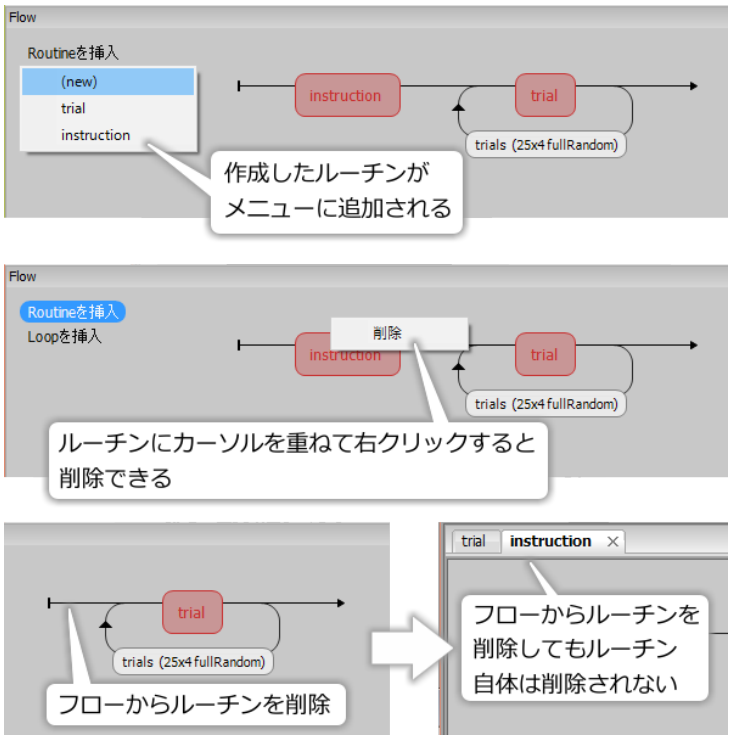


図 3.26　フローペインにおけるさまざまな操作。フローからルーチンを削除してもルーチンそのものは削除されません。

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 1.82.02 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、以下のキーを出来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

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出来ると期待します。

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 3.27)。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出来ます。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 3.27　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。対して、trial や instruction ルーチンは赤色で表示されています (図 3.28)。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。赤いルーチンは、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。

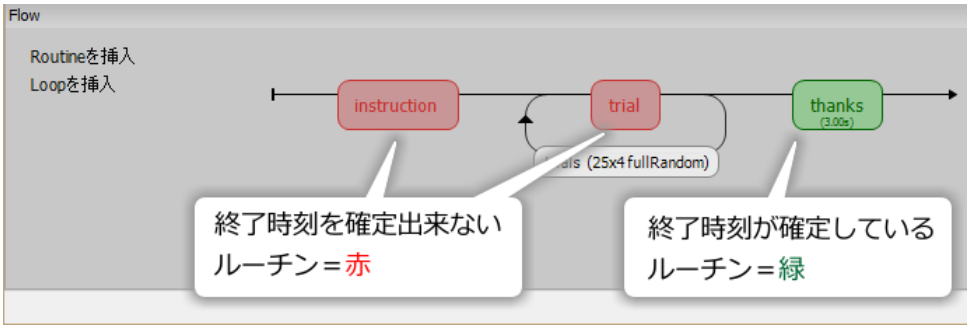


図 3.28　赤いルーチンは終了時刻が確定していないことを、緑色のルーチンは終了時刻が確定していることを表しています。緑色のルーチンにはルーチンの実行時間の見積もりが表示されています。

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 3.29 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出来ます。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 3.29 下)。

ルーチンの新規作成、コピー＆ペーストときたら、削除と名前変更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 3.30)。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。

続いてルーチンの名前変更ですが、バージョン 1.84.0 より前の Builder を使用している場合は残念ながらルーチン名の変更が出来ません。あまりお勧めできる方法ではありませんが、psyexp ファイルを直接編集することによって名前の変更を行うことは可能です。詳しくは第 10 章をご覧ください。PsychoPy 1.84.0 以降を使用している場合は、フローペインで名前を変更したいルーチンを右クリックして「名前の変更」を選択するか、ルーチンペインで名前を変更したいルーチンを表示した状態で「実験」メニューから「Routine の名前を変更」を選択してください (図 3.31)。

以上で第 3 章の解説を終えたいとと思います。簡単な実験であればこの章の内容だけで十分に実現できるはずです。最後にこの章の締めくくりとして、この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。第 3 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出来る。

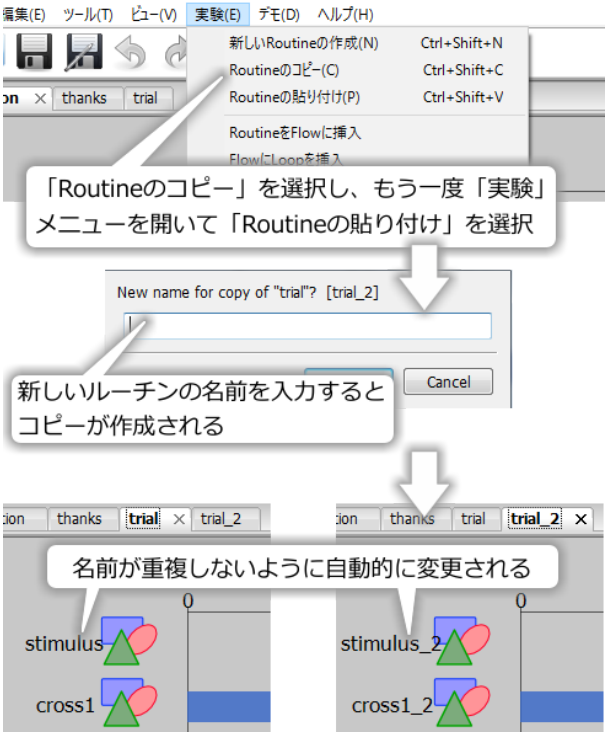


図 3.29　ルーチンのコピー＆ペーストおよび削除。コンポーネントの名前が重複しないように、ペーストされたルーチン内のコンポーネント名には元のコンポーネント名にアンダーバー + 数字が自動的につけられます。

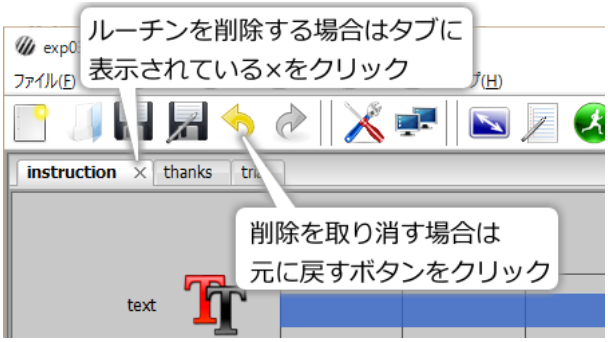


図 3.30　ルーチンを削除するにはタブに表示されている×をクリックします。誤って削除してしまったなどの理由で削除を取り消したい場合は元に戻るボタンを使います。

- 既存のルーチンをフローに追加することが出来る。
- 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出来る。
- 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出来る。
- 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出来る。

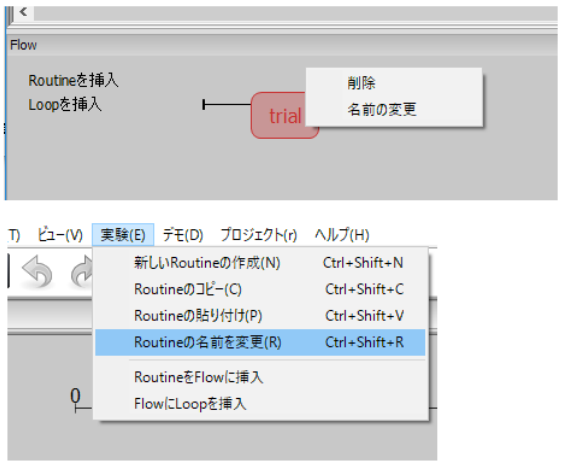


図 3.31　フローペインのルーチンのアイコン上で右クリックするか、「実験」メニューから「Routine の名前を変更」を選択するとルーチン名を変更することが出来ます。

3.10 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。

3.11 この章のトピックス

3.11.1 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。

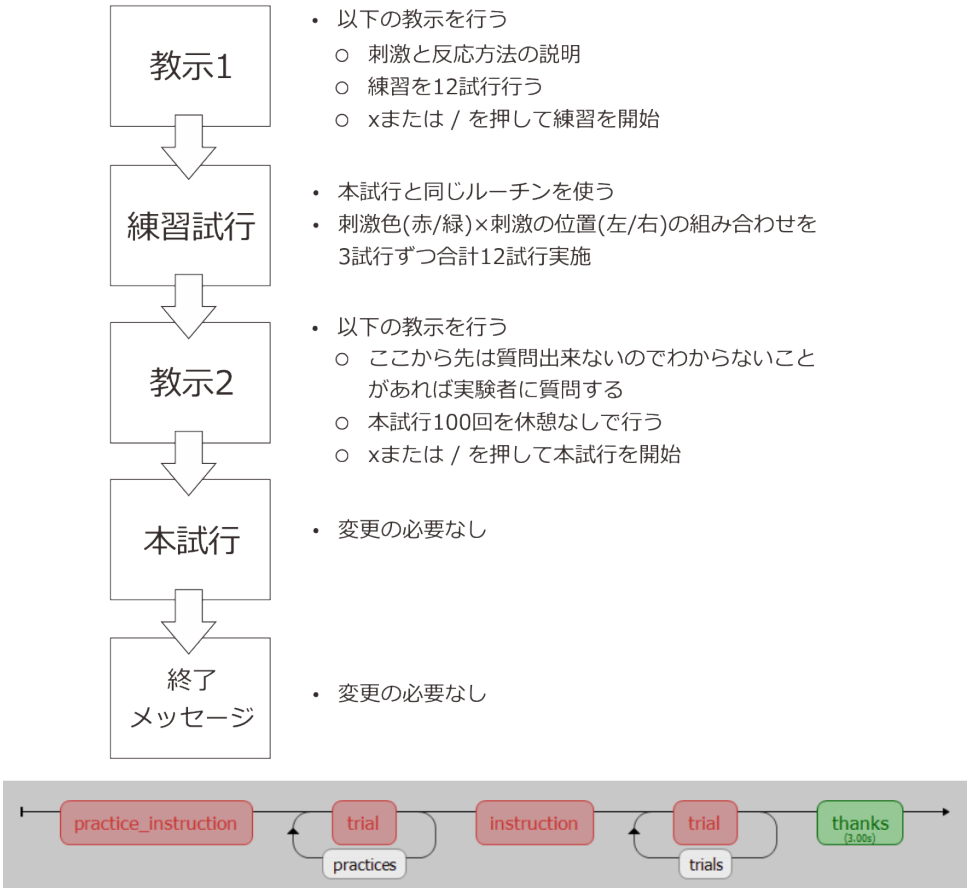


図 3.32　練習問題。本試行を開始する前に 12 試行の練習を行うように改造してみましょう。一番下は作成例のフローを示しています。

  - **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。
  - もし一度実行して字が小さすぎたりするようならば **単位** や **文字の高さ $** を調節する。

- 実験設定ダイアログ
  - Full-screen Window のチェックを外す。このチェックを外しておかないと ESC による強制終了が効かなかった時に強制終了させるのが面倒なので忘れずにチェックを外しておく。

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。通常は

ESC キーを押すと強制終了できますが、この実験では画面上に escape と ESC キーのキー名が表示されて実験が終了しない場合があります。その場合はあわてずに Builder のウィンドウを選んで実験停止ボタンを押してください。実験が終了します。

3.11.2 Builder 組み込みの条件ファイル作成機能

ループのプロパティ設定ダイアログにおいて、 **繰り返し条件** の入力欄の上にマウスカーソルを置いて右クリックすると、条件ファイルの編集ダイアログを表示することが出来ます (図 3.33)。「パラメータ」の行の param_A などと書かれている欄をパラメータ名に書き換え、「条件 01」以降の行に値を入力して条件ファイルを作成できます。「データ型」という行のプルダウンメニューはデータ型の指定に用います。値が文字列であれば utf-8、数値であれば int(整数) や float(小数)、 **位置 [x, y] $** や Size [w, h] で用いる [ ] で囲まれた数値の組み合わせの場合は list を指定するとよいでしょう。条件数が足りない場合は「条件追加」、パラメータ数が足りない場合は「パラメータ追加」をクリックします。「プレビュー」ボタンを押して内容確認、「名前を付けて保存」をクリックしてファイルに保存します。

とても便利な機能のように見えますが、値のコピー＆ペーストといった編集機能が Excel や LibreOffice Calc と比べて非常に貧弱なので、大規模な条件ファイルを作るのは困難です。また、条件ファイルの保存形式が Pickle という特殊な形式ですので、後で Excel などを使って開くことが出来ません。予算が足りない、大学の端末に勝手にインストールできないなどの理由で Excel を導入することが難しいでも、LibreOffice Calc であれば無料で利用できますし、インストールせずに USB メモリに入れて利用できるポータブル版が用意されています。Calc を利用する方が良いと思います。

なお、xlsx 形式の条件ファイルを **繰り返し条件** に指定している時に入力欄を右クリックすると、読み込んだ条件ファイルの内容をプレビューすることが出来ます。

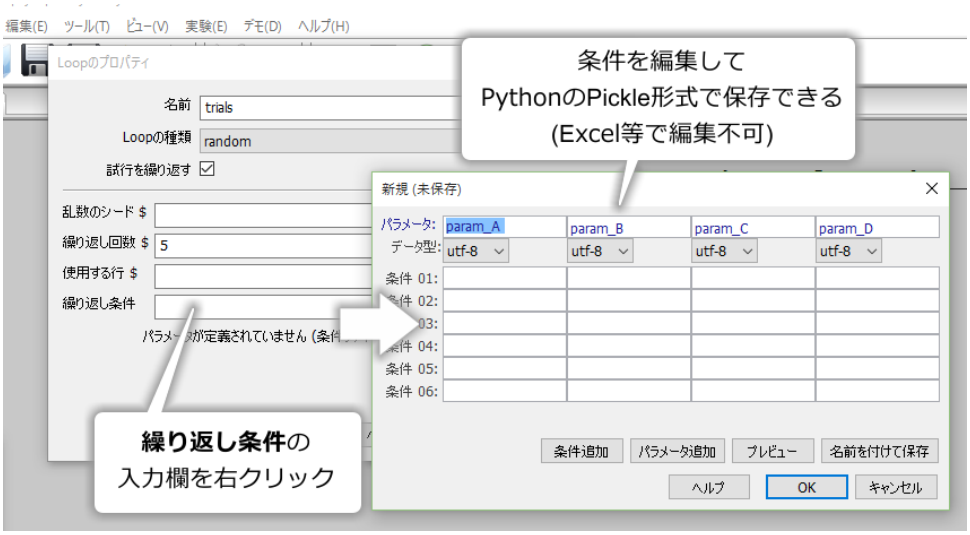


図 3.33 **繰り返し条件** の入力欄をクリックすると、Builder 組み込みの条件ファイル作成ダイアログを開くことが出来ます。

3.11.3 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。

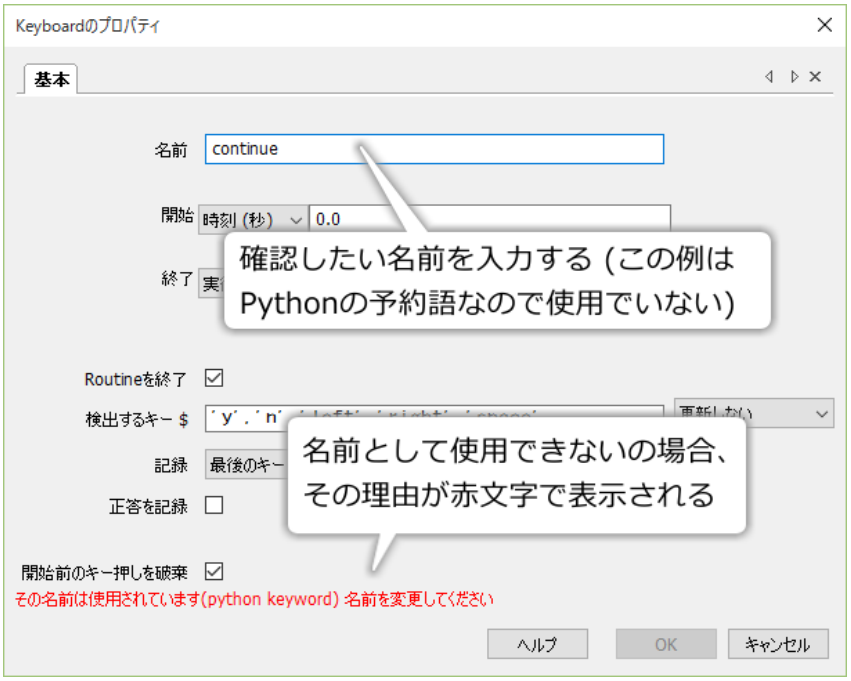


図 3.34　Builder で適当なコンポーネントをルーチンに配置して **名前** に文字列を入力することによって、その文字列が予約されていないかを確認することが出来ます。

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。

3.11.4 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出来ます。

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。疑似乱数にはシード (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。

3.11.5 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 8 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、第 9 章) 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 10 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。

3.11.6 $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$を入力すると条件ファイル内で定義されたプロパティの値を代入することが出来ると述べました。では、Text コンポーネントで「$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出来ます。

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。何も考えずに前後を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、Please から possible まではダブルクォーテーション囲まれていないことになってしまいます。Python は文字列をシングルクォーテーションで囲んでも良いのでシングルクォーテーションを使えば回避できるのですが、もう一つの回避方法として、「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」という方法が用意されています。この方法を使うと、先の文字列は

> “He said, **\”** Please respond as quickly and precisely as possible. **\”** “

と表記することが出来ます。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。

Builder の Text コンポーネントで$を含む文字列を表示したい場合にも、この\によるエスケープが利用できます。つまり、\$と書けば$を表示することが出来ます。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$記号を単なる文字として解釈する」という話はこの問題でも同様ですので、Text コンポーネントの **文字列** プロパティの値を条件ファイルから読み込んで表示する場合にはエスケープをする必要はありません。$を含む文字列をそのまま表示することが出来ます。

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 が割り当てられています。この数値を文字コードと呼びます。困ったことに、現在のコンピュータでは歴史的な経緯から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。

3.11.7 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。

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、その前にこの章で新たに使用する Grating コンポーネントについて解説しておきましょう。第 3 章までの知識で 図 4.1 の刺激を作成するのは困難ですが、Grating コンポーネントを使用すると簡単に作成することができます。

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。Polygon および Text コンポーネントには無くて Grating コンポーネントにあるプロパティは **テクスチャ** 、 **マスク** 、 **位相 (周期に対する比) $** 、 **空間周波数**

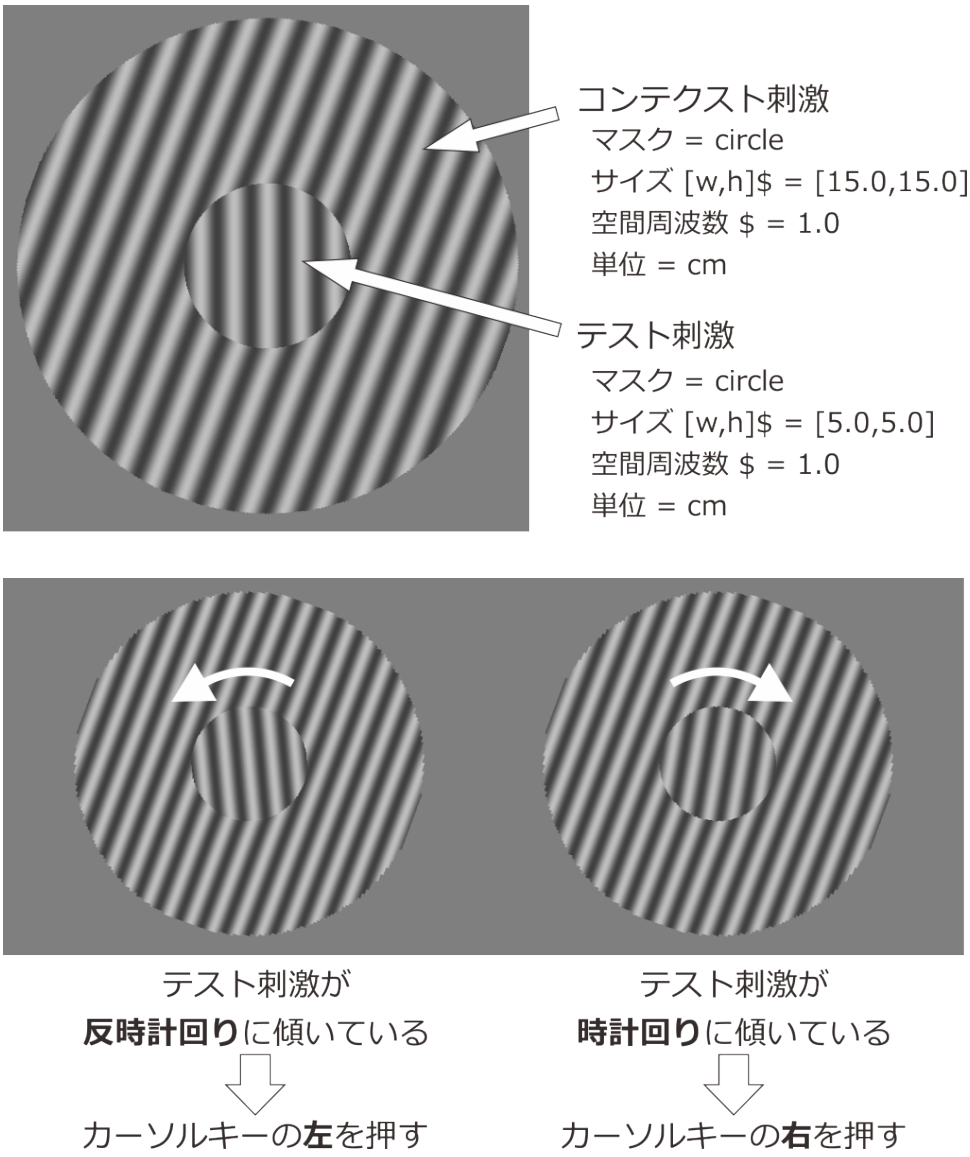


図 4.1　この章の実験。実験参加者は、テスト刺激の縞が垂直方向より反時計回りに傾いているか、時計回りに傾いているかを判断します。テスト刺激の周囲のコンテクスト刺激が判断に与える影響を明らかにするのが目的です。

$ の 4 つです。このうち **テクスチャ** は複雑なので *Grating* **コンポーネントの テクスチャ プロパティについて** を参照していただくとして、残りの 3 つのプロパティを変化させるとどのような刺激が提示されるかを 図 4.2 に示しました。

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。 **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)。 **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 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。 **単位** が pix の場合には複雑で、「 **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。

最後の **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)。

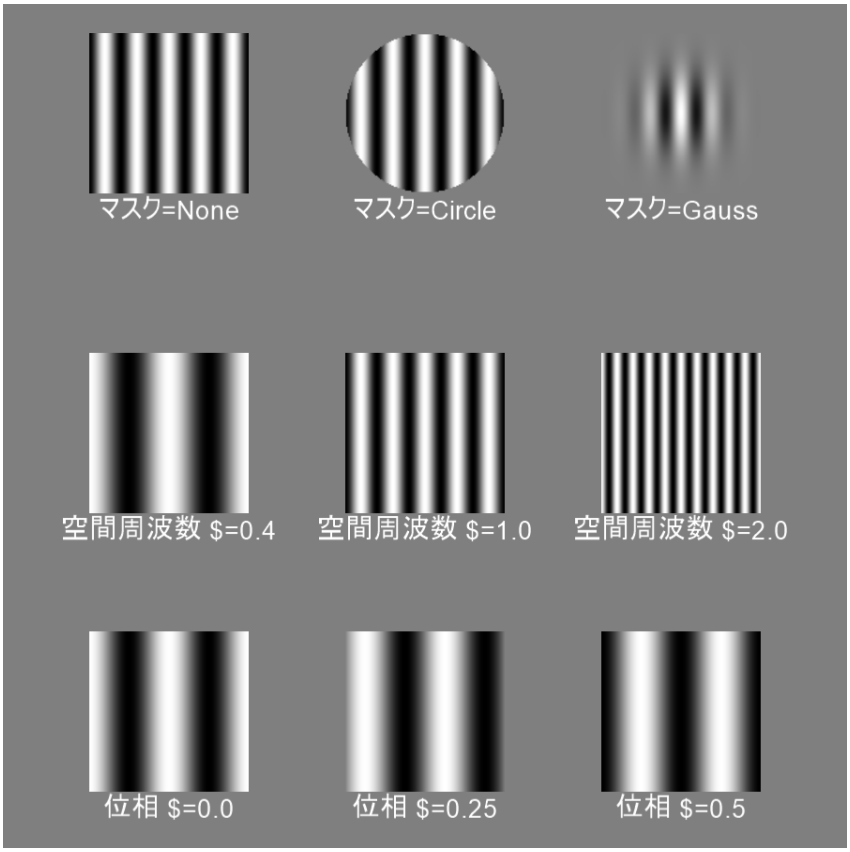


図 4.2　Grating コンポーネントのプロパティのうち、 **マスク** (上段)、 **空間周波数 $** (中段)、 **位相 (周期に対する比) $** を変化させた例。いずれも **サイズ [w, h] $** は [5.0,5.0] で、 **単位** は cm を指定しています。

あと、 **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、 **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 **色** の値が$[1,1,1] であれば-1 から +1 まで変化します。 **色** が$[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。 **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。

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。

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。

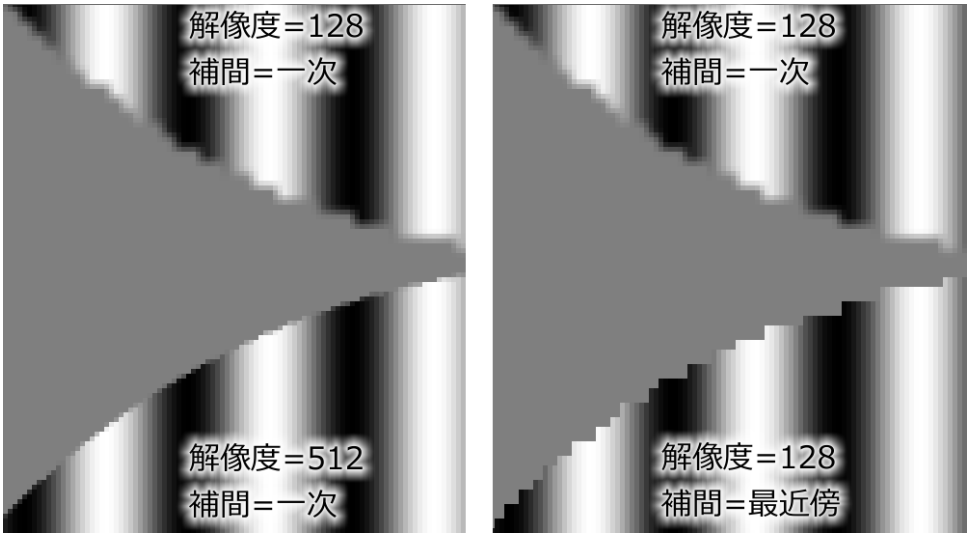


図 4.3　Grating コンポーネントの **テクスチャの解像度 $** および **補間** オプションの効果。

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出来る。
- Grating コンポーネントで **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出来る。
- 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 実験の作成

まず実験に用いる刺激のパラメータを決定しましょう。 図 4.1 に、刺激の図の横にテスト刺激とコンテクスト刺激に使うパラメータが書かれています。 **サイズ [w, h] $** がテスト刺激で 5.0cm、コンテクスト刺激で 15.0cm、そしていずれの刺激も **空間周波数 $** が 1.0 ですから、テスト刺激には 5 周期分、コンテクスト刺激には 15 周期分の縞模様が描かれています。ノート PC を使用している場合など、スクリーンの垂直方向の寸法が 15cm 未満の場合は、コンテクスト刺激がスクリーンに収まるように刺激を小型化してください。例えばテスト刺激とコンテクスト刺激をそれぞれ 4cm と 12cm、あるいは 3cm と 9cm にするといった具合です。テスト刺激を 3cm に縮小する程度であれば **空間周波数 $** は 1.0 のままで構いません。いずれの刺激も **マスク** に circle を指定して円形にします。 **色** はいずれも$[0.5, 0.5, 0.5] にしておきましょう。

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 4.4 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 4.4 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。

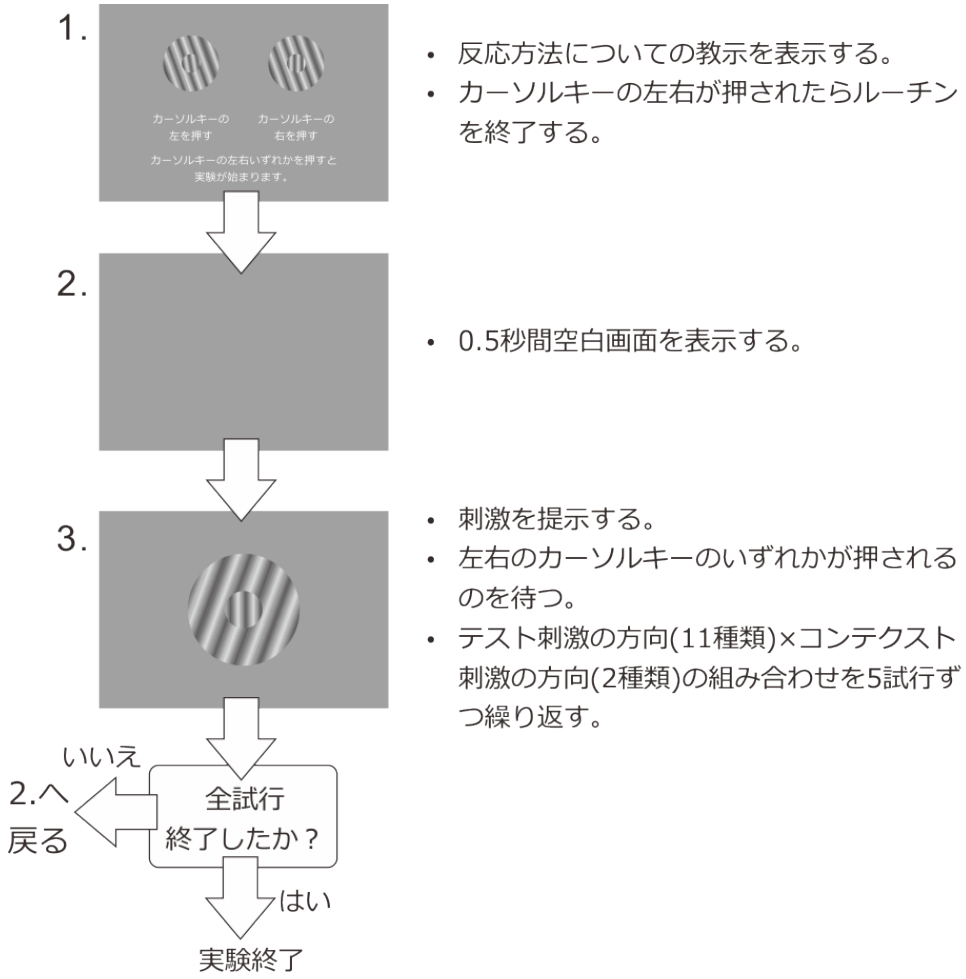


図 4.4　実験の手続き。

さて、以上が手続きですが、第 3 章を終えたみなさんでしたらこの実験を Builder で作成することが出来るはずです。以下にこの章の解説のために作成した実験の作成手順を示します。なお、実験は exp04a.psyexp というファイル名で保存します。先述したとおり、使用するスクリーンが小さくて直径 15cm の刺激を提示できない場合は適当なサイズに縮小してください。

- 実験設定ダイアログ
  - 「xlsx 形式のデータを保存」をチェックする。
  - **単位** を cm にする。
- trial ルーチン
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - 2 つの Grating コンポーネントを配置し、 **名前** をそれぞれ testStim と contextStim とする。 **色** を $[0.5,0.5,0.5] にする。 **マスク** を circle に、 **空間周波数 $** を 1.0 にする。 **テクスチャの解像度 $** を 512 にする。

  - contextStim の上に testStim が表示されるようにする。
  - contextStim の **サイズ [w, h] $** を [15.0, 15.0] にする。 **回転角度 $** を contextDir にして、「繰り返し毎に更新」にする。
  - testStim **サイズ [w, h] $** を [5.0,5.0] にする。 **回転角度 $** を testDir 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、 **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。
  - すべての Grating コンポーネントと Keyboard コンポーネントの **開始** を「時刻 (秒)」で 0.5 に、 **終了** を空白にする。

- instruction ルーチン (作成する)
  - フローの先頭に挿入する。
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - Grating コンポーネントを 4 個配置し、 **名前** を right_large、right_small、left_large、left_small とする。 **色** を$[0.5,0.5,0.5] にする。 **マスク** を circle に、 **空間周波数 $** を 1.0 にする。 **テクスチャの解像度 $** を 512 にする。
  - Grating コンポーネントのうち、名前に large を含むものの **サイズ [w, h] $** を [15.0,15.0] とする。small を含むものの **サイズ [w, h] $** を [5.0,5.0] とする。
  - Grating コンポーネントのうち、名前に left を含むものの **位置 [x, y] $** を [-10.0, 0.0] とする。right を含むものの **位置 [x, y] $** を [10.0, 0.0] とする。
  - Grating コンポーネントのうち、名前に large を含むものの **回転角度 $** を 20.0 にする。
  - Grating コンポーネントのうち、left_small の **回転角度 $** を-10.0 にする。right_small の **回転角度 $** を 10.0 にする。
  - Text コンポーネントを 3 個配置して、 **名前** を TextCC、TextC、TextInst とする。
    - それぞれの **文字の高さ $** を適当な値 (1.0 など) にする。
    - TextCC の **位置 [x, y] $** に [-10, -8]、 **文字列** に「反時計回り」と入力する。
    - TextC の **位置 [x, y] $** に [10, -8]、 **文字列** に「時計回り」と入力する。
    - TextInst の **位置 [x, y] $** に [0, -10] と入力する。 **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押して反応するように教示するメッセージを入力する。
  - Keyboard コンポーネントをひとつ配置し、 **検出するキー $** を'left', 'right' にして **記録** を「なし」にする。
  - 全ての Grating コンポーネント、Text コンポーネント、Keyboard コンポーネントの **終了** を空白する。

- 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。
  - **繰り返し条件** に exp04_20.xlsx と入力する。
- 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。

さて、これで各ブロックの手続きは完成です。以上の手順を見ずに自力で作成した方も、条件ファイルの設定、特に最後の項目「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。

4.4 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 4.5 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。

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、xlsx 記録ファイルや summaries 記録ファイルの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。とても便利なテクニックですので、ぜひ覚えておいてください。

では、皆さんも完成した実験を実行して、xlsx 記録ファイルを用いて心理物理曲線を描いてみてください。余裕がある人は、一度 trial-by-trial 記録ファイルから心理物理曲線を描くデータ処理も行ってみるとよいでしょう。

チェックリスト

- 恒常法の実験において、xlsx 記録ファイルまたは summaries 記録ファイルの正答率の値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。

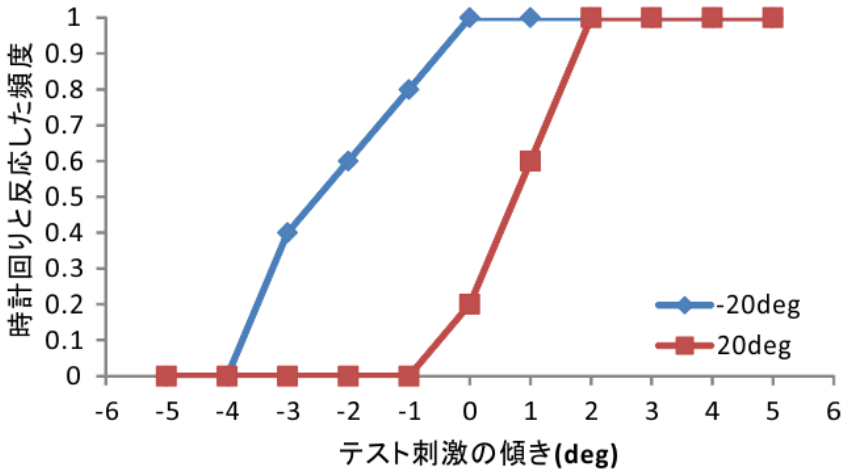


図 4.5　心理物理曲線の例。横軸にテスト刺激の傾き、縦軸に時計回りに傾いているという反応の頻度をプロットしています。折れ線グラフはそれぞれコンテクスト刺激の傾きが-20 度と 20 度の条件に対応しています。

4.5 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第 4 章の本題です。exp04a.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。

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出来ます。 図 4.6 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。exp04a.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、その行の上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 のどちらの行の上に追加しても同じ結果が得られますので、行を追加する時にはどの [+] を押しても結構です。

では、[+] を押して新しい行を追加して、Field に cndFileName、Default 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 4.6 の一番下の図のように実験情報ダイアログに

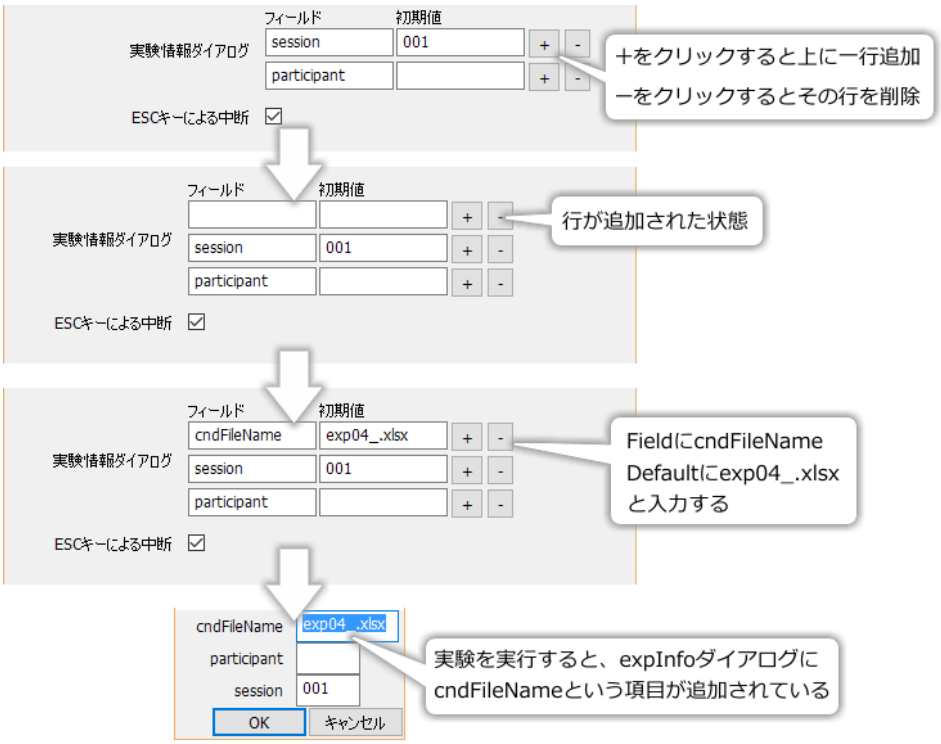


図 4.6　実験情報ダイアログに項目を追加する手順。

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。Field は実験情報ダイアログに表示する項目の名前、Default 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 Default 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出来ます。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 PsychoPy が自動的に決定します。言い換えると、利用者が自由に並び替えることは出来ません。

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出来ます。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 exp04a.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、**繰り返し条件** を$expInfo['cndFileName'] に変更してください。**繰り返し条件** には$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 4.7)。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。実験を exp04b.psyexp の名前で保存しておきます。

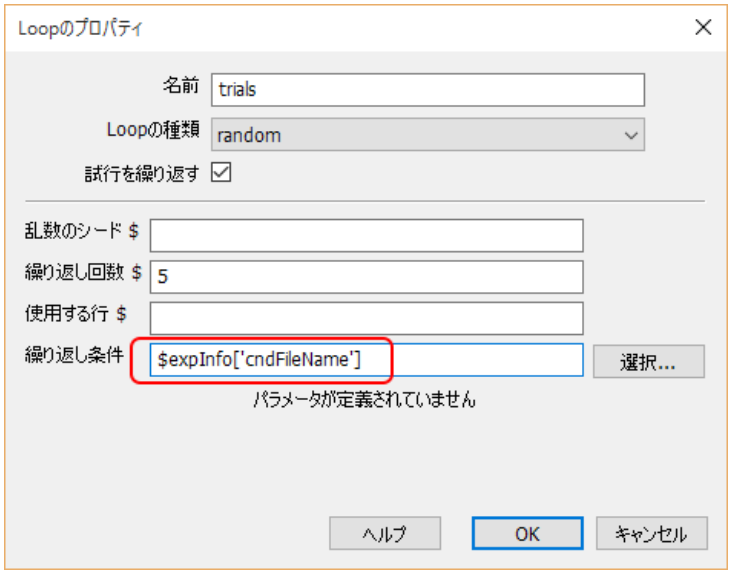


図 4.7　**繰り返し条件** に実験情報ダイアログの項目を指定します。先頭の$と項目名の前後の' を忘れないように注意してください。

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 exp04b.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 exp04b.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、削除することが出来る。
- 実験情報ダイアログの項目の初期値を設定することが出来る。
- 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出来る。
- 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出来る。

4.6 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、exp04b.psyexp を改造してみましょう。

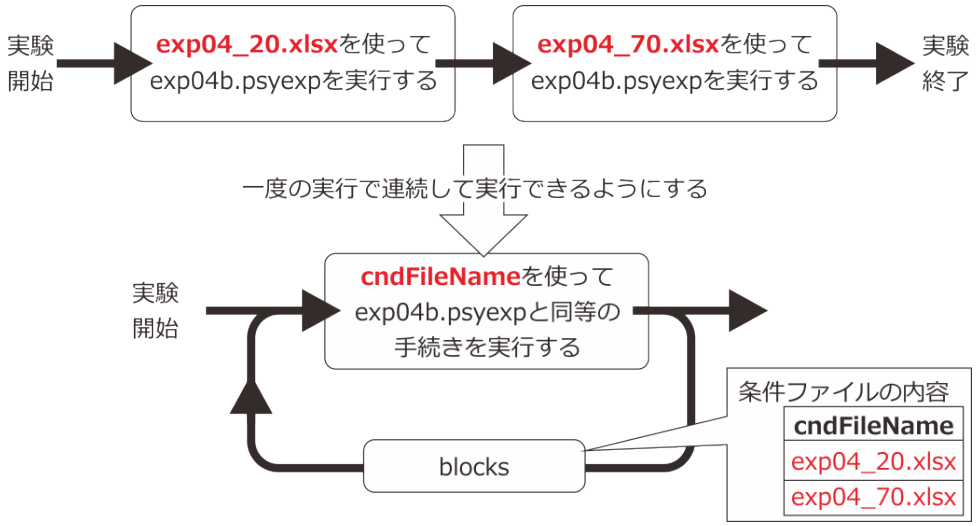


図 4.8　読み込む条件ファイルを変更しながら繰り返すことによって、-20 度／ 20 度と-70 度／ 70 度の条件を連続して実行する実験を作成します。

図 4.8 に改造の方針を示します。exp04b.psyexp を使う場合は、 図 4.8 の上の図のように、条件ファイル名を変更しながら 2 回 exp04b.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出来たように、図 4.8 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出来るはずです。

さっそく改造してみましょう。Builder で exp04b.psyexp を開いて、念のため別名で保存しておきましょう (exp04c.psyexp とします)。そして、 図 4.9 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、 **繰り返し条件** に exp04blocks.xlsx と入力しておきます。そして、**試行を繰り返す** のチェックを忘れずに外してください。この項目の意味については後で触れます。 **Loop の種類** は random のままでよいで

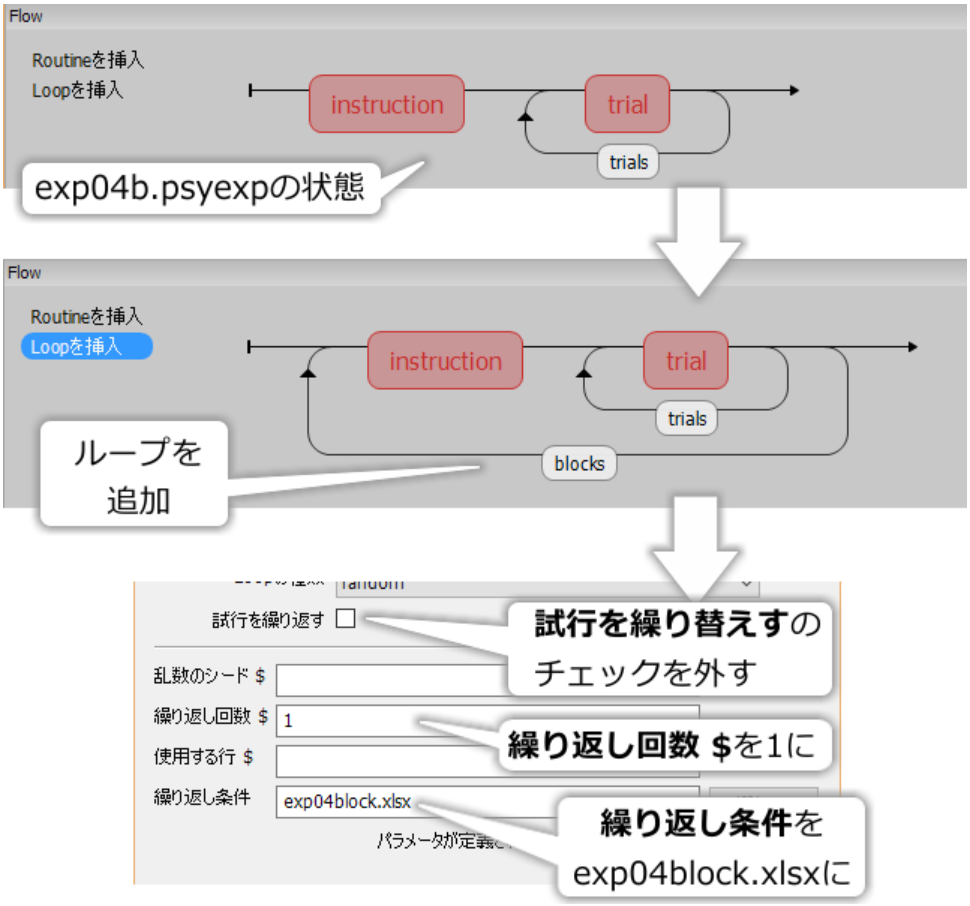


図 4.9　多重に繰り返しを挿入します。

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 4.10 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 **繰り返し条件** を$cndFileName に変更します (図 4.10 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 4.10 下)。

以上で改造は終了です。exp04c.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。

exp04c.psyexp を実行することによって作成される xlsx 記録ファイルの例を 図 4.11 に示します。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 4.11 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

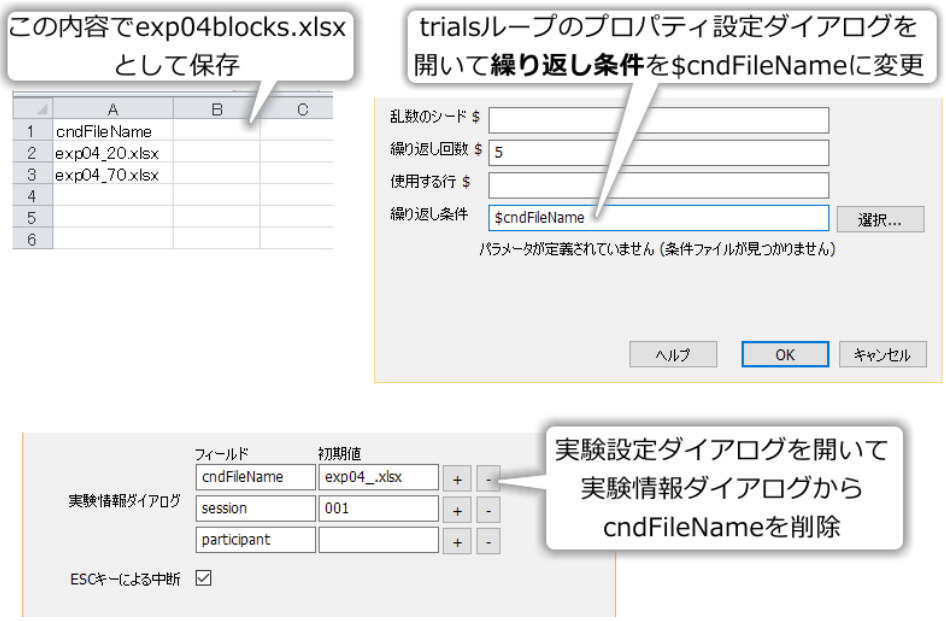


図 4.10　blocks ループを使って条件ファイルを切り替えるように trials ループなどを修正します。

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。

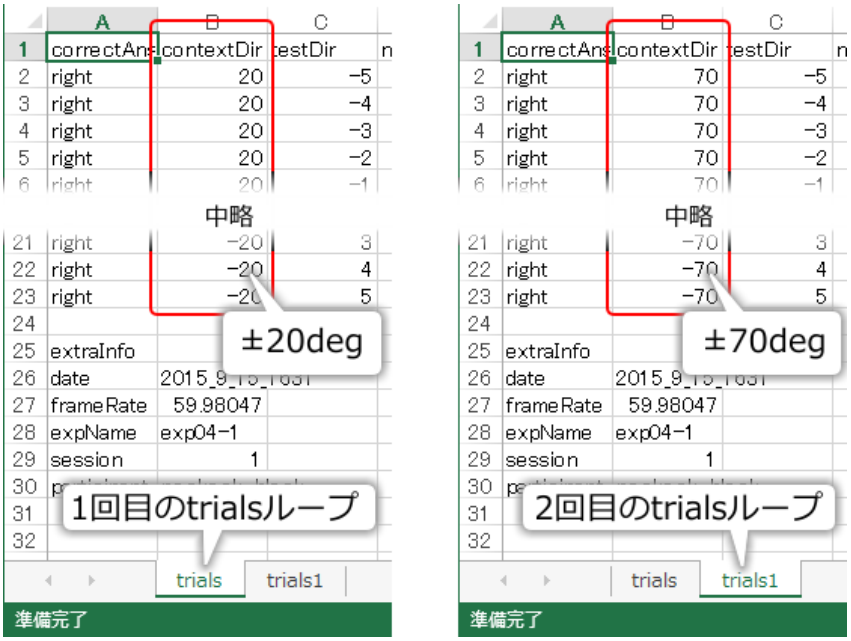


図 4.11　多重ループを用いた実験によって作成される xlsx 記録ファイルの例 (**試行を繰り返す** をチェックしない場合)。

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

先ほど説明を飛ばした blocks ループの **試行を繰り返す** というプロパティですが、これ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 4.12 に示します。trials、trials1 というシートが出来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出来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。

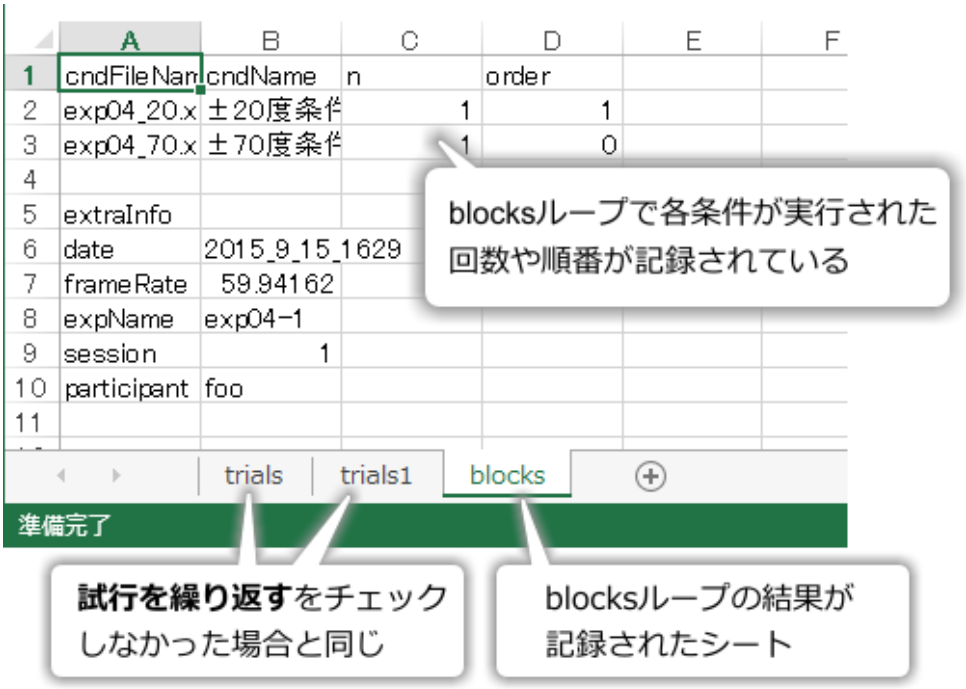


図 4.12　多重ループを用いた実験によって作成される xlsx 記録ファイルの例 (**試行を繰り返す** をチェックした場合)。

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 4.13 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。

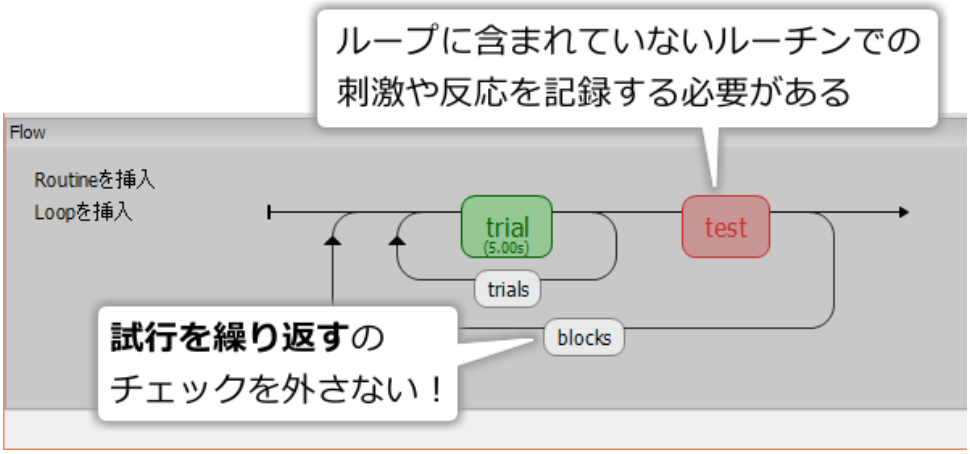


図 4.13　**試行を繰り返す** のチェックを外してはいけない例

以上で第 4 章の解説は終了です。多重繰り返しを活用することによって、ブロック化された手続きの実験を簡単に作成することができますのでぜひマスターしてください。

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。
- 多重繰り返しで xlsx 記録ファイルに無駄なシートが作成されないようにすることができる。

4.7 練習問題：条件ファイルを使う順番を指定できるようにしよう

exp04c.psyexp では blocks ループの **Loop の種類** に random を指定したので、実行の度にどちらの条件が先に実行されるかが無作為に変化します。ところが、この章の実験のようにブロックの順序を実験参加者間で変更する場合は、各順番に同数の参加者が割り振られるようにバランスを取ることがよく行われます。つまり、実験参加者が 20 人いるとしたら、10 人は-20 度／ 20 度条件を先に、残り 10 人は-70 度／ 70 度条件を先に行うようにするということです。実行順序を無作為に決定してしまうと、9 人と 11 人といった具合にバランスが崩れてしまう可能性があります。確実にバランスが取れるようにするには、-20 度／ 20 度条件が先になるように書かれた条件ファイルと、-70 度／ 70 度条件が先になるように書かれた条件ファイルを用意して、どちらの条件ファイルを使用するかを実験情報ダイアログで指定するという方法があります。第 4 章のまとめとして、exp04c.psyexp を改造してこの方法で実験参加者数のバランスを取ることができる実験を作成してください。さらに加えて、trials ループを開始する前に、これから行われるブロックについての情報を表示するルーチンも追加してください。

文章だけではわかりにくいので、図 4.14 をご覧ください。まず、exp04c.psyexp をベースとして、実験情報ダイアログにブロックの順番を指定するための項目を設けてください。そしてこの項目に例えば exp04blocks.xlsx

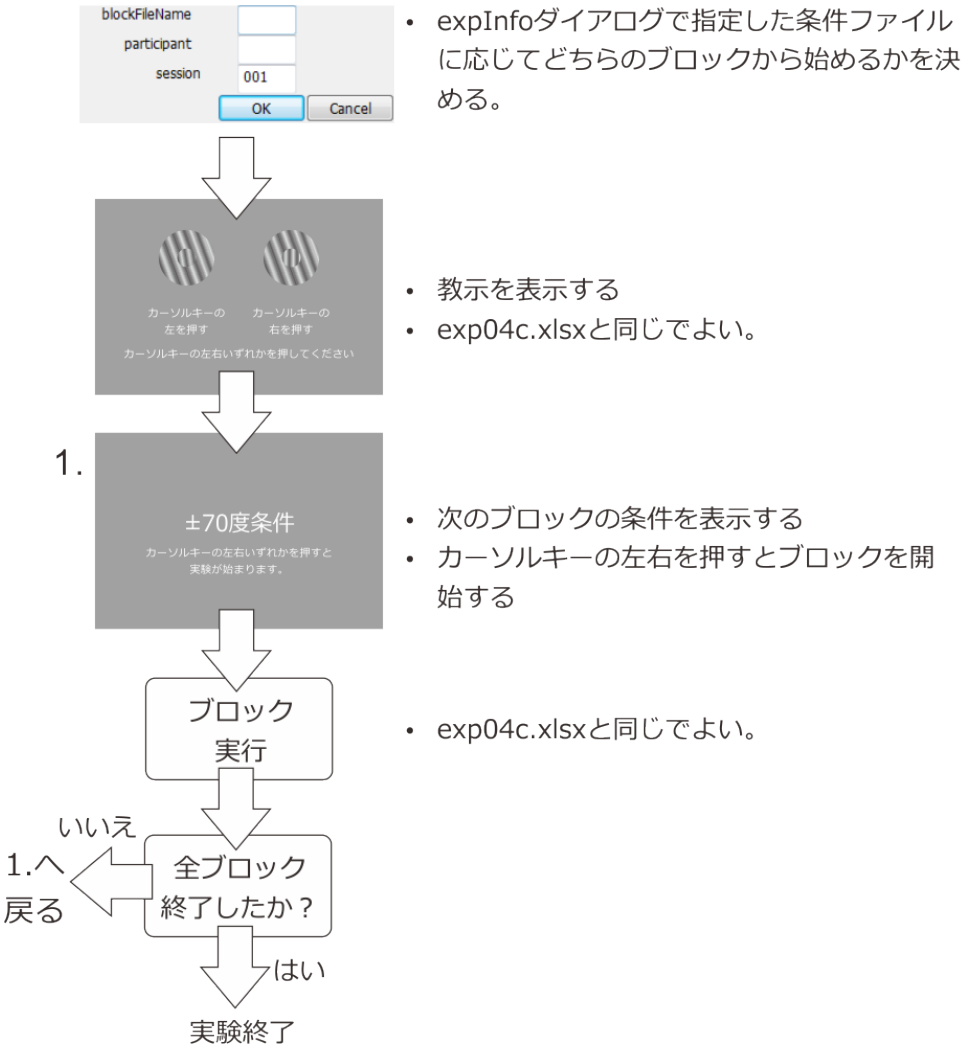
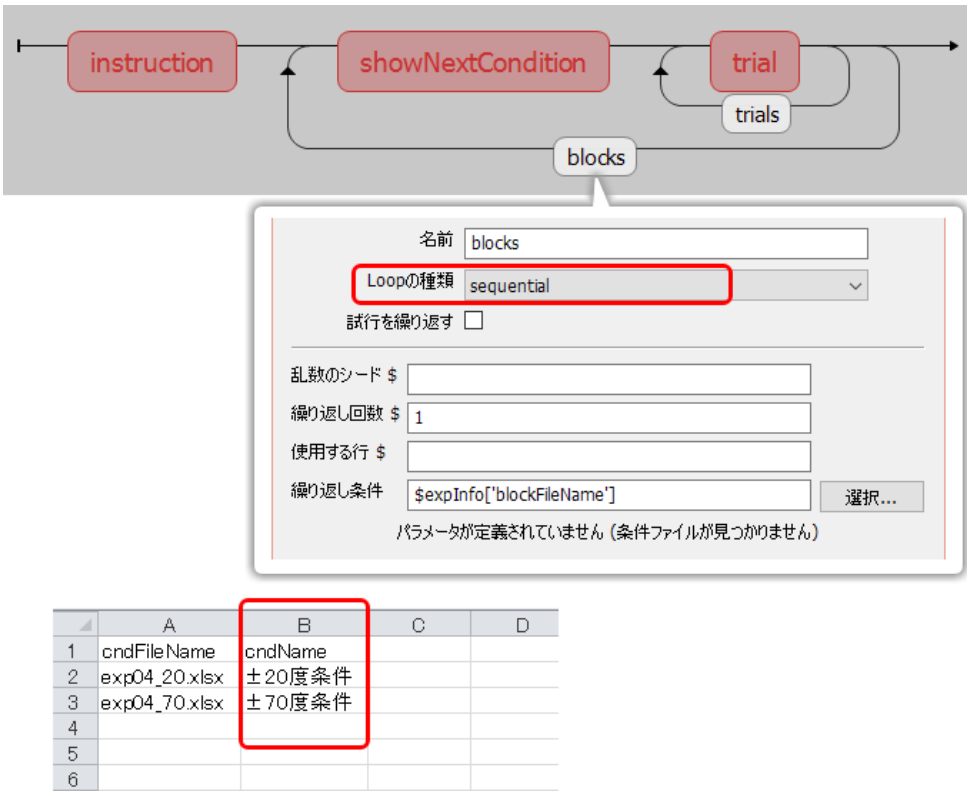


図 4.14　練習問題で作成する実験の流れ

を指定したら-20 度／ 20 度条件から始まって、exp04blocks2.xlsx を指定したら-70 度／ 70 度条件から始まるようにしてください。そして、教示画面を表示した後に、これから始まるブロックが-20 度／ 20 度条件と-70 度／ 70 度条件のどちらなのかを示す画面を表示してください。最初のブロックが終了した後は、教示画面ではなくこれから始まるブロックを示す画面に戻るようにしてください。以上が練習問題です。

ちょっとこれだけでは難しいという方向けに、 図 4.15 にヒントを示します。条件ファイルでブロックの順番を指定するためのポイントは、blocks ループの **Loop の種類** の値です。ブロックの順番を指定するための条件ファイルは、 図 4.15 下のようなファイルと、2 行目と 3 行目を入れ替えたファイルを用意すればよいでしょう。なぜこれでうまくいくかわからない方は、第 3 章の **Loop の種類** の解説を読み直してください。健闘を祈ります。



	A	B	C	D
1	cndFileName	cndName		
2	exp04_20.xlsx	±20度条件		
3	exp04_70.xlsx	±70度条件		
4				
5				
6				

図 4.15　練習問題のヒント。

4.8 この章のトピックス

4.8.1 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 4.16 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、 **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 4.17 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。

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出来ます。 図 4.17 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

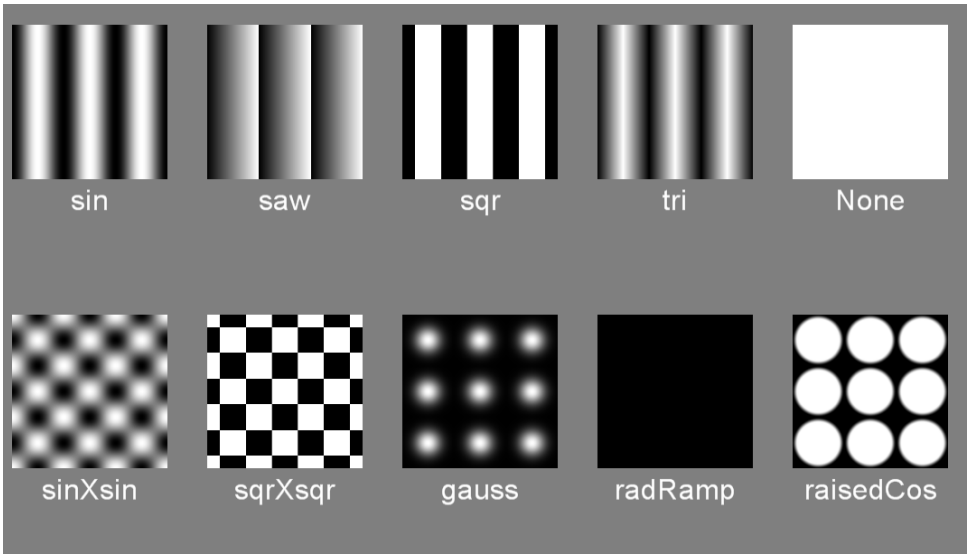


図 4.16　Grating コンポーネントの **テクスチャ** として指定できる値。これらの他に画像ファイル名を指定することが出来ます。いずれも **単位** は height で **空間周波数 $** は 3.0 を指定しています。

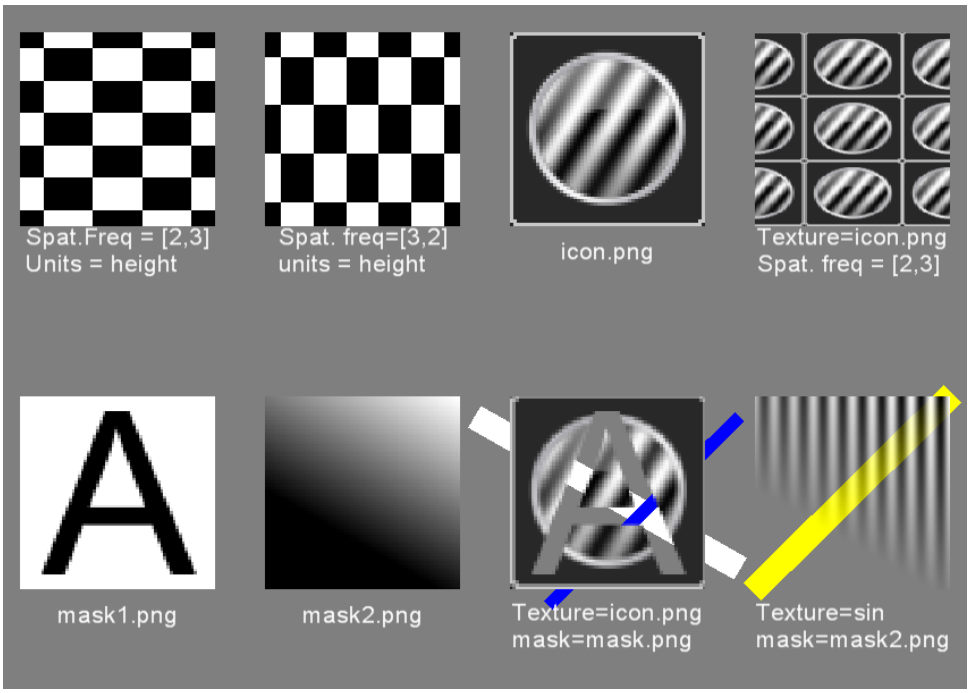


図 4.17　**空間周波数 $** に 2 つの値を指定する例と (上段の左側 2 つ)、 **テクスチャ** に画像ファイルを指定する例 (上段の右側 2 つ)、 **マスク** に画像ファイルを指定する例 (下段)。

ります。文章では説明しにくいので 図 4.17 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 4.17 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 4.17 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。

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) で実験を作成出来る事が 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「あれが出来ない、これが出来ない」という事だらけであまり役に立ちません。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「出来る事」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適しているので取り上げました。

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、 **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。

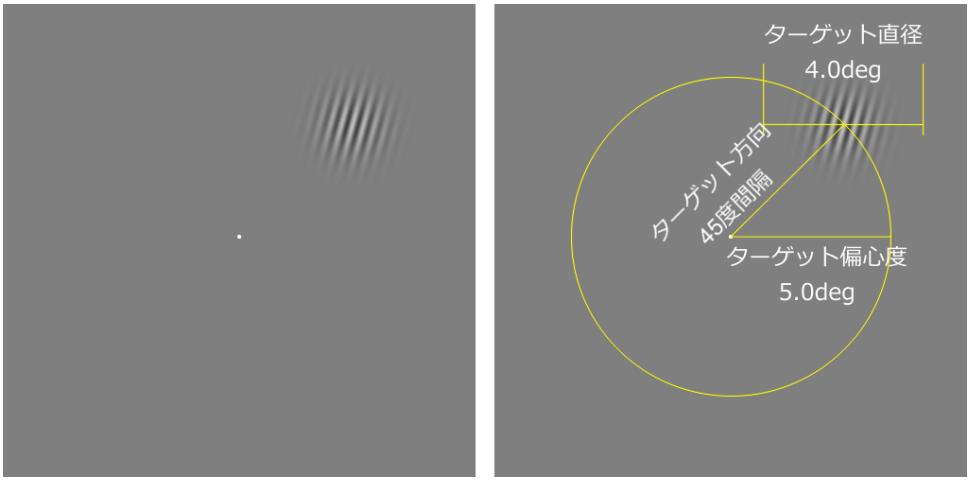


図 5.1　実験に使用する刺激。

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$[0,0,0]、すなわち背景と全く同一で知覚することが出来ませんが、6 秒間かけて一定の速度で$[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出来たら出来るだけ速く、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出来たと考えられます。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出来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 をマスターするための例題だと考えてください。

第 4 章までに解説したテクニックでは、時間の経過とともにだんだんコントラストが上昇するグレーティングを提示するのは困難です。持続時間が 0.1 秒程度のルーチンを作成して、0.1 秒毎のコントラストの値を並べた条件ファイルを作成して、sequential のループを用いて次々とコントラストを変更していけば提示自体は不可能ではありません。しかし、キーが押されたらその時点で提示を中断するのはこの方法では出来ませんし、なにより実験記録ファイルに全試行 0.1 毎にキーが押されただの押されなかっただのという記録が出力されてしまうのでデータ処理の時点で悲惨なことになります。この章では、Builder の実験に python のコードを組み

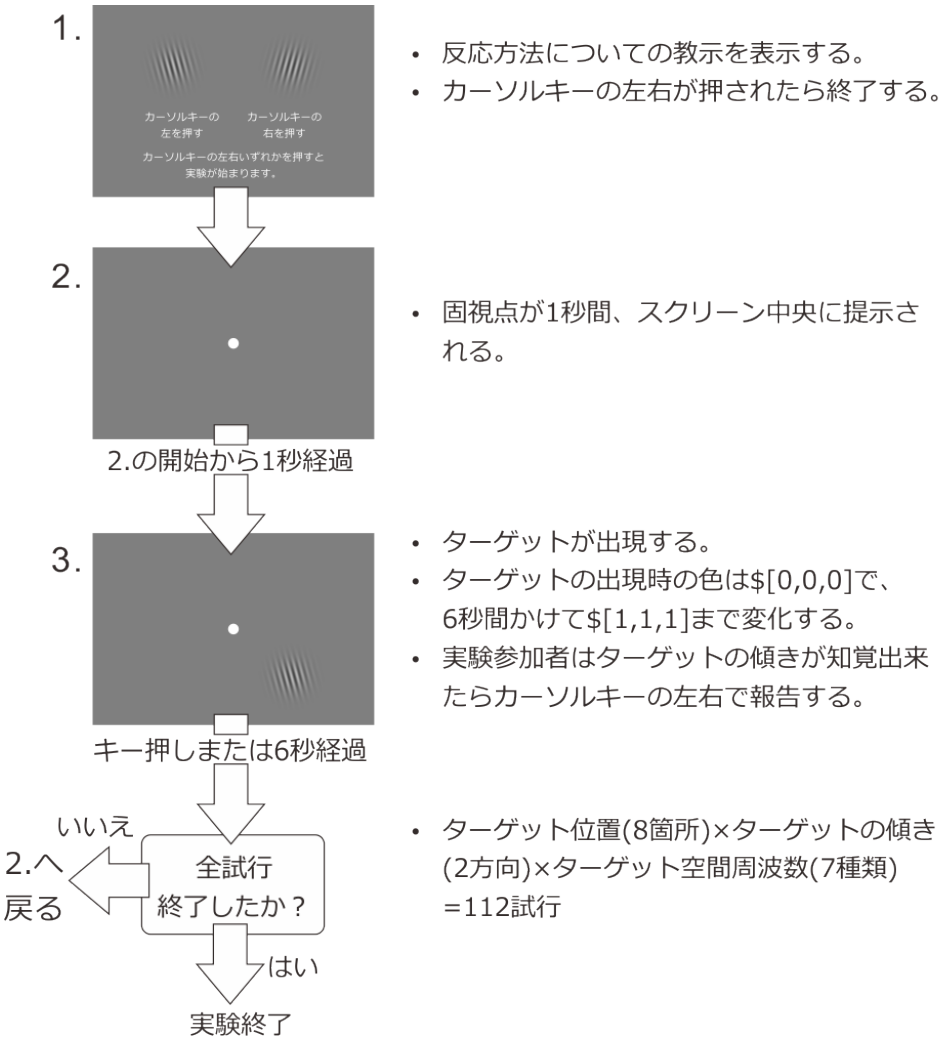


図 5.2　実験の手続き。

込むことで問題を回避します。まずは、第 4 章までに解説済みの作業で出来るところまで実験を作成しましょう。以下のように実験を作成して exp05a.psyexp の名前で保存するものとします。

- 実験設定ダイアログ
  - 「xlsx 形式のデータを保存」をチェックする。
  - **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。
- trial ルーチン
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - 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。
    - **名前** を targetStim にする。
    - **マスク** を gauss にする。

        * **回転角度 $** を targetDir にして、「繰り返し毎に更新」にする。
        * **空間周波数 $** を targetSF にして、「繰り返し毎に更新」にする。
        * **サイズ [w, h] $** を [4.0, 4.0] にする。
        * **テクスチャの解像度 $** を 512 にする。
    - Keyboard コンポーネントをひとつ配置し、 **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。
    - Polygon コンポーネントをひとつ配置し、 **名前** を fixation にする。 **頂点数** を 12 に、 **サイズ [w, h] $** を [0.1, 0.1] にする。
    - すべてのコンポーネントの **終了** を実行時間 (秒) で 6.0 にする。
- instruction ルーチン (作成する)
    - フローの先頭に挿入する。
    - 最初から Static コンポーネントが配置されている場合は削除する。
    - Grating コンポーネントを 2 個配置し、 **名前** を right_sample、left_sample とする。以下のように設定する。
        * **マスク** を gauss にする。
        * **空間周波数 $** を 1.0 にする。
        * **サイズ [w, h] $** を [4.0, 4.0] にする。
        * **テクスチャの解像度 $** を 512 にする。
        * left_sample の **位置 [x, y] $** を [-5.0, 0.0] とする。right_sample の **位置 [x, y] $** を [5.0, 0.0] とする。
        * left_sample の **回転角度 $** を-10.0 にする。right_sample の **回転角度 $** を 10.0 にする。
    - Text コンポーネントを 3 個配置して、Letter height $を適当な値 (0.5 など) にする。ひとつを left_sample の下に位置するようにして **文字列** に「反時計回り」と入力する。別のひとつを right_sample の下に位置するようにして **文字列** に「時計回り」と入力する。最後に残った Text コンポーネントの **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出来るだけ速く押して反応するように教示するメッセージを入力する。
    - Keyboard コンポーネントをひとつ配置し、 **検出するキー $** を'left', 'right' にして **記録** を「なし」にする。
    - 全てのコンポーネントの **終了** を空白する。
- 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。

  - 最初から Static コンポーネントが配置されている場合は削除する。
  - Polygon コンポーネントをひとつ配置し、**名前** を fixation_2 にする。**頂点数** を 12 に、**サイズ [w, h] $** を [0.1, 0.1] にする。**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 ルーチン繰り返すように挿入する。
  - **繰り返し回数 $** の値を 1 にする。
  - **繰り返し条件** 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の解説がかなり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 **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出来ます。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出来ます。C 言語のように変数に「型」がある言語

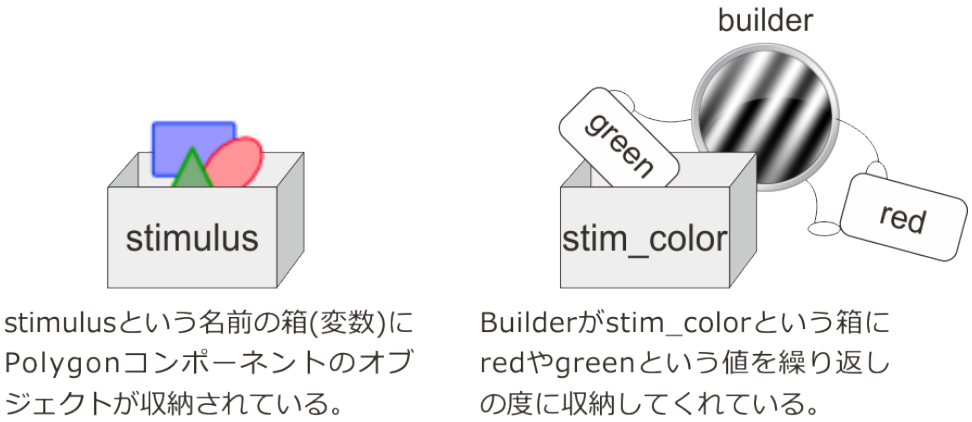


図 5.3　変数とはいろいろなものを収納しておける箱のようなもの。

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、扱うことが出来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 章で刺激の色を RGB 値で指定したり、刺激の位置や大きさを指定したりする時に「両端の角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出来ます。[ ] が欠けているとリストとして解釈することが出来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。python 内部では ASCII 型と Unicode 型の文字列があり、Builder では通常 Unicode 型の文字列 (第 6 章) が用いられる。
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出来る。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。
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。囲んで第 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 の値を得ることが出来ます (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。

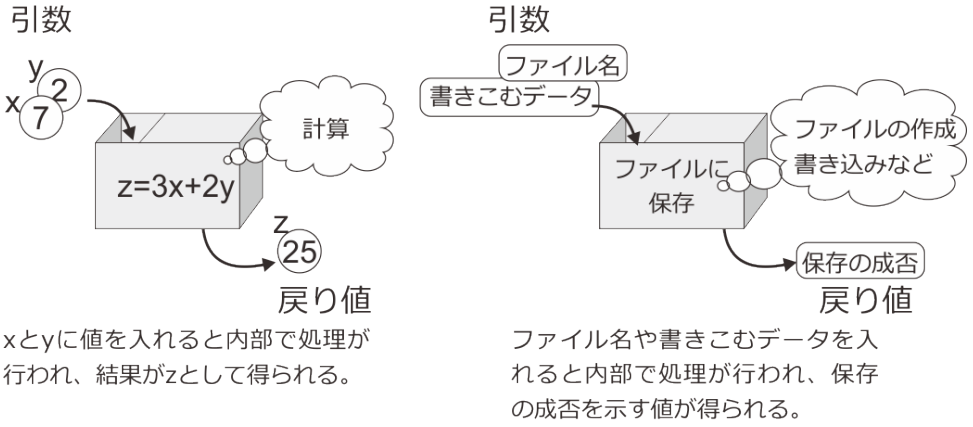


図 5.4　数学で習った関数と python における関数。

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 exp04a.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 $\times$ 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出来ます。

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には第 6 章で解説するコンポーネントを使う必要があります。この点については **他の数学関数を使用する方法** も参照してください。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」という計算が出来ます (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。

これで 45 度の sin と cos を計算する式が出来ましたが、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出来ます。 **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)

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。

算術演算子の話が出てきたついでに、python の初心者が陥りがちなポイントについて補足しておきます。普通、分数を使わずに 4 ÷ 3 の答えを書けと言われたら 1.33333… と答えると思うのですが、PsychoPy のベースになっている python2 では整数同士の割り算の結果は整数になるよう切り捨てられます。つまり、4 / 3 の値は 1 となります。切り捨てですから 5 / 3 の値も 1 です。このことに注意しておかないと思い通りの位置に刺激が出現しないなどのトラブルにつながります。小数点以下まできちんと計算してほしい時には 4 / 3.0 または 4.0 / 3、4.0 / 3.0 といった具合に小数の割り算にしてください。

さて、ターゲットの方向が 45 度の時の式を完成させましたが、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出来るので、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出来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出来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出来る
- 関数の引数に別の関数の戻り値を使うことが出来る。
- 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出来る。
- python の算術演算子 8 つ挙げてその働きを説明することが出来る。
- 4 ÷ 3 といった整数同士の割り算で、小数点以下を切り捨てない解を得る式の書き方を答えられる。

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 **色** の値を変更する」ことと、「 **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 exp05a.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 Static コンポーネントが最初から配置されている場合は削除して、Text コンポーネントをひとつ配置します。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、 **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については第 9 章で扱います。

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 exp05a.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 **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。

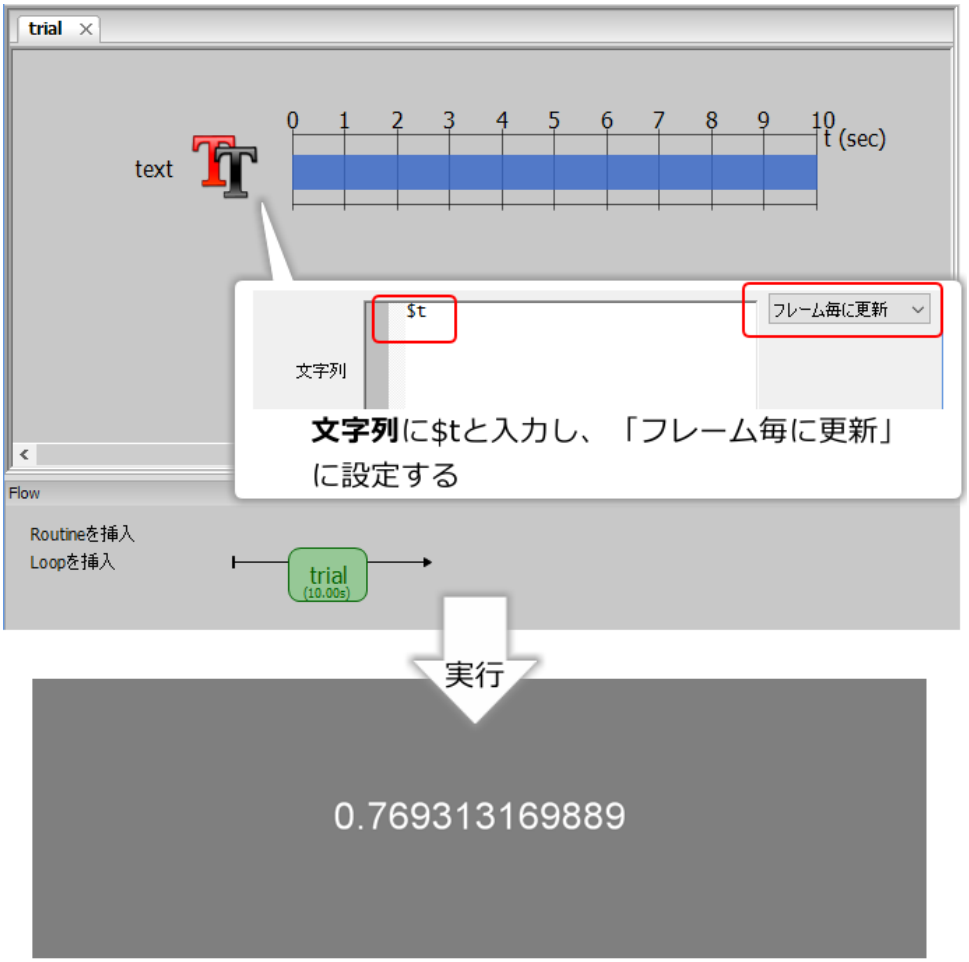


図 5.6　Builder の内部変数 t のテスト。画面上にルーチン開始から経過した時間が表示されます。

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 **時刻指定における *frame* について** を参考にしてください。

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 の内部変数を答えられる。

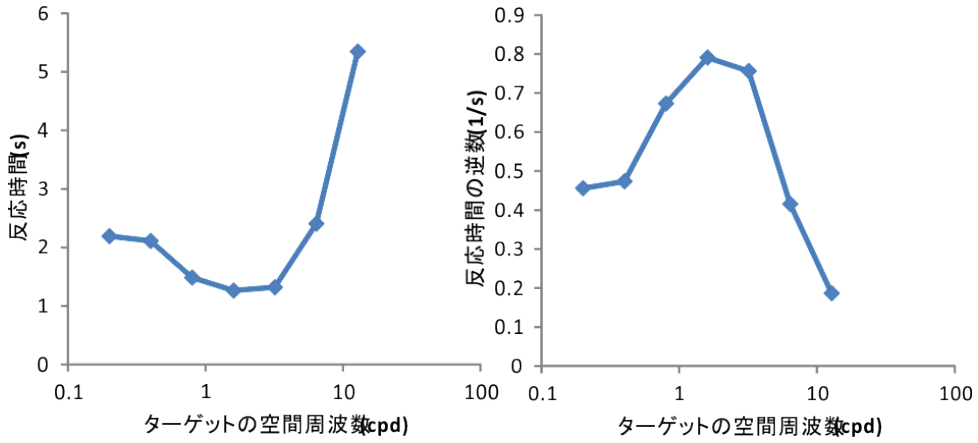


図 5.7　実験結果の例。左は反応時間、右は反応時間の逆数を縦軸にプロットしています。横軸が対数軸になっている点に注意してください。

- 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出来る。

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。この点を解説するために、exp05a.psyexp を改造して taregtStim の大きさと、固視点からの距離を実験情報ダイアログで変更できるようにしてみましょう。exp05a.psyexp を開いて exp05b.psyexp という別名で保存し直してください。

図 5.8 に作業の概要を示します。実験情報ダイアログに追加する項目のうち、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 eccentricty 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 章を復習しましょう。

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、5 を expInfo['eccentricity'] に書き換えればよいはずです。

```
[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']]
```

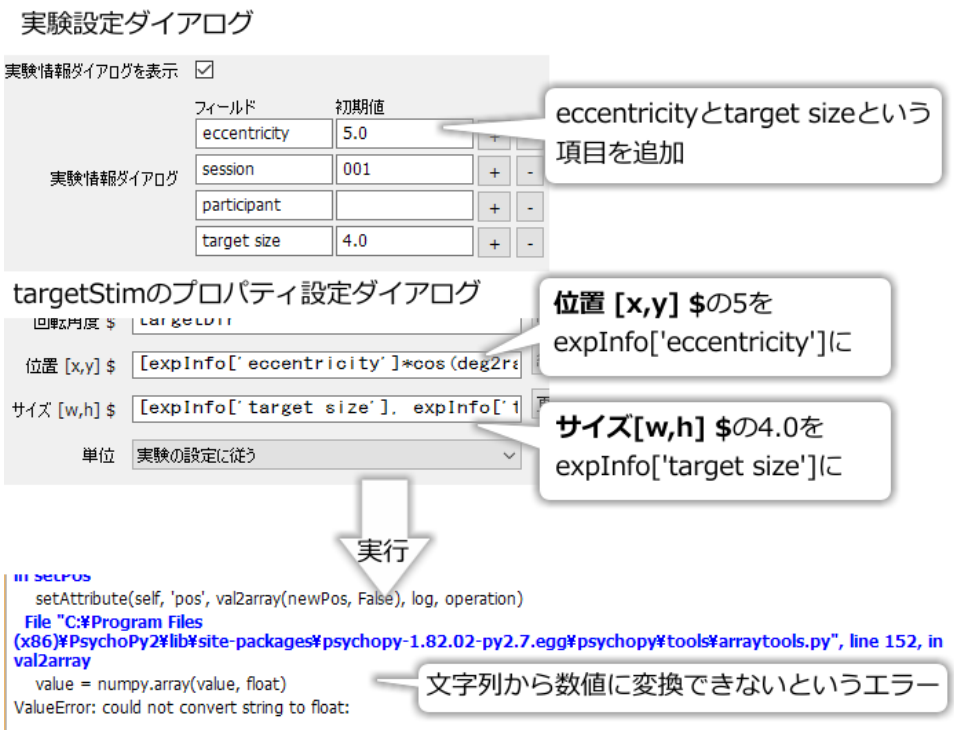


図 5.8　実験情報ダイアログから数値を読み込んでプロパティに設定するとエラーが表示されます。実験情報ダイアログに入力されたものが数値ではなく文字列となっているのが原因です。

これで改造は終了のはずなのですが、いざ実験を実行してみると、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 下)。英語のエラーメッセージで難しそうですが、一番下の行を読むと could not convert string to float と書かれています。string とは文字列、float とは浮動小数点数 (表 5.1) のことですから、文字列を浮動小数点数に変換できなかったということです。どういう事かといいますと、例えば eccentricity に 4.0 と入力されていた場合、Builder には 4.0 という数値ではなく「4」、「.」、「0」という三文字の文字列に見えているということです。人間としては「数値を入力したいに決まっているだろう、そのぐらい察してくれよ」と言いたくなりますが、Builder はそのような配慮はしてくれません。仕方がないので、明示的にこれは数値として解釈しなさいと Builder に教えてなければいけません。そのために使用できるのが 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出来ます。

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 (第 9 章) 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。

これで改造は完了ですが、実験情報ダイアログについて触れたついでに xlxs 記録ファイルへの出力についてひとつ補足しておきます。exp05b.psyexp を実行して出来た xlsx 記録ファイルを見ると、ワークシートの最後に 図 5.9 のように実験情報ダイアログに入力した値が記録されているのがわかります。分析の際に役に立つことも多いので覚えておいてください。

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出来る。
- 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出来る。
- 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出来る。

5.6 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、exp05a.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 **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。

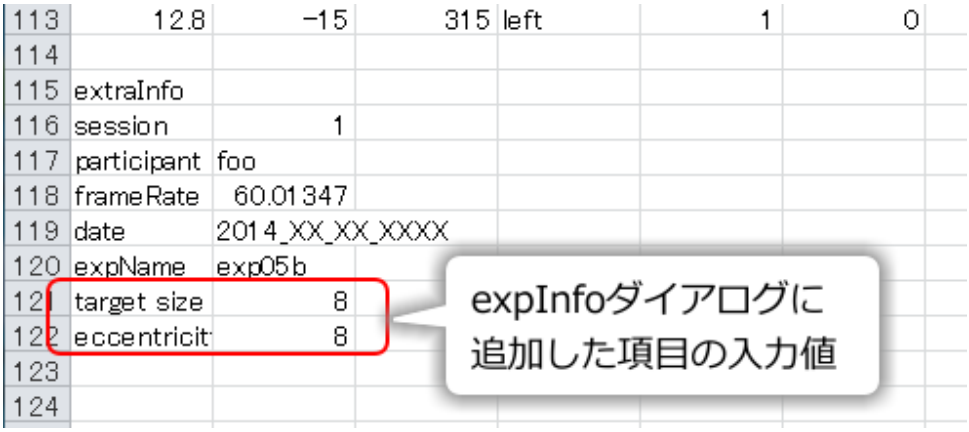


図 5.9　xlsx 記録ファイルを開くと、ワークシートの最後に実験情報ダイアログに入力された値が記録されています。

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、exp05a.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 **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、 **色** に入力した式を exp05a.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。 **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出来ます。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出来ます。

5.7 この章のトピックス

5.7.1 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、第 6 章で紹介する 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 コンポーネントについて詳しくは第 6 章を参照していただきたいのですが、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。

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『ぶーぶー』じゃないよ」、「いま『ぶーぶー』がみえたね」といった事例を通じて「ぶーぶー」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

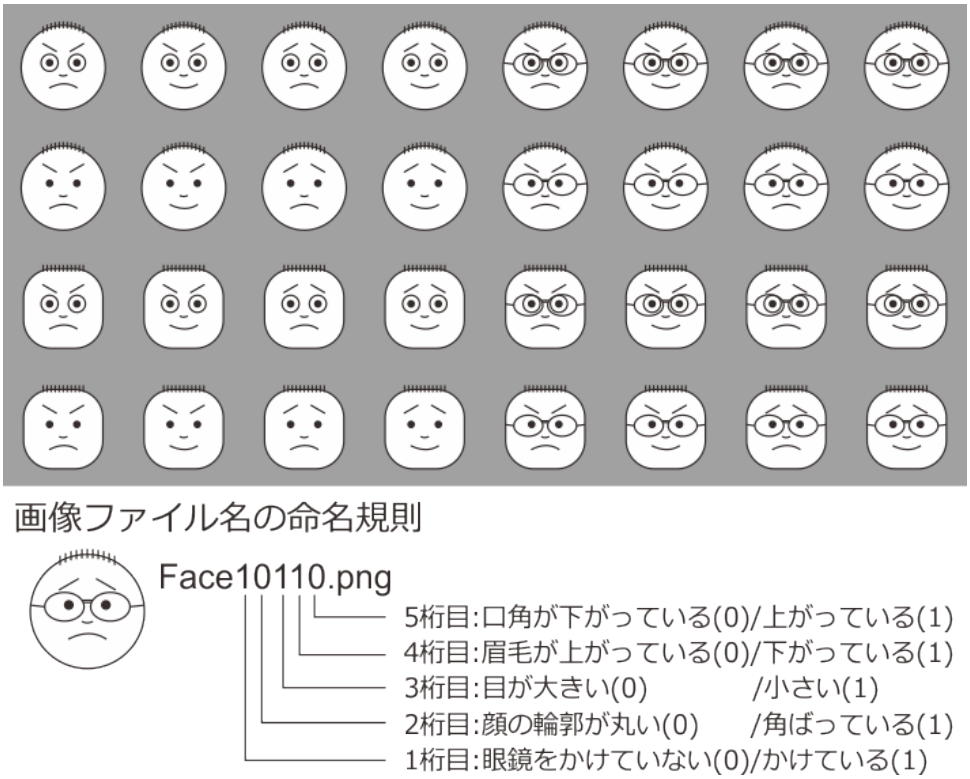


図 6.1　実験に用いる刺激。5 次元の特徴にそれぞれ 2 種類の値があり、合計 2 の 5 乗=32 種類の画像あります。

たら正事例の条件がどれだけわかったか、確信度を 1 から 7 の 7 段階尺度 (1 が最も確信度が低い) で回答させます。以上の手続きを 1 セッションとします。セッション終了後に、実験参加者に正事例の条件を口頭で回答させます。実験者は回答をメモ用紙に記録し、正しかった場合はそこで実験を終了します。回答が誤っていた場合はもう 1 セッション実験を実行します。5 セッション実行しても正解しなかった場合は、そこで実験を終了します。Builder では 1 セッション分のフローをひとつの実験として作成し、1 セッション実行する毎に Builder から実験を実行するものとします。

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について解説しましょう。画像を提示する Image コンポーネント、尺度を表示して回答を記録する RatingScale コンポーネントの二つです (図 6.3)。

6.2 Image コンポーネント

Image コンポーネントは画像ファイルをスクリーン上に提示するコンポーネントです。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 5 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、ひとつだけ注意が必要なプロパティが **色** です。Image を空白にして **色** を指定すると、 **色** で指定した色で塗りつぶされた長方形が提示されます。古いバージョンの Builder では Polygon コンポーネントが存在せず、代わりにこの Image コンポーネントを使って長方形を提示していました。Image で画像ファイルを指定して、なおかつ **色** を指定すると、 **色** で指定した色で画像を着色

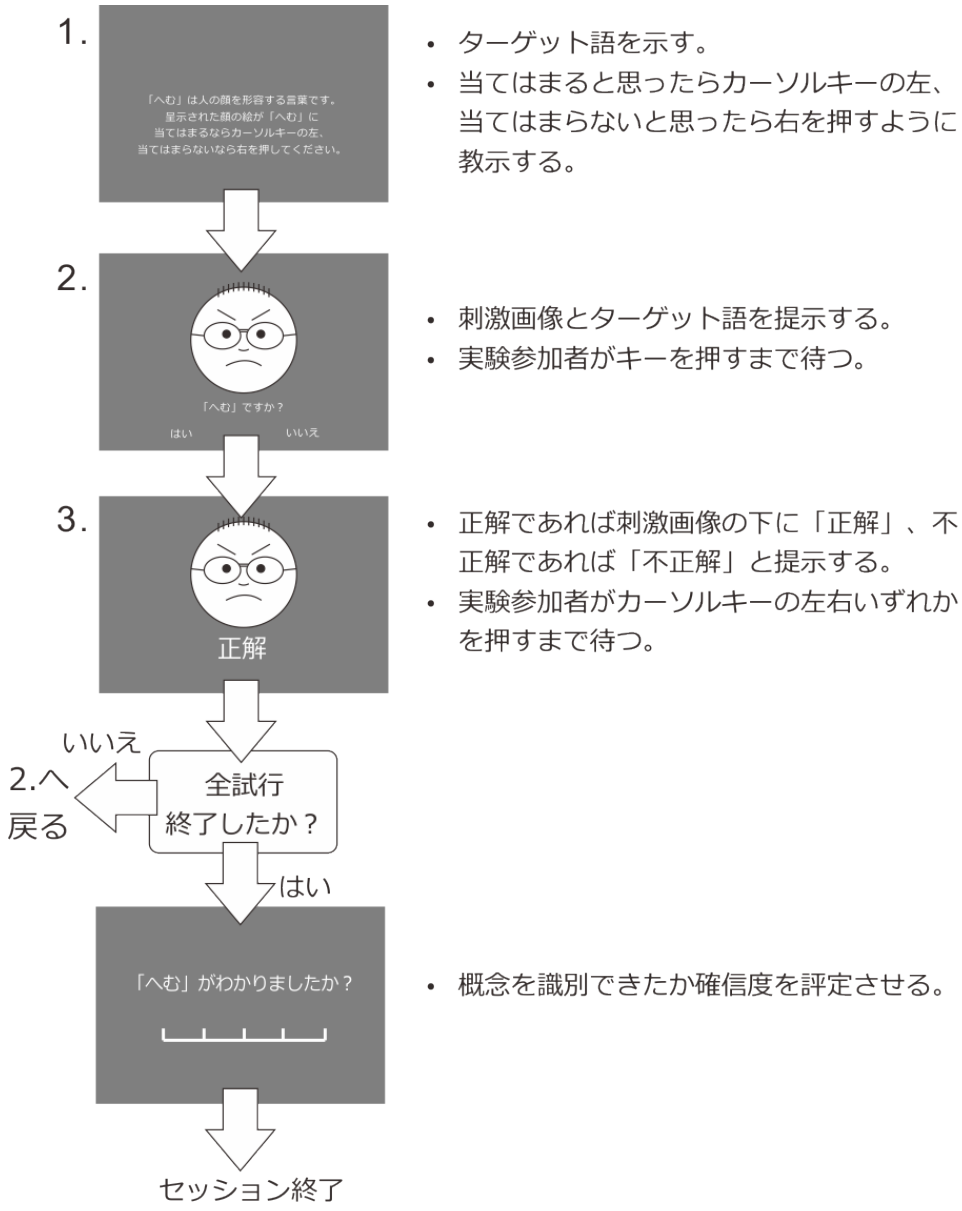


図 6.2　実験の概要。

します。着色の度合いが指定できないので、同様のことを行いたい場合は画像の上に半透明の Polygon コンポーネントを重ねる方がよいと思います。

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

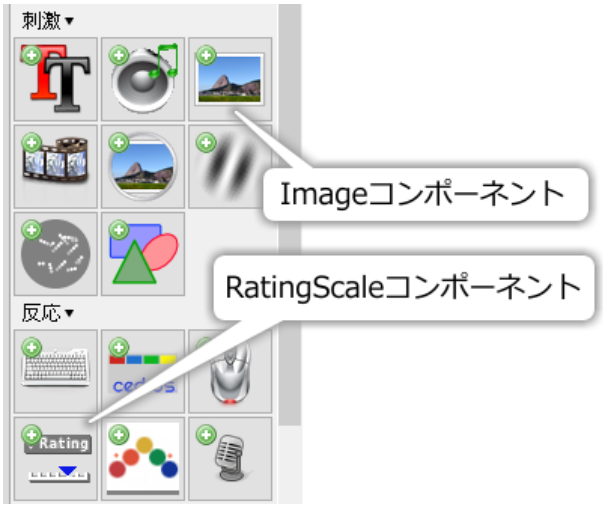


図 6.3　Image コンポーネントと RatingScale コンポーネント

- 画像ファイルをスクリーン上に提示することが出来る。
- 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出来る。

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 (**$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、カレントディレクトリである 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出来るので、..\..\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、カレントディレクトリ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出来ます。 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立ちます。しっかり理解しておきましょう。

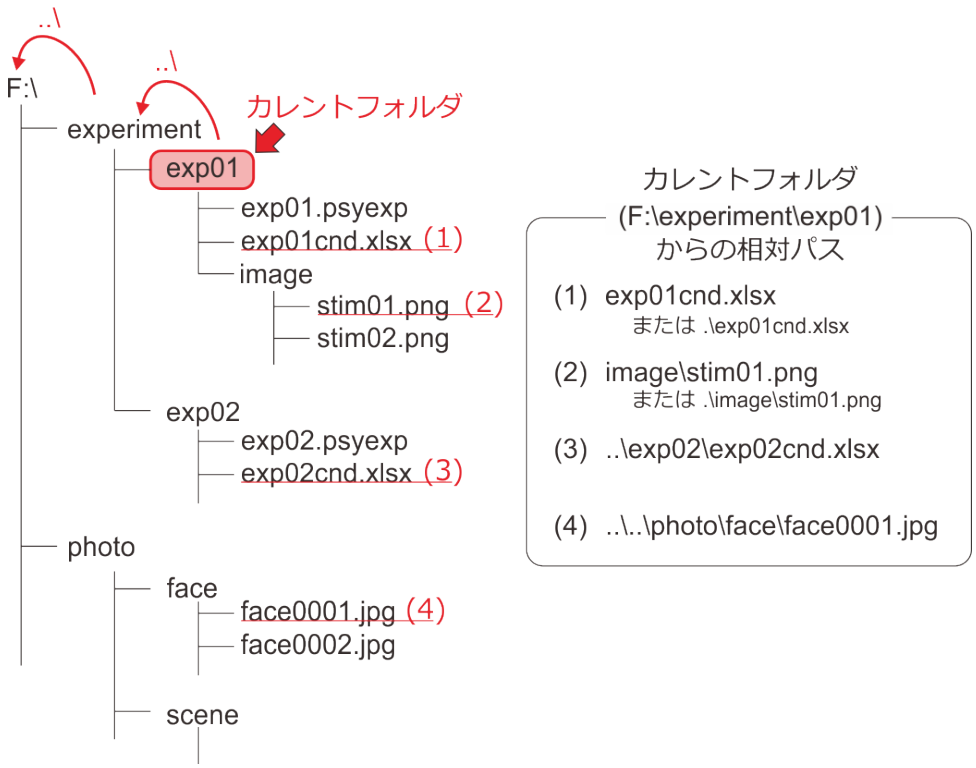


図 6.4　相対パスによるファイルの指定。

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 と書いてもパスとして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出来るのです。「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出来ませんので、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 とも書くことが出来ます。 (2) も同様に.\image\stim01.png と書くことが出来ます。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出来る。

- 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出来る。
- 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 RatingScale コンポーネント

RatingScale 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。ここまでに紹介した他のコンポーネントと違って、スクリーン描画機能と反応取得機能の両方を持っている複雑なコンポーネントです。かなり荒削りというか、開発途上な感が強いコンポーネントなので、入門テキストで取り上げるべきか迷いましたが、この種の反応記録方法を必要とする実験も多いでしょうからここで取り上げます。

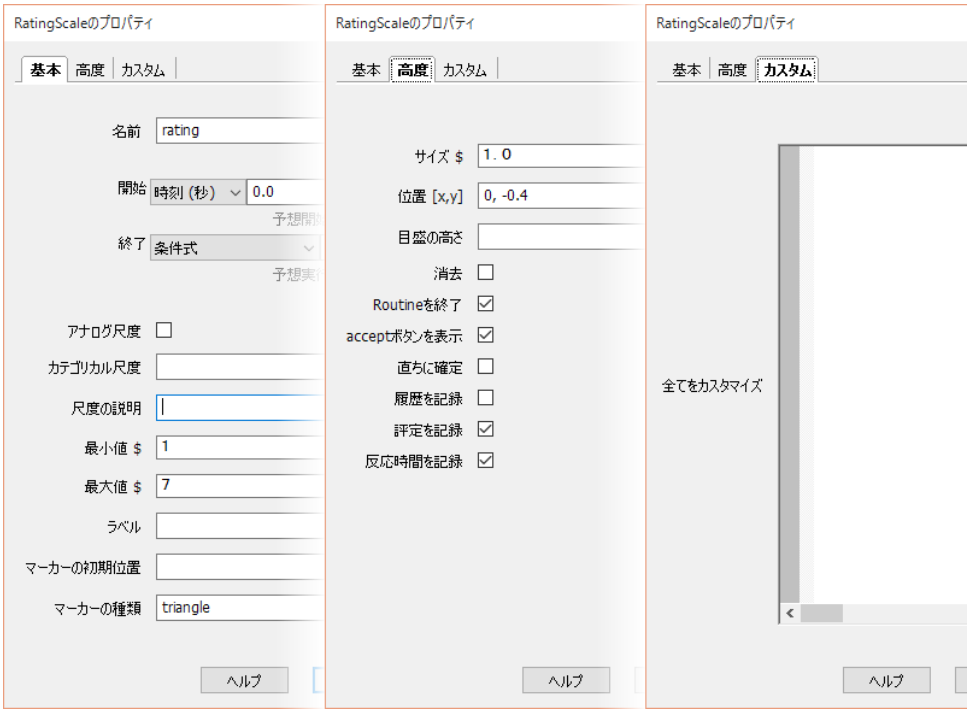


図 6.5　RatingScale コンポーネントのプロパティ設定ダイアログの内容

図 6.5 に RatingScale コンポーネントのプロパティ設定ダイアログの内容を示します。これまでに紹介してきたコンポーネントと同一のプロパティは **名前** 、 **開始** 、 **終了** のみです。これらのプロパティの意味は他のコンポーネントと同一です。続いて **アナログ尺度** と **カテゴリカル尺度** ですが、これらの項目の設定で提示される尺度が変わります。 **アナログ尺度** のチェックがオフで、 **カテゴリカル尺度** が空白であれば、N 件法の尺度が提示されます (図 6.6 一番上)。 **ラベル** に「あてはまらない, あてはまる」のようにカンマ区切りで入力すると、両端のラベルとして表示されます。 **ラベル** を空白のままにしておいた場合は **最小値 $** と **最大値 $** が表示されます。

N 件法の尺度では、参加者はマウスで尺度上の位置をクリックして回答を選択することが出来ます。選択すると尺度の下の key, click と書かれたボタンに選択した値が表示されますので、このボタンをクリックしたら反応が確定されます。反応を確定するまでは何度でも選択できます。キーボードで反応する場合は、キーボードの数値のキー (テンキーではなくキーボード上段左端から右へ 1、2、3、…と並んでいるキー) で選択します。図 6.6 一番上のように両端にラベルを付けている場合は、キーボードの数値キーを押して反応するように参加者に求めるのは厳しいと思いますが、数値キー自体は使用できます。一度いずれかの値を選択した後であれば、カーソルキーで左右に選択項目を動かして、Enter キーで確定することもできます。

図 6.6 上から二番目の例では、 **最小値 $** に負の値を指定しています。このような場合、数値キーで直接-3 などを入力することが出来ないので、数値キーによる選択は出来ません。これに対応して、尺度の下のボタンが click line に変更されています。

図 6.6 上から三番目は、カテゴリカル尺度の例です。 **カテゴリカル尺度** に選択肢をカンマ区切りで入力して作成します。N 件法の尺度と同様にキーボードからも操作が可能なのですが、一番左端が 0 キー 、左から二番目が 1 キー となってしまいますので、やはりあまり実用的ではないでしょう。

図 6.6 上から四番目は、アナログ尺度の例です。 **アナログ尺度** がチェックされていれば、アナログ (無段階) 尺度が提示されます。マウスカーソルを使って尺度上の位置をクリックして選択し、下の accept?と表示されたボタンを押して確定します。キーボードから反応する場合は、まず数値キーの 0 か 1 を押してマーカーを表示した後、カーソルキーの左右でマーカーを動かすことが出来ます。Enter キーで確定です。

図 6.6 の一番下は **マーカーの種類** の使用例です。この例では Hover という値を指定しています。Hover はカテゴリカル尺度でのみ有効で、図のように横線が表示されずに選択肢 (この例では A, B, C) のみが表示されています。選択肢にマウスカーソルを重ねると、文字の色が変化します。初期値は triangle で、上から三番目の例に表示されているような青い三角形が選択した位置に表示されます。

N 件法の尺度の場合、 **尺度の説明** に文字列を記入すると 図 6.6 一番上の例のように尺度の上にメッセージを表示できます。しかし、文字の大きさや色を自由に指定できませんので、細かい調整をしたい場合は Text コンポーネントを使った方が早いと思います。 **マーカーの初期位置** に値を指定すると、最初からいずれかの値を選択した状態にすることが出来ます。キーボードを使って選択させたい場合は便利な機能です。 図 6.6 上から三番目では **マーカーの初期位置** に「その他」を指定しています。カテゴリカル尺度の場合は、左端を 0 とした数値を入力しても構いませんし、選択肢の文字列を直接入力しても構いません。

以上で「基本」タブの解説は終わりです。続いて「高度」タブの内容を見ていきましょう。

サイズ 相対的な大きさを指定します。他のコンポーネントの **サイズ [w, h] $** のように **単位** の影響をうけませんので注意が必要です。

位置 [x, y] 位置を指定します。こちらは実験設定ダイアログの **単位** に従います。

目盛の高さ 目盛の高さを指定します。正の値は上向き、負の値は下向きを表します。0 ならば目盛は描画されません。 **サイズ** 同様、 **単位** の影響を受けません。

消去 反応が確定した尺度が画面から消去されます。

Routine **を終了** Keyboard コンポーネントの **Routine を終了** と同様に、反応が確定したら直ちにルーチンを終了します。ただし、複数の RatingScale がスクリーン上に配置されている場合などには、すべての

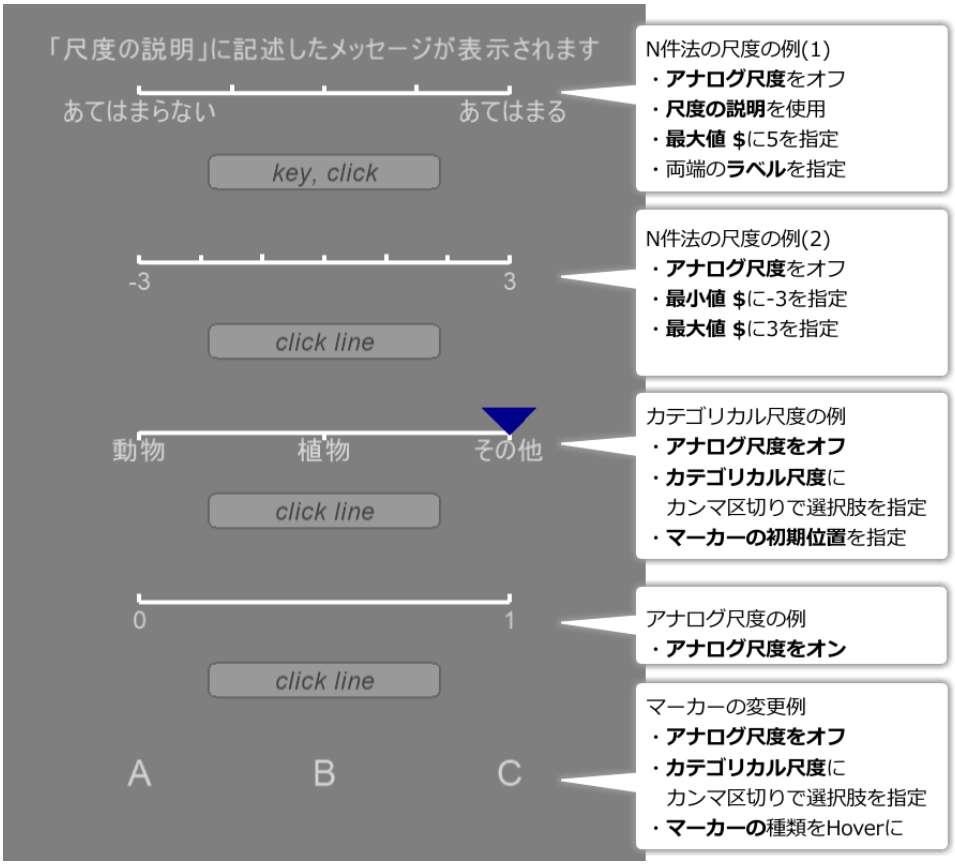


図 6.6　RatingScale コンポーネントで作成できる尺度の例

RatignScale が確定しないとルーチンが終わりません。

accept **ボタンを表示**　尺度の下の確定ボタンを表示しません。

直ちに確定　尺度を一度クリックしたら直ちに反応が確定されます。選びなおしは出来ません。

履歴を記録　反応確定までにクリックされた選択肢を反応時間付きですべて記録ファイルに保存します。

評定を記録　確定時に選択されていた値を記録ファイルに保存します。あまりこのチェックを外して使用する機会はないと思います。

反応時間を記録　最終的に反応を確定するまでに要した時間を記録ファイルに保存します。

最後に「カスタム」のタブですが、このページには **すべてをカスタマイズ** という項目しかありません。これは PsychoPy のコードを直接記述するためのもので、本書で想定しているレベルを超えていますので解説を省略します。興味がある方は *RatingScale* **コンポーネントの すべてをカスタマイズ について** *(上級)* を参考にしてください。

チェックリスト

- N 件法の尺度をスクリーンに提示して反応を記録することが出来る。
- アナログ尺度をスクリーンに提示して反応を記録することが出来る。

- カテゴリカルな尺度をスクリーンに提示して反応を記録することが出来る。

6.5 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。この章の解説では、Builder で実験を新規作成し、以下の作業を行って exp06.psyexp の名前で保存したものとします。「適当な値 (○○など) に設定する」と書かれている項目は、使用しているモニターの寸法などによっては括弧内の値ではスクリーンからはみ出たり文字が小さすぎたりするかもしれませんので、必要に応じて変更してください。

- 実験設定ダイアログ
  - 「xlsx 形式のデータを保存」をチェックする。
  - **単位** を pix にする。
  - **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。
- trial ルーチン
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - Image コンポーネントをひとつ配置して、 **名前** を faceImage にする。 **終了** を空白にする。 **サイズ [w, h] $** を [400,400] にする。
  - Text コンポーネントを 3 つ配置して、 **名前** を textYes、textNo、textQuestion にする
    - すべて **終了** を空白にし、 **文字の高さ $** を適当な値 (36 など) にする。
    - textYes がスクリーン左下に提示されるように **位置 [x, y] $** を適当な値に設定する ([-250,-280] など)。 **文字列** に「はい」と入力する。
    - textNo がスクリーン右下に提示されるように **位置 [x, y] $** を適当な値に設定する ([250,-280] など)。 **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] $** を適当な値に設定する ([0,-220] など)。
  - 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。
    - **名前** を key_choice にする。
    - **終了** を空白にする。
    - **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。
- instruction ルーチン (作成する)
  - フローの先頭に挿入する。

  - 最初から Static コンポーネントが配置されている場合は削除する。
  - Text コンポーネントをひとつ配置し、**名前** を textInst にする。**終了** を空欄にし、**文字の高さ $** に適当な値 (24 など) を入力する。
  - Keyboard コンポーネントをひとつ配置し、**終了** を空白にする。**検出するキー $** を'left','right' にし、**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。
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - Image コンポーネントをひとつ配置して、**名前** を faceImage_2 にする。**終了** を空白にする。**サイズ [w, h] $** を [400,400] にする。
  - Text コンポーネントをひとつ配置し、**名前** を textFeedback にする。**終了** を空白にし、**文字の高さ $** を適当な値 (36 など) にする。faceImage_2 の下に提示されるように **位置 [x, y] $** を適当な値に設定する ([0,-220] など)。
  - Keyboard コンポーネントをひとつ配置し、**終了** を空白にする。**検出するキー $** を'left','right' にし、**記録** を「なし」にする。
- rating ルーチン (作成する)
  - フローの末尾に挿入する。
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - RatingScale コンポーネントを配置し、**名前** を certaintyRating にする。**サイズ $** を適当な値 (0.7 など) にする。
- 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/chapter06/image.zip) 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。
- exp06cnd01.xlsx(条件ファイル)
  - 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設定する。ファイル名 (FileXXXXX.png) のみでパスは入力しないこと。

  - 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。

- exp06cnd02.xlsx(条件ファイル)

  - exp06cnd01.xlsx をコピーし、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。

以上で準備完了です。Image コンポーネントの **画像** をはじめ、いくつかの Text コンポーネントの **文字列** やループの **繰り返し条件** 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。

- Image コンポーネントの **画像** に、条件ファイルから読み込んだ刺激画像ファイル名にフォルダ名を付け足して相対パスを完成させる式を入力する。
- 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 ルーチンの Text コンポーネントの **文字列** 、および rating ルーチンの RatingScale コンポーネントの Scale Description 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。

6.6 文字列を結合して画像ファイルのパスや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。

```
$u'これは「' + expInfo['word'] + u'」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、+ 演算子のところで区切ってみましょう。

```
$ u'これは「'    +    expInfo['word']    +    u'」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、数値演算は 5+3+8 でも 8+3+5 でも同じですが (可換性)、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。最初と最後の文字列の前に u が付いているのは、この文字列が Unicode 文字列 (*Unicode* **文字列** 参照) であることを示しています。日本語のかなや漢字、全角記号などを含む文字列を式に書く場合は、この u が必要であると覚えておいてください。

RatingScale ルーチンも同様に設定してしまいましょう。rating ルーチンを開いて、certaintyRating の Scale Description に以下の式を入力してください。紙面の都合上 2 行になっていますが、入力する時は途中で改行せずに 1 行で入力してください。

```
$u'「' + expInfo['word'] + u'」がわかりましたか？
 (1:全くわからない 7:とてもよくわかった)'
```

あともうひとつ、instruction ルーチンの textInst の **文字列** が残っていますが、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。○○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。

```
「○○」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。
```

○○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、 図 6.7 上段のように **文字列** に記入して実験を実行してみても、エラーメッセージすら表示されずに終了してしまいます。エラーメッセージが表示されないので何が起こっているのか非常にわかりにくい厄介なエラーです。 図 6.7 上段の式がエラーになるのは、$が先頭に入力されていることと、文字列が途中で改行されていることが原因です。

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出来ます。それなのになぜ 図 6.7 上段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。 **色** の設定を思い出してください (第 3 章)。 **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

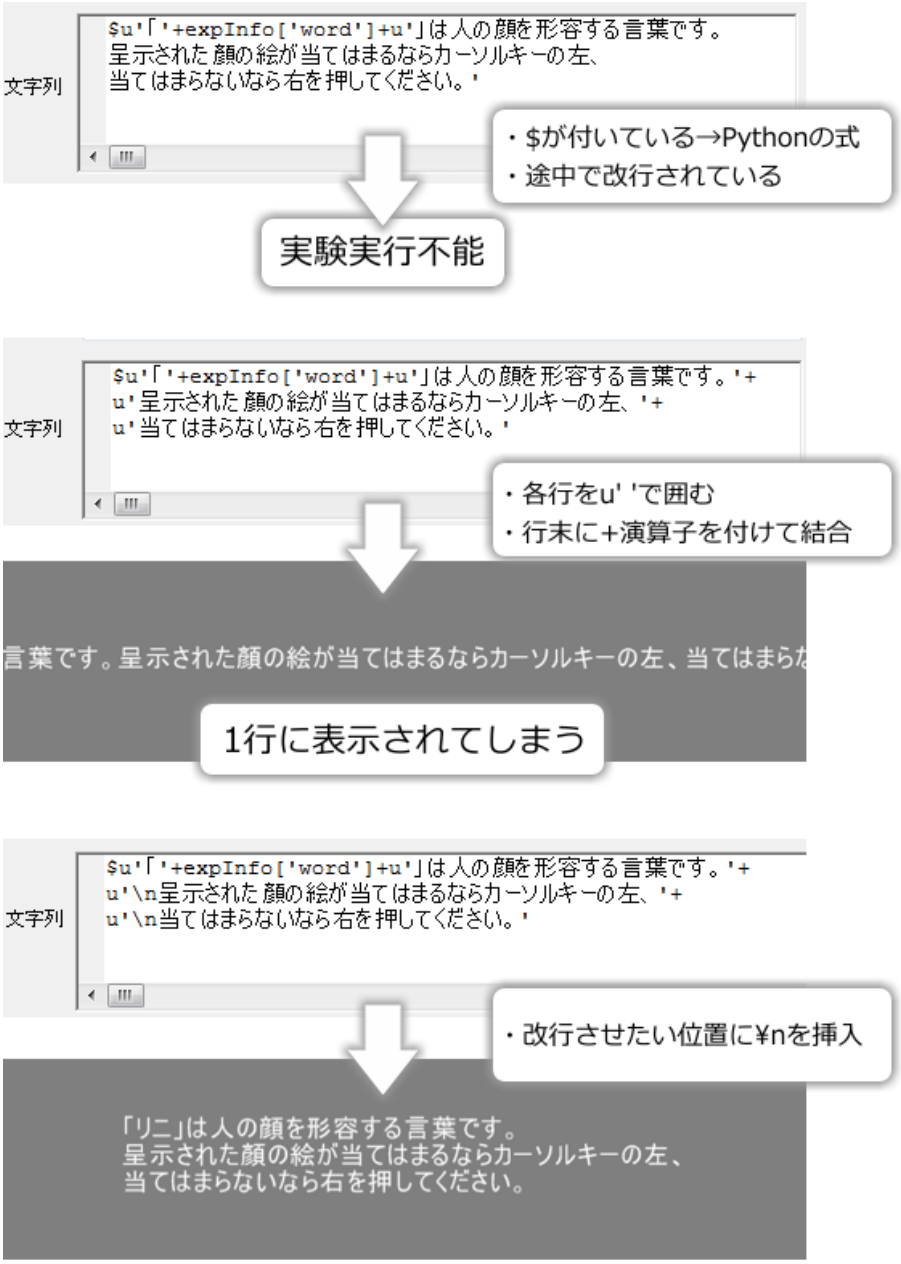


図 6.7　複数行にわたる文の提示。$を用いて Python の式を入力すると **文字列** の自動改行が無効になるので Python の式として改行を明示する必要があります。

となるのです。

これを踏まえて 図 6.7 上段に戻ります。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 (u' に対応する' が 1 行目に無い)。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。従って、この実験は実行できないのです。

ではどう変更すればいいかというと、ひとつは教示文の全てを 1 行で書いてしまうという方法です。ただ、今回の教示文は長いので、1 行で書くと後から内容を非常に確認しづらくなります。もうひとつの方法は、Python の「演算子で終わっている行はその次の行まで式が続いているとみなす」という文法規則を利用する方法です。具体的には、 図 6.7 中段のように各行の文字列部分を洩れなくクォーテーションで囲んで、最終行以外は行末に + 演算子を置きます。こうすると Python からはひとつの式が複数行に続いていると解釈されますし、+ 演算子で文字列が結合されるので問題なく実行することが出来ます。ただ、残念ながら、実行すると 図 6.7 中段のように教示文が 1 行に出力されてしまい、今回目指している出力とは異なる結果になってしまいます。$を使って **文字列** に入力されている通りに改行するという機能は、 **文字列** が Python の式と解釈される場合には無効になってしまうのです。

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。二つ方法がありますが、ここでは将来的に Python 以外の言語に触れることになった場合にも役に立つ可能性が高い方法を紹介しておきましょう。それは、文字列の中に明示的に「ここで改行せよ」という指示を入れるという方法です。PC における文書データでは、「ここで改行する」ということを示す特殊な「文字」が存在します。テキストエディタやワープロソフトを使用していて、行の末尾に矢印のような記号が表示されているのを見たことはないでしょうか (図 6.8)。この記号こそ「ここで改行する」という文字で、試しにこの文字を Del キーや Backspace キーで消してみると、前後の行がつながってしまうはずです。この文字を改行文字と呼びます。Python では、改行文字を\n と表記します。

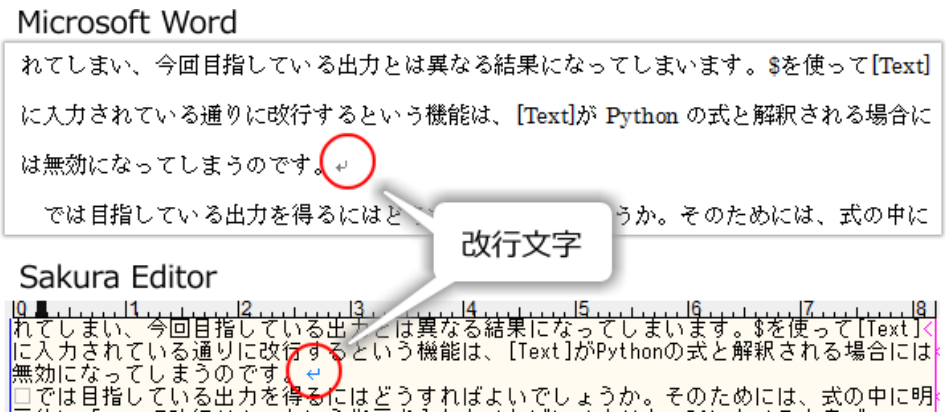


図 6.8　改行文字の表示例。行末の矢印のような記号が改行文字です。

図 6.7 下段のように、改行すべき位置にこの文字を挿入すれば、目指している出力を得ることが出来ます。 図 6.7 下段では 2 行目と 3 行目の文字列の先頭に\n を挿入していますが、+ 演算子で結合されるのですから 1 行目と 2 行目の末尾に\n を挿入しても同じ結果が得られます。末尾に挿入すると、以下のような式になります。

```
$u'「'+expInfo['word']+u'」は人の顔を形容する言葉です。\n'+
u'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'+
u'当てはまらないなら右を押してください。'
```

図 6.7 下段の式かこの式のいずれかを instruction ルーチンの textInst の **文字列** に入力してください。これで教示文へのターゲット語の埋め込みが完成しました。なお、複数行にまたがる教示文を入力する「もうひとつの方法」について知りたい方は *Python* **で複数行にまたがる文字列を表記する方法** を参照してください。

最後に実験情報ダイアログの condition へ入力された値から条件ファイル名を得る問題が残っていますが、ここまでの解説を理解していればもうこれ以上の解説は不要でしょう。trials ループの設定ダイアログを開き、**繰り返し条件** 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 を指定することが出来ます。

```
$'exp06cnd' + expInfo['conditionFile'] + '.xlsx'
```

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出来る。
- 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出来る。
- 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出来る。

6.7 Python における変数への代入、比較演算子、論理演算子、条件分岐を学ぼう

ここまでの作業でとりあえず実験を実行できるところまでたどり着きましたが、最後の難題である「判断の正誤をフィードバックする」が残っています。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」と提示したいのですが、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。実は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 6.9 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。

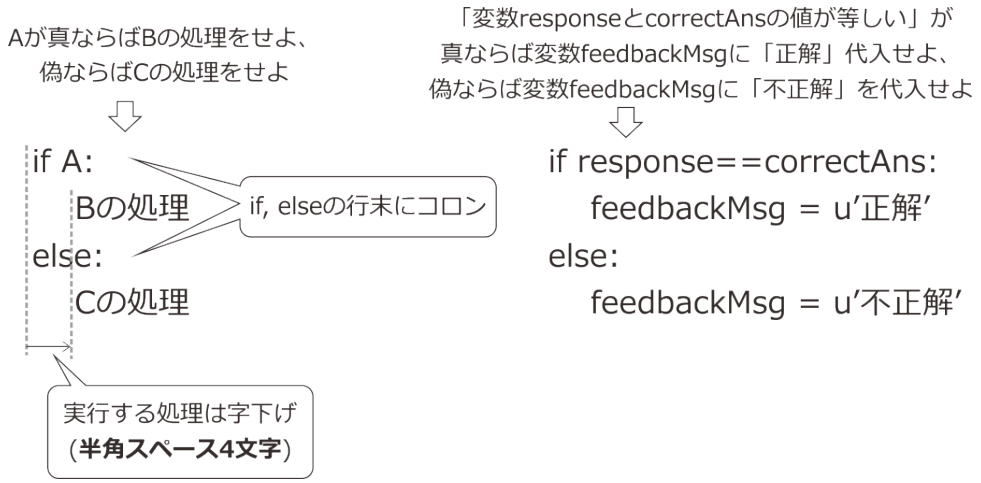


図 6.9　条件分岐 (if 文) の書式。右は実際のコードの例です。

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 `u'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 `u'正解'` と設定する
↓
・変数 `feedbackMsg` に `u'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 6.9 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 6.9 左側と見比べて、 図 6.9 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 *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 **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出来ます。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出来ます。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 6.9 の B と C に対応するコードについて解説しましょう。

```
feedbackMsg = u'正解'
feedbackMsg = u'不正解'
```

これらの式では代入演算子 (=) が使用されています。代入演算子は X = Y という形で使用し、右辺の式を評価した結果を左辺の変数に代入します。代入の方向は右から左で固定されていますので、「7 = x と書いて変数 x に 7 を代入する」ことは出来ません。左辺は変数でなければいけませんので、x+7 = 3 といった書き方も出来ません。あと、プログラミングに慣れていない人にはちょっと奇妙に思われるかもしれないけれどもとても便利な用法を紹介しておきましょう。

```
x = x+7
```

この式では、変数 x に格納された値に 7 を加えて、その結果をまた変数 x に代入しています。例えば上記の式を実行する前に x に 4 が格納されていたのであれば、この式を評価した結果 x には 11 が格納されています。この式が奇妙に思われる方は、恐らく数学において=記号が等号 (右辺と左辺の値が等しい) として使われることが頭にあるのではないかと思います。=を等号として解釈するとこの式は確かに奇妙です。しかし、すでに述べたように Python における等号は==と書きますので、この式の=を等号と解釈してはいけません。プログラミングに慣れていないうちはよく=と==を間違えますので気を付けてください。

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ますので、Python では二項算術演算子 (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 6.10 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ありがたくないのですが、図 6.10 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。

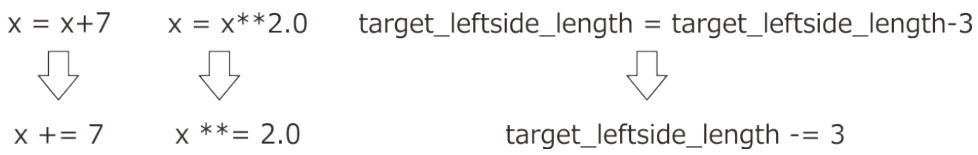


図 6.10　二項算術演算子と代入演算子の組み合わせ。変数名が非常に長い時に有効です。

さて、ずいぶん解説が長くなってしまいましたが、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と、この節で解説した if 文を Builder でどのように入力するかの二点です。次節でこれらの問題を取り上げます。

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出来る。
- 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出来る。
- 文字列の一致・不一致に応じて処理を分岐させることが出来る。
- 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出来る。
- 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出来る。
- 変数に値を代入する式を書くことが出来る。
- x+=7、x**=2 といった二項算術演算子と代入演算子を組み合わせた式の働きを説明することが出来る。

6.8 オブジェクトについて学んで Code コンポーネントを使っ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。前節で解説した Python の if 文を Builder の中で使用するためには、Code コンポーネント (図 6.11) を使用します。このコンポーネントのプロパティ設定ダイアログは今まで紹介してきたコンポーネントのそれと大きく異なっており、他のコンポーネントと共通のプロパティが **名前** しかありません。**名前** の下には **実験開始時** 、 **Routine 開始時** 、 **フレーム毎** 、 **Routine 終了時** 、 **実験終了時** というプロパティがあります。入力欄が非常に大きいので、それぞれにタブが用意されていてページを切り替えて入力します。

実験開始時 に Python のコードを記入すると、実験開始時にそのコードが実行されます。同様に、 **Routine 開始時** に入力したコードは、その Code コンポーネントが配置されたルーチンが開始される時に実行されます。 **Routine 終了時** や **実験終了時** も同様です。複数のルーチンに Code コンポーネントが置かれていた場合、 **実験開始時** に入力されたコードは psyexp ファイルに登録されている順番 (Flow ペインで「Routine を挿入」した時にメニューに並ぶ順) に実行されますが、そのことに依存したコードを書くべきではありません。同一ルーチンに複数の Code コンポーネントがある場合、 **Routine 開始時** 、 **フレーム毎** 、 **Routine 終了時** は Routine ペインに並んでいる順番 (上から下) に実行されます。何か意図があって複数の Code コンポーネントを同一ルーチンに並べるのであれば構わないのですが、原則としてひとつのルーチンには Code コンポーネントは 1 個にしておくのがよいでしょう。

では作業に入りましょう。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。ルーチンの終了時に、押されたキーに応じて変数 feedbackMsg に「正解」または「不正解」という文字列を代入しましょう。 **Routine 終了時** に以下のコードを入力してください。

```
if key_choice.keys == correctAns:
    feedbackMsg = u'正解'
```

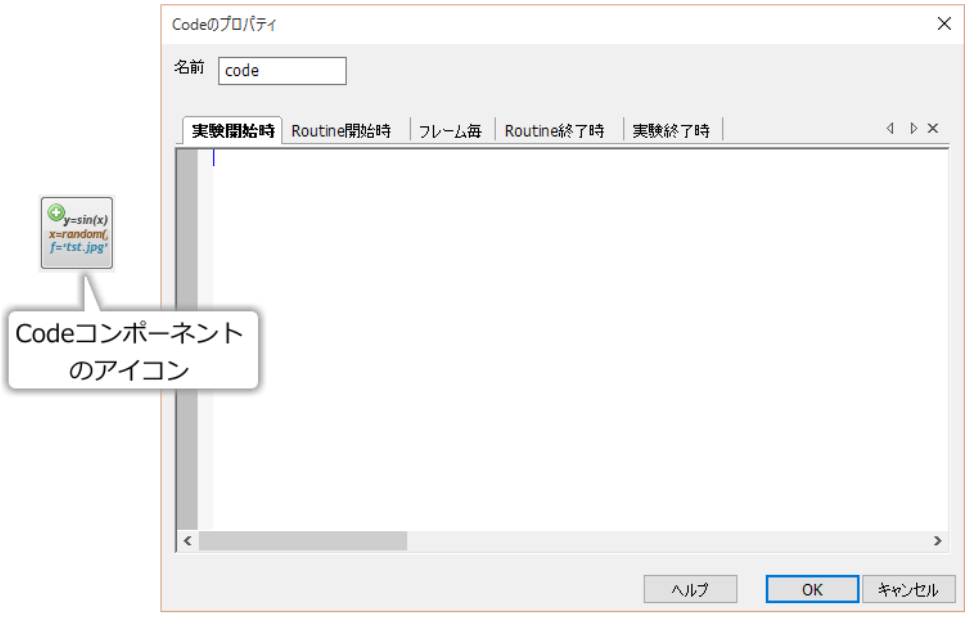


図 6.11　Code コンポーネントのアイコンとプロパティ設定ダイアログ。

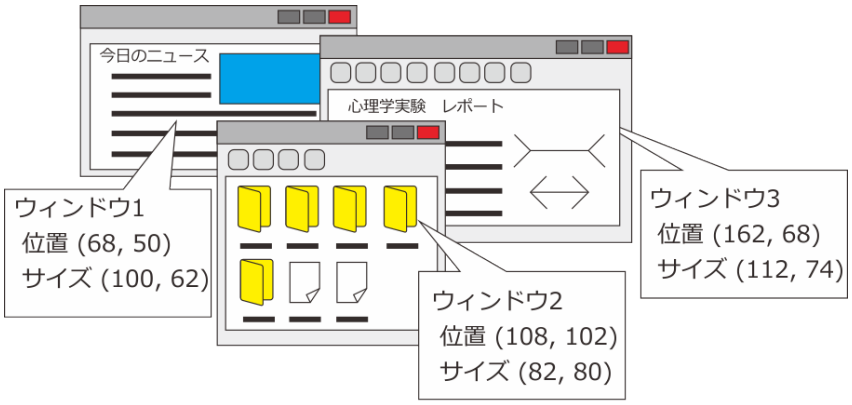
```
else:
    feedbackMsg = u'不正解'
```

ほぼ前節で解説した通りですが、1 行目の押されたキーに対応する部分が異なっています。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ので、詳しく解説しましょう。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頑張って付いてきてください。

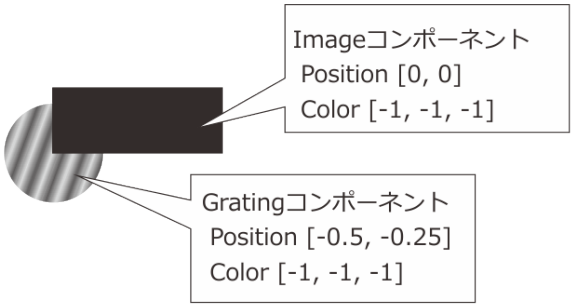
1 行目で correctAns と比較されている key_choice.keys に注目します。変数名にピリオドを含むことは出来ませんので、これは key_choice.keys という名前の変数ではなく、key_choice という変数に.keys というものが添えられた式のはずです。では、key_choice という変数には何が格納されているのでしょうか。Builder のキーボード反応計測用オブジェクト (psychopy.event.BuilderKeyResponse：以下 BuilderKeyResponse) というのがその答えです。オブジェクトとは、コンピュータ上で扱うさまざまな対象、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 6.12 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出来ました。 **色** プロパティで色を指定することも出来ましたが、Image コンポーネントと Grating コンポーネントでは同じ値を **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 6.12)。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。



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 6.12　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。

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 6.13 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出来ますので、( ) 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。

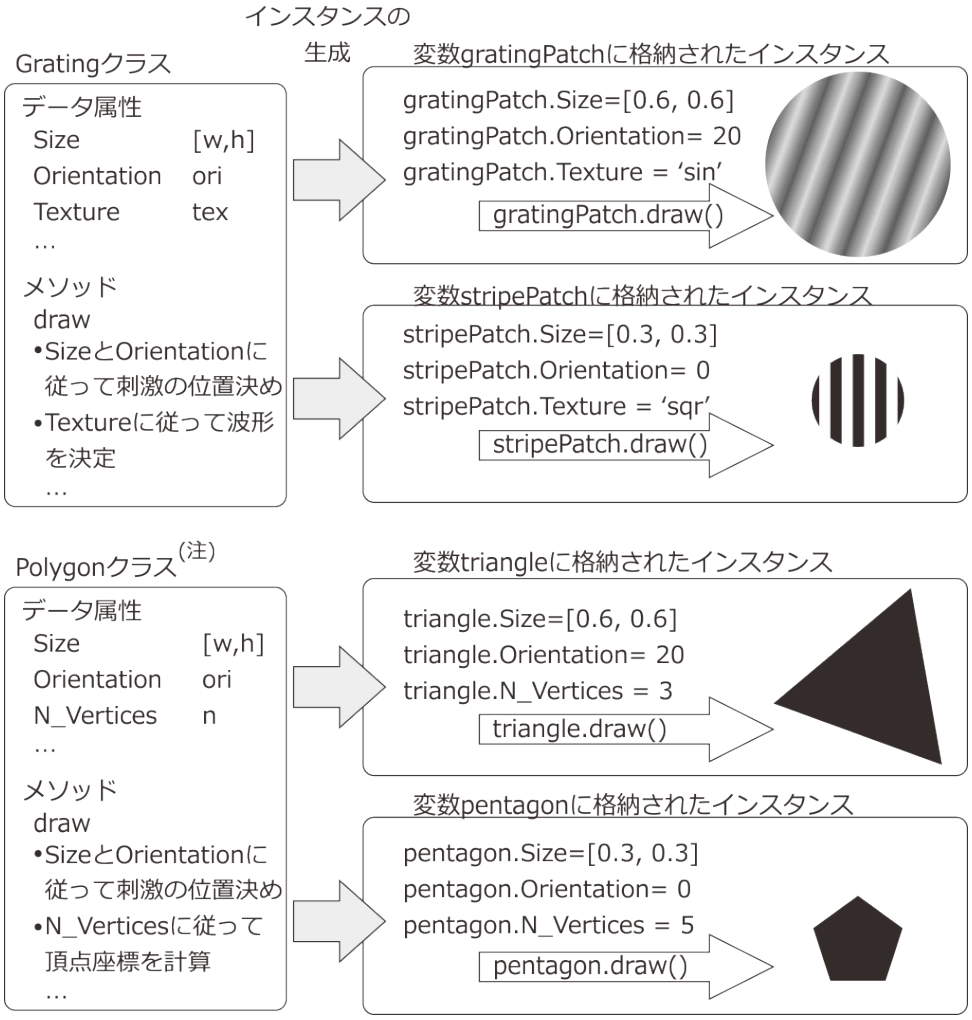


図 6.13　データ属性とメソッド、クラスとインスタンス。データ属性の値は個々のインスタンスで異なります。メソッドを呼ぶと自分のクラスで定義されているメソッドが呼び出されるので、異なるクラス間で同名のメソッドがあっても混乱しません。

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( ) 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 *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 Code コンポーネントに入力したコードの 1 行目にあった 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 Code コンポーネントに入力した式の解説はおしまいです。そして、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。

実験が終わったら、xlsx 記録ファイルを開いてみましょう (図 6.14)。xlsx ファイルには trials と ratings という 2 枚のシートがあるはずです。それぞれ、trials ループに含まれるルーチンからの出力と、ratings ループに含まれるルーチンからの出力に対応しています。trials のシートは前章までの内容から特に新しいことはありませんので、ratings のシートを確認しましょう。ここには RatingScale コンポーネントの出力が記録されいますが、RatingScale コンポーネントも Keyboard コンポーネントと同様に、コンポーネントの **名前** に指定した名前に.response_mean とついている列に反応の平均値、.response_raw とついている列から右方向へ各試行の反応が出力されています。さらにその右には反応時間の平均値と各試行の反応時間が出力されています。なお、RatingScale コンポーネントのプロパティ設定ダイアログでカテゴリカルな尺度を使用するように設定していた場合は、反応の平均値が計算できませんので平均値の列は省略されます。

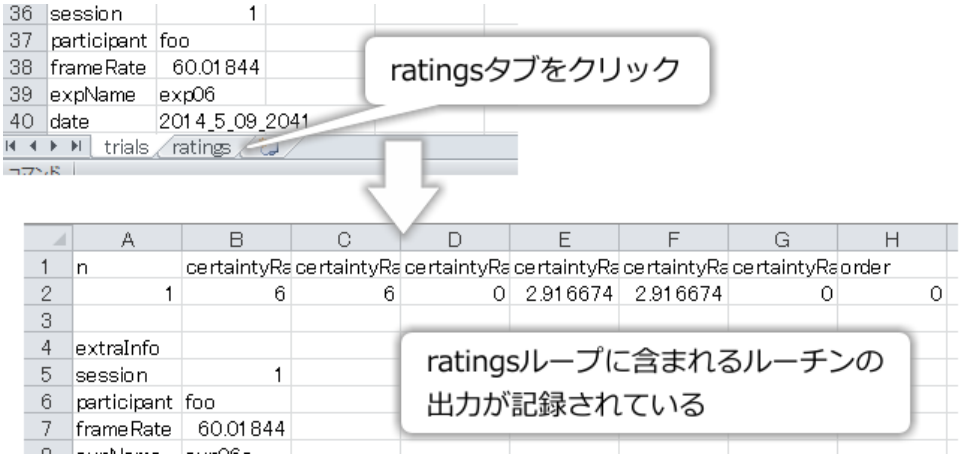


図 6.14　xlsx 記録ファイルの例。結果がループ毎にシートに分かれて出力されています。ratings シートを選択すると、ratings ループに含まれているルーチンからの出力を確認できます。

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。

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。
- 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出来る。

6.9 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。

6.10 この章のトピックス

6.10.1 RatingScale コンポーネントの すべてをカスタマイズ について (上級)

このトピックは Builder を使わずに直接 Python のスクリプトを書けるレベルの方を想定して書きますのでそのつもりでお読みください。

RatingScale コンポーネントの **すべてをカスタマイズ** は、RatingScale オブジェクトのコンストラクタに渡す追加の引数を記述するためのものです。このプロパティを指定せずに RatingScale オブジェクトを作成すると、Python のコードに変換されたときに以下のようにコンストラクタが呼び出されます。

```
rating = visual.RatingScale(win=win, name='rating',
    marker=u'triangle', size=1.0, pos=[0.0, -0.4],
    low=1, high=7, labels=[u''], scale=u'')
```

ここで、他のプロパティ値はすべてそのままで customize_everything に以下のように書きこんでみます。

```
foo=bar, baz='qux'
```

コードをコンパイルしてみると、コンストラクタの呼び出しが以下のように変わります。

```
rating = visual.RatingScale(win=win, name='rating', foo=bar, baz='qux')
```

つまり、 **名前** 以外のプロパティ設定ダイアログの項目はすべて無視されて、 **すべてをカスタマイズ** に入力した文字列がそのまま win と name 以外のコンストラクタの引数になるわけです。もちろん psy-

chopy.visual.RatingScale には foo や baz といった引数は無いので、このコードはエラーとなります。コンストラクタの引数を理解して自分で正しい呼び出しコードが書ける人向けの機能だと言えます。

6.10.2 Unicode 文字列

*$*を含む文字列を提示する でも触れたように、コンピュータでは文字に数値を割り振ることによって文字を管理しています。そしてやはり *$*を含む文字列を提示する で触れたように、地域によって使用される文字コードが異なるため、ある文字コードで書かれた文書を別の文字コードを使用するコンピュータで閲覧すると本来とは異なる文字で表示されてしまうという問題が生じます。この問題を解消するために、世界各地のコンピュータで使用されている文字を網羅する共通の文字コードを制定しようとしたのが Unicode です。Unicode を利用することによって、従来は異なる文字コードで表現されていた複数の言語を同一の文書に混在させることが簡単になりました。英語圏で開発されている PsychoPy で問題なく日本語の文章を刺激として提示できるのは Unicode のおかげです。なお、Unicode には UTF-8、UTF-16 など複数の符号化方式がありますが、Builder 内部で Unicode 文字列を使う限りはこれらの違いを意識する必要はありません。

Unicode のおかげで文字コードにまつわる問題が大幅に改善されましたが、Unicode も万能ではありません。例えば *$*を含む文字列を提示する で紹介した \ と ¥ の問題は文字コードと文字のグリフ (見た目) をどう対応させるかという問題なので、Unicode でも解決できません。日本語のユーザーにとって身近なところでは波ダッシュ (〜) をどう表示するかといった問題や、漢字の部首の「しんにょう」を一点で表示するか、二点で表示するかといった問題などがあります。

6.10.3 Python で複数行にまたがる文字列を表記する方法

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出来ます。

```
message1 = '''必要に応じて休憩を取ってください。

準備が出来たらスペースキーを押して実験を再開してください。'''

message2 = """これで実験は終了です。

ご協力ありがとうございました。"""
```

といった感じです。本文中の問題は、以下のように書くことが出来ます。先頭の 「 は通常のシングルクォーテーションを使っていて、ターゲット語の後の 」以降の部分にシングルクォーテーションを 3 個連ねる表記を使用しています。なお、先頭の$を省略してますので、Builder で試すときには$を忘れずにつけてください。

```
u'「'+expInfo['word']+u'''」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。'''
```

6.10.4 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出来ます。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 6.15 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、Python 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出来ます。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 6.15 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。

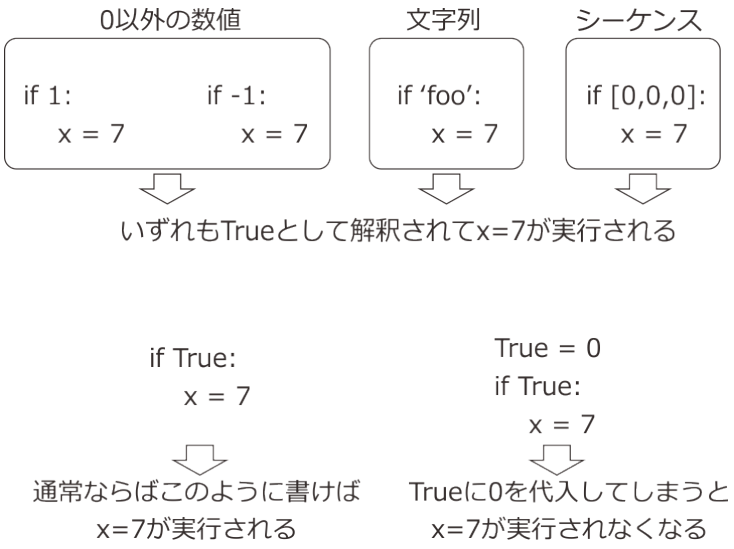


図 6.15　True と False に関する注意事項。

6.10.5 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出来ます。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 0x8a18 と 0x5fc3）u’ 記憶’ < u’ 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。

6.10.6 Builder のコンポーネントと PsychoPy のクラスの対応

図 6.16 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出来るので、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。

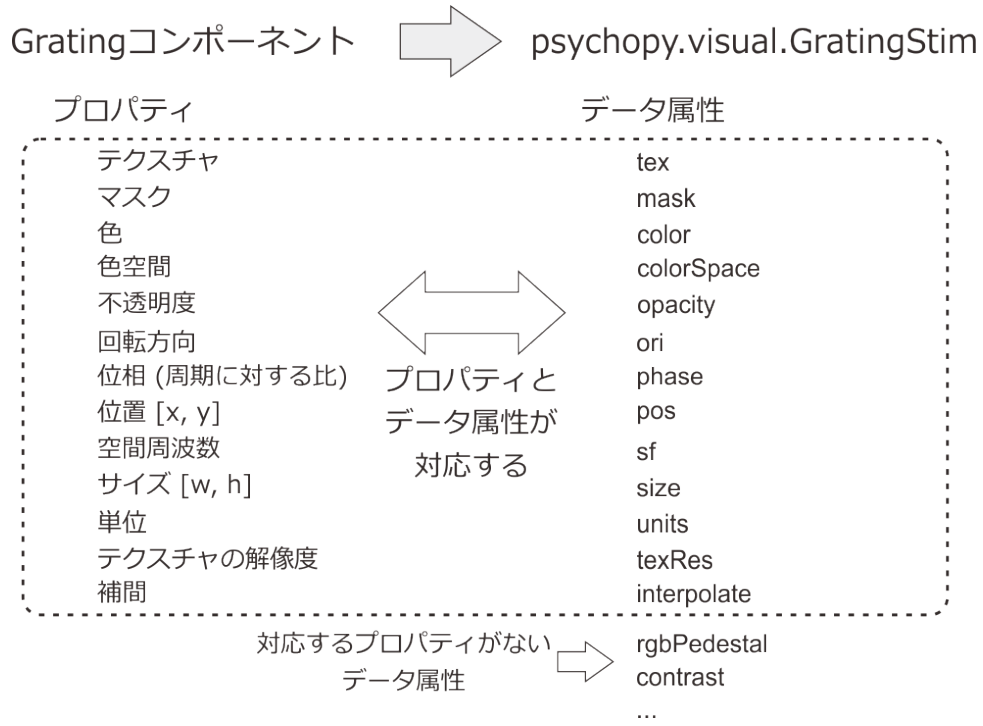


図 6.16　Builder の Grating コンポーネントのプロパティと、それに対応する PsychoPy のクラスのデータ属性。

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出来ません。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。

図 6.17 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、 **頂点数** 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、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出来ました。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。

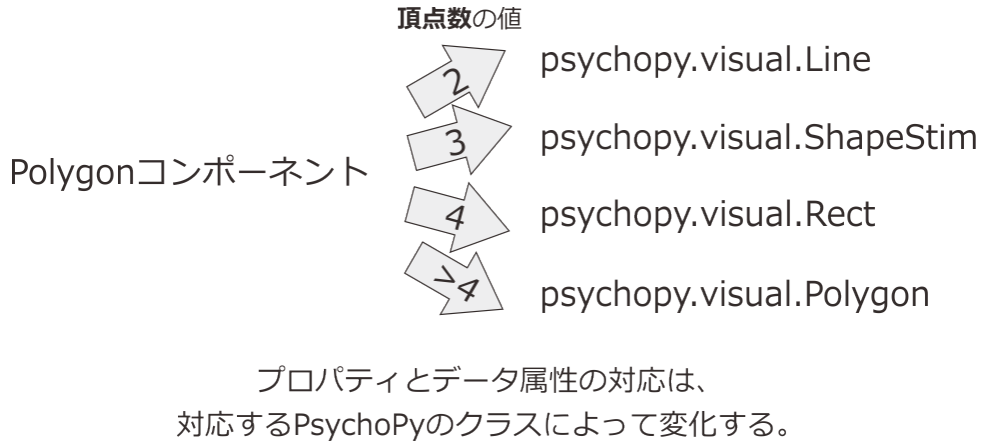


図 6.17　Builder の Polygon コンポーネントに対応する PsychoPy のクラス。

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出来るでしょうから、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 200pix、矢羽の長さは 50pix 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 200pix 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出来ます。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 200pix 離れた位置に提示されるものとします。

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。

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。同一ルーチンに Keyboard コンポーネントを 2 つ配置して、一方は **Routine を終了** をチェックしてもう一方

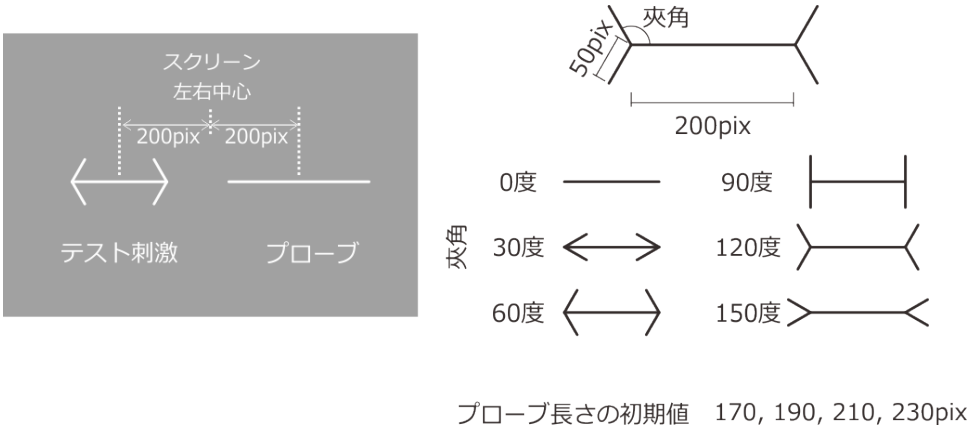


図 7.1　実験に用いる刺激。テスト刺激の水平線 (主線) の長さを 200pix、矢羽の長さを 50pix で固定し、夾角を 0 度から 150 度まで変化させます。参加者は主線の長さとプローブの長さが等しく見えるようにプローブの長さを調整します。試行開始時のプローブの長さは 170、190、210、230pix のいずれかです。

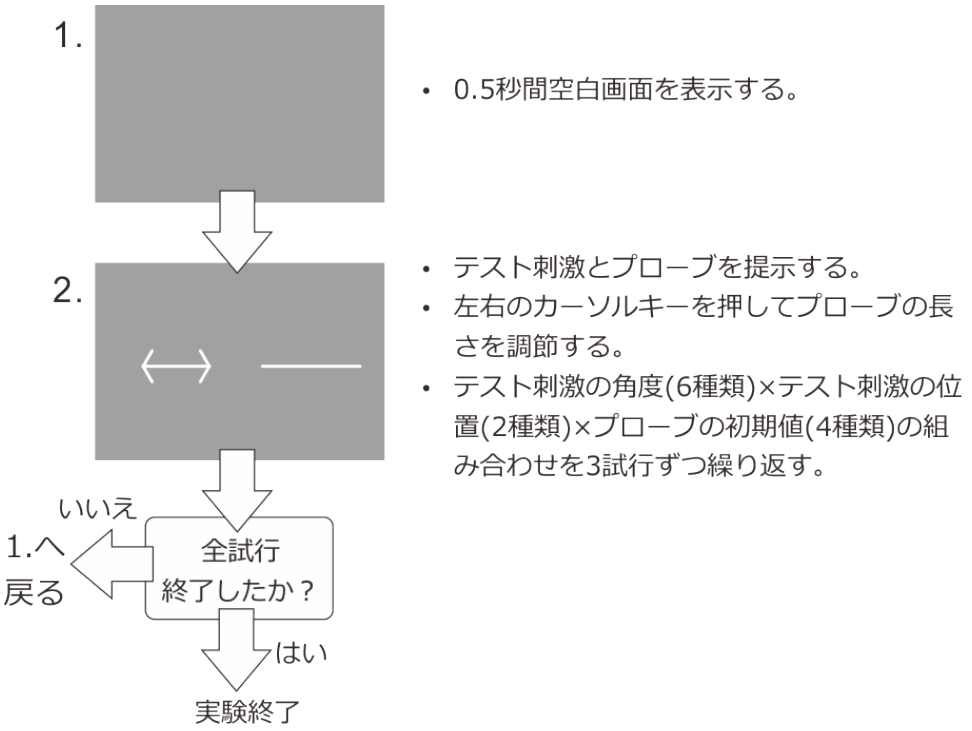


図 7.2　実験の流れ。

は **Routine を終了** をチェックしないという方法で、特定のキーが押されたときだけルーチンを終了させることも出来なくはないのですが、第 6 章で学んだ if 文を使う方が柔軟な処理を実現できますので、この章では if 文を使った方法を解説します。

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。

- 実験設定ダイアログ
  - 「xlsx 形式のデータを保存」をチェックする。

  - **単位** Units を pix にする。

- trial ルーチン
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - Polygon コンポーネントを 6 つ配置し、**名前** を probe, testline, arrowTR, arrowBR, arrorTL, arrorBL とする。
    - すべての **開始** を「時刻 (秒)」の 0.5 とし、 **終了** を空白にする。すべての **頂点数** を 2 にする。
    - testline の **サイズ [w, h] $** を [200,1] にする。arrowTR、arrowBR、arrorTL、arrorBL の **サイズ [w, h] $** を [50,1] にする。
    - testline の **位置 [x, y] $** を [testPos, 0] にして、「繰り返し毎に更新」に設定する。
    - probe の **位置 [x, y] $** を [-testPos, 0] にして、「繰り返し毎に更新」に設定する。
    - probe の **サイズ [w, h] $** を [probeLen,1] にして、「フレーム毎に更新する」を設定する。

  –Keyboard **コンポーネントを** 1 **つ配置して、 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。
    - **記録** を「なし」にする。
  - Code コンポーネントを 1 つ配置する。

- 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。
  - **繰り返し条件** に exp07cnd.xlsx と入力する。

- exp07cnd.xlsx(条件ファイル)
  - testPos、angle、probeLen 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(-200、200)、6 種類の angle(0、30、60、90、120、150)、4 種類の probeLen(170、190、210、230) 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 (パラメータ名の列を除く)。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。trial ルーチンに 6 個の Polygon ルーチンを配置しましたが、その内の 5 個 (testline、arrowTR、arrowBR、arrorTL、arrorBL) を使ってミューラー・リヤー図形を描きます。これらの位置を指定するのはちょっと複雑ですので次節でていねいに見ていきましょう。

7.2 7.2 Polygon コンポーネントで線分を描画しよう

Polygon コンポーネントで **頂点数** に 2 を指定することで、線分を描画することが出来ます。多くのプログラミング言語でスクリーン上に刺激を描くライブラリの多くは線分の両端の座標を指定するのですが、Builder では **位置 [x, y] $** で図形の中央の座標を、 **サイズ [w, h] $** で大きさを指定することになっているため、他のプログラミング言語に慣れている人ほど戸惑うかもしれません。

Polygon コンポーネントで描く線分の長さは、 **サイズ [w, h] $** の幅の値 (1 番目の値) で指定します。2 番目の値は無視されますので、適当な値 (例えば 0) を入れておきましょう。線分の太さは **輪郭線の幅 $** で指定します。線分の色は **輪郭線の色** で指定します。 **塗りつぶしの色** は無視されます。

問題は **位置 [x, y] $** です。多角形を描画する時と同様に中心の座標を指定しないといけませんが、今回は「矢羽用の線分の端点を主線の端点と一致させる」必要がありますので、端点が意図した位置にくるように線分の中心位置を計算して **位置 [x, y] $** に指定しないといけません (図 7.3 上)。線分の中点の座標を (0, 0)、線分の長さを L、回転角度を q とすると、線分の端点の座標は ±(L/2 × cos(q), L/2 × sin(q)) です。± が付いているのは線分には端点が 2 つあるからです。線分の中点を (a, b) の位置へ移動させると、端点の座標は (a, b)±(L/2 × cos(q), L/2 × sin(q)) です。今、端点の一方である (a + L/2 × cos(q), b + L/2 × sin(q)) が (x, y) に一致するように線分の中点の座標を決めたいとしましょう。その時、(a, b) は (a + L/2 × cos(q), b + L/2 × sin(q)) = (x, y) を満たしますので、これを解いて a = x - L/2 × cos(q)、b = y - L/2 × sin(q) を得ます。本来ならばこれで線分の中点をどこへ置けばよいかの計算は終わりなのですが、Builder では正の回転方向が時計回りであり、通常の三角関数の計算の時と回転方向が逆であることを考慮しないといけません。対策は簡単で、回転時の Y 座標の符号を反転するだけです (図 7.3 下)。従って、a = x- L/2 × cos(q)、b = y + L/2 × sin(q) が求める答えです。

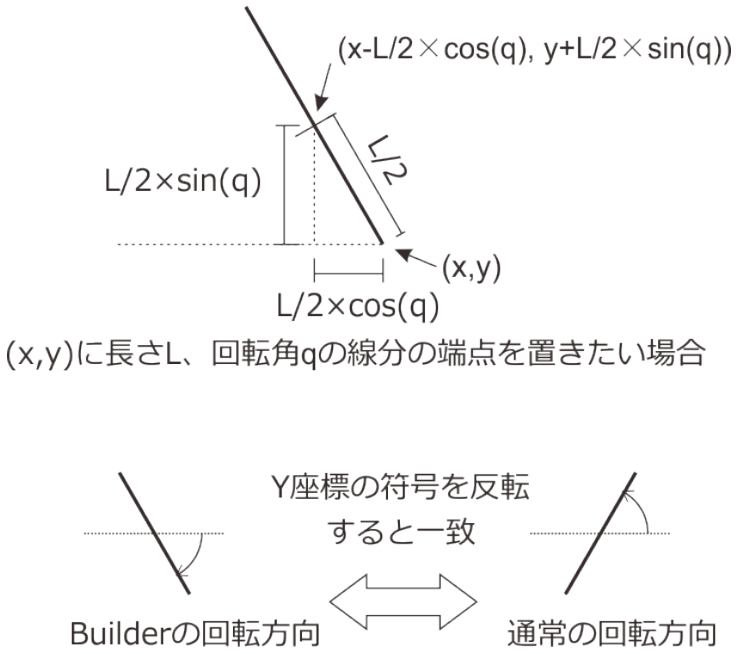


図 7.3　Polygon コンポーネントで斜めの線分を描画する時の位置の計算。Builder の回転方向と通常の三角関数における回転方向が逆なので、Y 座標の符号を反転させる必要があります。

同様の要領で、ミューラー・リヤー図形を描けるように 4 本の線分の中心位置を求めたのが 図 7.4 です。4 本すべての線分の回転角が q で統一できるように符号を決めていますので、確認してください。

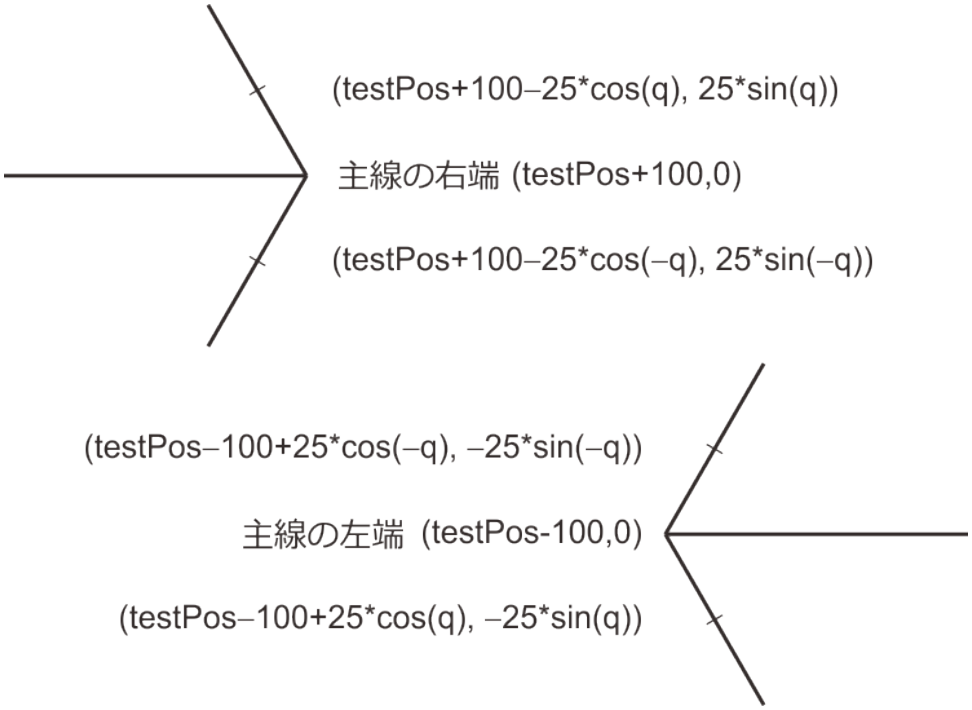


図 7.4　ミューラー・リヤー錯視の矢羽用の Polygon コンポーネントの座標。テスト刺激の位置 (X 座標) を testPos で、ミューラー・リヤー図形の夾角を q で表しています。

それでは exp07a.psyexp を開いてこれらの式を入力していきましょう。一点だけ注意しないといけないのは、図 7.4 の式の回転角 q の単位がラジアンである点です。条件ファイルに入力した夾角 angle は単位が度ですので、q の部分を deg2rad(angle) に書き換える必要があります。この点に注意して、以下のように作業してください。

- arrowTR
  - **回転角度 $** に angle と入力し、「繰り返し毎に更新」を設定する。
  - **位置 [x, y] $** に [stimPos + 100 -25 * cos(deg2rad(angle)), 25 * sin(deg2rad(angle))] と入力し、「繰り返し毎に更新」を設定する。
- arrowBR
  - **回転角度 $** に-angle と入力し、「繰り返し毎に更新」を設定する。
  - **位置 [x, y] $** に [stimPos + 100 -25 * cos(deg2rad(-angle)), 25 * sin(deg2rad(-angle))] と入力し、「繰り返し毎に更新」を設定する。
- arrorTL
  - **回転角度 $** に-angle と入力し、「繰り返し毎に更新」を設定する。

  - **位置 [x, y] $** に [stimPos - 100 + 25 * cos(deg2rad(-angle)), -25 * sin(deg2rad(-angle))] と入力し、「繰り返し毎に更新」を設定する。

- arrorBL

  - **回転角度 $** に angle と入力し、「繰り返し毎に更新」を設定する。

  - **位置 [x, y] $** に [stimPos - 100 + 25 * cos(deg2rad(angle)), -25 * sin(deg2rad(angle))] と入力し、「繰り返し毎に更新」を設定する。

以上で条件ファイルのパラメータを読み込んでミューラー・リヤー図形を描く準備が出来ました。次はいよいよ Code コンポーネントを使ってプローブの長さの変更に挑戦します。

チェックリスト

- Polygon コンポーネントを使って線分を描画することが出来る。
- Polygon コンポーネントを使った線分の長さ、角度が決まっている時に、その端点が指定された座標に一致するように **位置 [x, y] $** を指定することが出来る。

7.3 7.3 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、第 6 章よりも処理が複雑です。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

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出来ます。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出来ます。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 5 を引く
elif key == 'right':
    プローブの長さに 5 を足す
elif key == 'space':
    ルーチンを終了する処理
```

else はどこへいった？と思われるかもしれませんが、else に対応する処理が何もない場合は else を省略することが出来ます。今回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は何もする必要がないので else を省略できます。

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, 1] 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。5 を加えるには probeLen += 5、5 を引くには probeLen -= 5 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 -= 5
elif key == 'right':
    probeLen += 5
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、第 6 章の 表 6.3 をよく読みなおしてほしいのですが、正確に言うと keys 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。

ではどのよ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。「それらの名前」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 **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 を利用する場合、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等しいか比較しても意味がありません。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 -= 5
elif 'right' in theseKeys:
    probeLen += 5
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出来るようになりました。作業内容を exp07a.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。次節では、Coder コンポーネントを用いて実験記録ファイルに出力する変数を追加する方法を解説します。

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。

7.4 7.4 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出来ます。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出来ませ

ん。ですから Text コンポーネントの **文字列** に$u' 第' + (trials.thisN+1) + u' 試行' と書くと当然エラーになります。ここで第 5 章において実験情報ダイアログからターゲットの大きさや偏心度の値を得たときの処理を思い出してください。あの時は実験情報ダイアログの値は文字列で、刺激のパラメータとして利用する時には数値に変換しないといけないのでした。今回はこの逆で、数値である trials.thisN+1 を文字列にしないといけません。第 5 章で紹介したように、この変換には関数 str( ) を用います。$u' 第' + str(trials.thisN+1) + u' 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 **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 probeLen 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 7.5 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。

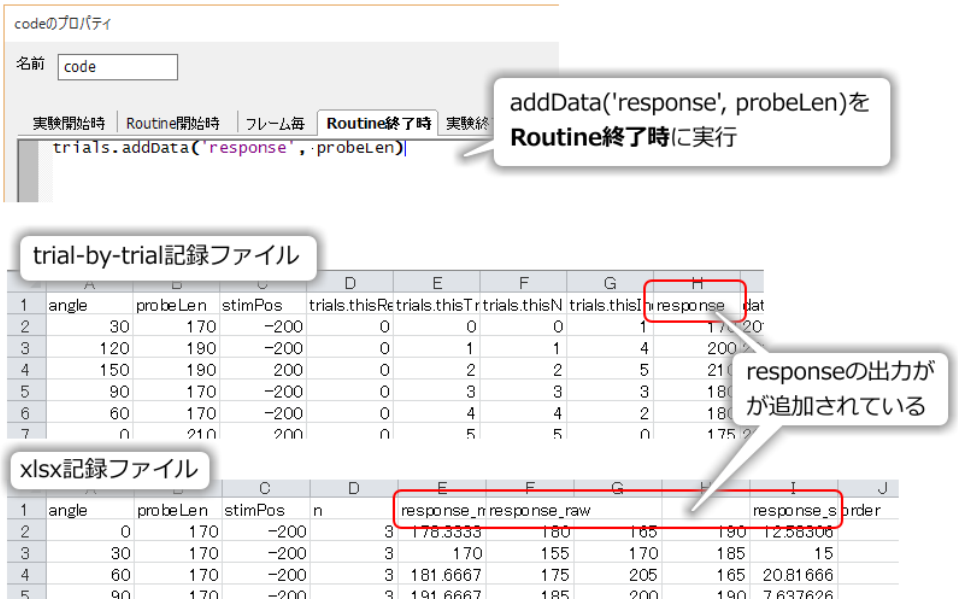


図 7.5　addData( ) メソッドによる変数の出力。

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）かを知る事が出来ます。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出来ます。

```
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出来ません。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出来る。
- 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出来る。

7.5 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。exp07a.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出来てしまいます。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出来ませんので、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 50pix から 350pix の範囲を超えて短くしたり長くしたり出来ないようにしてみましょう。

作成済みの exp07a.psyexp を exp07b.psyexp という別名で保存してください。そして 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出来るはずです。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。

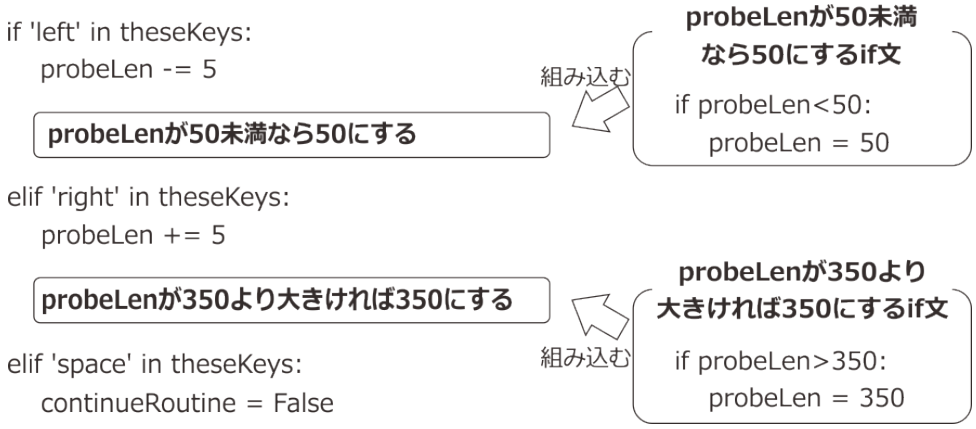


図 7.6　プローブの長さを制限する方法その 1。exp07a.psyexp の Code コンポーネントに書いたコードに組み込む処理を日本語で記入したものを左側に、組み込む処理に対応するコードを右側に示しています。

第一の方法は、長さを 5 増加させた時に 350 より大きくなっていないか確認して、なっていれば 350 に修正し、長さを 5 減少させた時に 50 未満になっていないか確認して、なっていれば 50 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 7.6 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出来ます。「probeLen が 50 未満なら 50 にする」という部分だけを考えると、これは 50 以上なら何もしないということですから、else は省略出来て 図 7.6 右上のように書けます。同様に「probeLen が 350 より大きければ 350 にする」という処理も 図 7.6 右下のように書けます。 図 7.6 左側のコードの日本語で記入した部分に、 図 7.6 右側の対応するコードを埋め込むと、 図 7.7 左に示すコードが得られます。これだけで完成です。

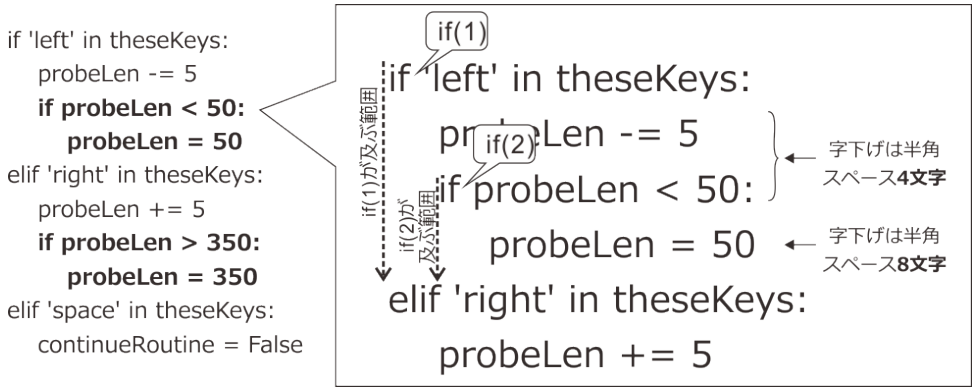


図 7.7　if 文を組み込んで得られたコード。if、elif、else の及ぶ範囲は、下方向に向かってこれらの語が出現した行と字下げ幅が狭いか同じ行に出会うまでです。if (2) が半角スペース 4 文字字下げされているので、if (2) の次の行は if (2) よりさらに 4 文字字下げして 8 文字字下げとなります。

図 7.7 左のコードをもう少ししっかり見ておきましょう。 図 7.7 左のコードの冒頭部分を拡大したのが 図 7.7 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 7.7 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 7.7 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 *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、exp07b.psyexp を開いて、 図 7.7 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** に入力してください。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 50pix や 350pix 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 ±5 から ±50 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。exp07b.psyexp を閉じて exp07a.psyexp を開きなおして、exp07c.psyexp という名前で保存して作業しましょう。第一の方法では、probeLen の長さを増減した直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 7.8 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 7.8 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 7.8 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。

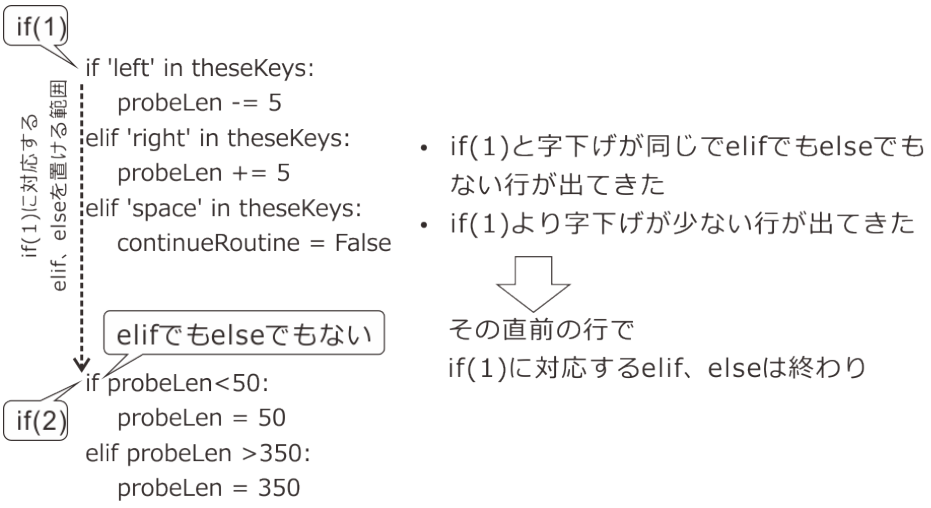


図 7.8　プローブの長さを制限する方法その 2。if に対応する elif、else を置ける範囲に注意。

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 7.8 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 7.8 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 50 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 170、190、210、230 の 4 通りしかなくて、±5 ずつしか増減しないのですから、5 を引いて 50 未満になるのは probeLen が 50 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 50 より大きい」時には probeLen から 5 を引いても 50 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 > 50:
    probeLen -= 5
```

同様に、probeLen に 5 を加えて 350 より大きくなるのは probeLen が 350 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 350 未満」の時には probeLen に 5 を足しても 350 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 < 350:
    probeLen -= 5
```

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 7.7 や 図 7.8 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を練習問題として、この章を終えることにしましょう。

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。

7.6 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。

「どちらもあっさり出来てしまった」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 7.8 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。

7.7 この章のトピックス

7.7.1 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出来ますが、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出来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。

7.7.2 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 を省略することが出来るのです。

7.7.3 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 7.9 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 7.9 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 7.9 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出来ます。それに対して 図 7.9 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 7.9 右のコードはエラーとなり実行できません。

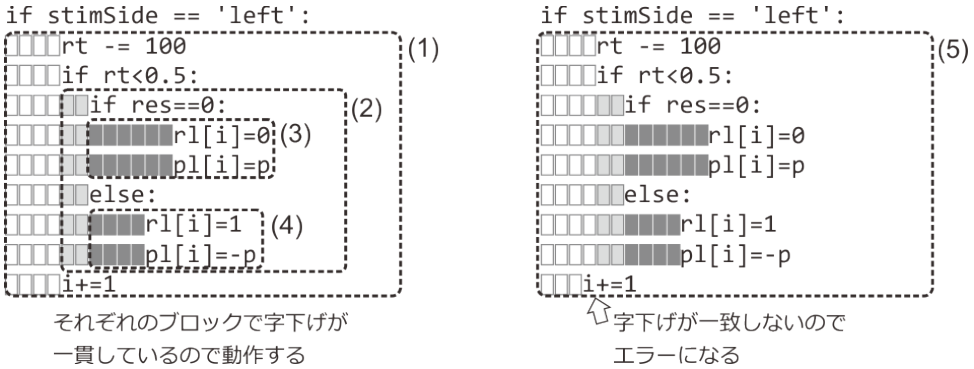


図 7.9　4 文字以外の半角スペースによる字下げ。それぞれのブロック内で字下げが一貫していればエラーにはなりません。

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 12 の例をご覧ください。図 7 12 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 12 左のコードはエラーとならず実行できます。一方、図 7 12 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。

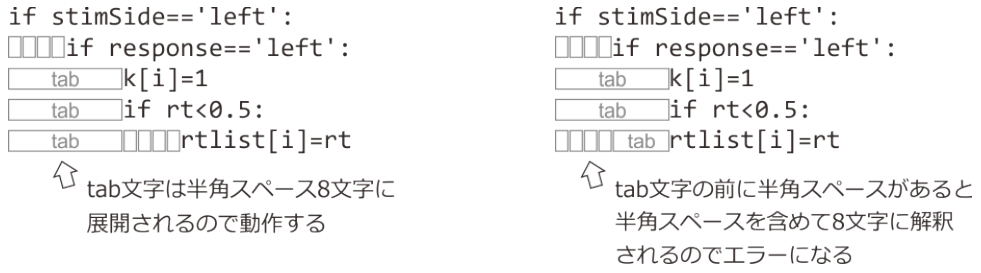


図 7.10　Tab 文字による字下げ。基本的には Tab 文字は半角スペース 8 文字に置換されると考えておけばよいですが、左の例のように半角スペースの後ろに Tab がある場合は半角スペースと Tab をまとめて 8 の倍数個のスペースとして解釈されてしまいます。

第 8 章

マウスで刺激を動かそう—鏡映描写課題

8.1 この章の実験の概要

これまでの章では実験参加者の反応計測のためにキーボード用いてきましたが、この章ではキーボードと並んで普及している PC の入力機器であるマウスを反応に用いる方法を学びます。ただ今までカーソルキーの左右を押していたのをマウスの左クリック、右クリックに置き換えただけでは面白くないので、マウスならではの課題である鏡映描写課題を Builder で作成したいと思います。

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 8.1 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出来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出来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 8.1 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 8.1 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 8.2 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 300pix 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 30pix のディスク (円) をゴール地点に提示します。パスは幅 20pix 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

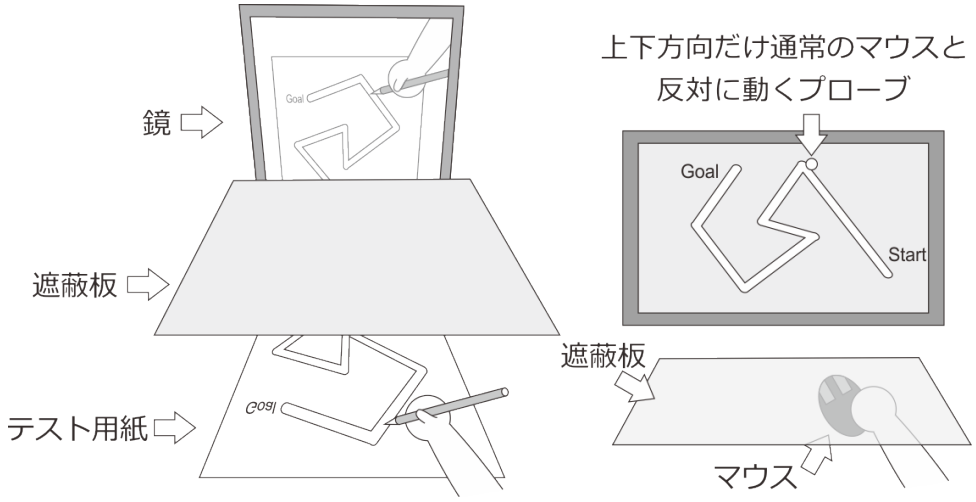


図 8.1　鏡映描写課題。遮蔽版の下に置かれたテスト用紙に描かれた図形を、鏡に映った像を見ながらペンでなぞります。スクリーンとマウスを用いてこの実験装置をシミュレートします。

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。

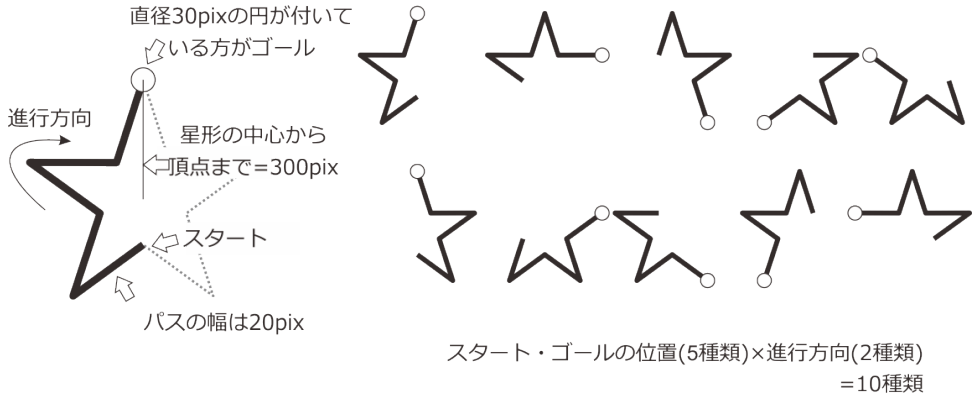


図 8.2　実験で使用するパス。星形の辺を時計回りまたは反時計回りに進んで半周します。5 種類のスタート・ゴールの組み合わせと 2 種類の進行方向で合計 10 種類のパスが得られます。

パスを描画するための座標値を計算してみましょう (図 8.3)。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 8.3 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 8.2 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章末の 図 8.20 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。

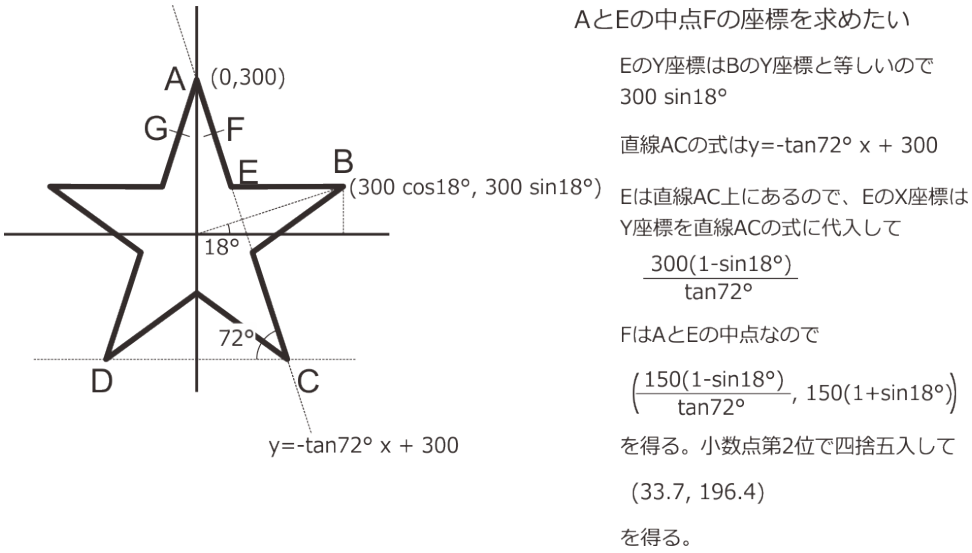


図 8.3　星形を描画するための座標値の計算。F の座標が得られれば G の座標も直ちに得られます。F と G の座標を 72 度ずつ回転させればすべての辺の中点が得られます。

このファイルを exp08cnd.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 8.3 から計算した辺の長さは 218.0pix なのですが、ぴったり 218pix にする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 240pix にしましょう。パスの幅は 20pix なので、 **サイズ [w, h] $** に指定する値は [240, 20] 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 10pix 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 8.4 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、出来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、exp08cnd.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、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。

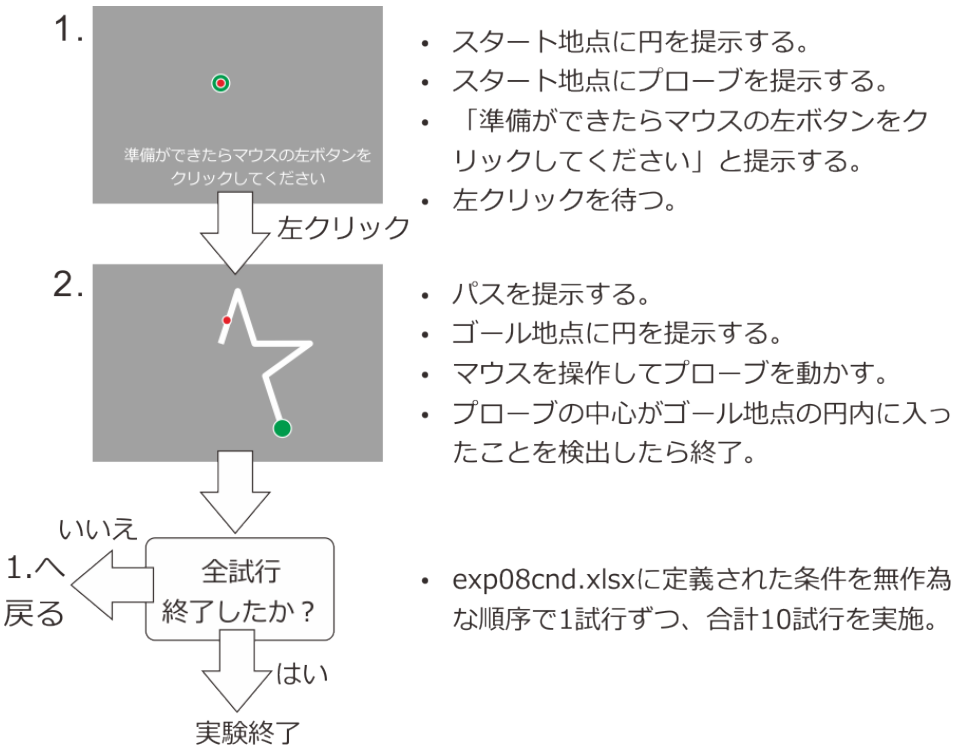


図 8.4　実験の流れ。

8.2 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。 図 8.5 左に示した Mouse コンポーネントのアイコンをクリックすると、 図 8.5 右のプロパティ設定ダイアログが開きます。

Mouse コンポーネントのプロパティ設定ダイアログのうち、 **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特に解説は必要ないでしょう。 **ボタン押しで Routine を終了** は Keyboard コンポーネントと同様に、チェックがついていればマウスのボタンが押されたときにルーチンが終了します。 **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 8.1 をご覧ください。最後の **時刻の基準** は Mouse コンポーネントで使用される時刻の 0 秒がルーチン開始時 (routine) か実験開始時かを選択します。

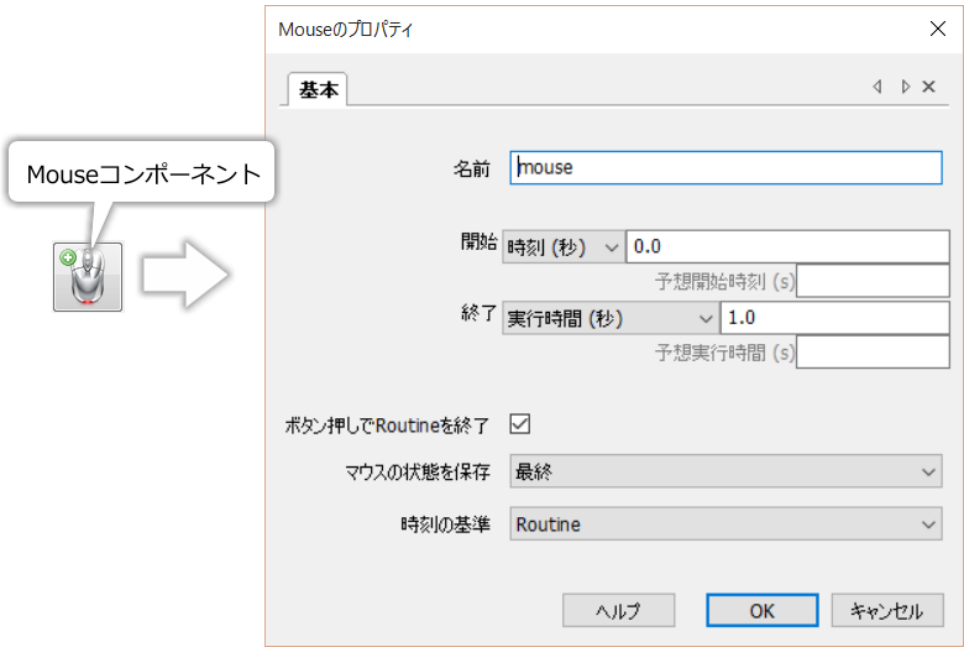


図 8.5　Mouse コンポーネントと設定ダイアログ。

表 8.1　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 8.1 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出来る。
- 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出来る。
- 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。

8.3 実験の作成

Mouse コンポーネントの解説も終わったところで、実験の作成に入りましょう。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。Builder で新規に実験を作成して以下の作業を行い、exp08a.psyexp という名前で保存するものとします。

- 実験設定ダイアログ
  - 「xlsx 形式のデータを保存」をチェックする。

  - **単位** を pix にする。

- trial ルーチン
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - Mouse コンポーネントをひとつ配置して、 **名前** を mouseTrial にする。
    - **終了** を空白にする。
    - **ボタン押しで Routine を終了** のチェックを外す。
    - **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 7 つ配置して **終了** を空白にし、 **名前** を path1、path2、path3、path4、path5、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。
  - pathN (N=1〜5) について以下の作業を行う。
    - **回転角度 $** を pathNori (N は pathN の N と一致させる) にして、「繰り返し毎に更新」に設定する。
    - **位置 [x, y] $** を pathNpos (N は pathN の N と一致させる) にして、「繰り返し毎に更新」に設定する。
    - **サイズ [w, h] $** を [240, 20] に設定する。
  - goalDisc について以下の作業を行う。
    - **頂点数** を 32 にする。
    - Fill color 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 [30, 30] にする。
  - probe について以下の作業を行う。
    - **頂点数** を 32 にする。
    - Fill color 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 [10, 10] 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** のチェックが付いていることを確認する (初期状態で付いている)。
    - **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 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。
    - **頂点数** を 32 にする。
    - Fill color 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 [30,30] にする。
  - probeReady について以下の作業を行う。
    - **頂点数** を 32 にする。
    - Fill color 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 [10,10] 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 24 にする。
    - **位置 [x, y] $** [0, -320] 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。
  - **繰り返し条件** に exp08cnd.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出来ないのかをはっきりさせましょう。まだ exp08a.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 6 章、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 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。

8.4 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 exp08a.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出来ます。

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 8.2 に代表的な Mouse クラスの主なメソッドを示します。

表 8.2　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。

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出来ます。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。exp08a.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出来る。

8.5 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 8.6 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。

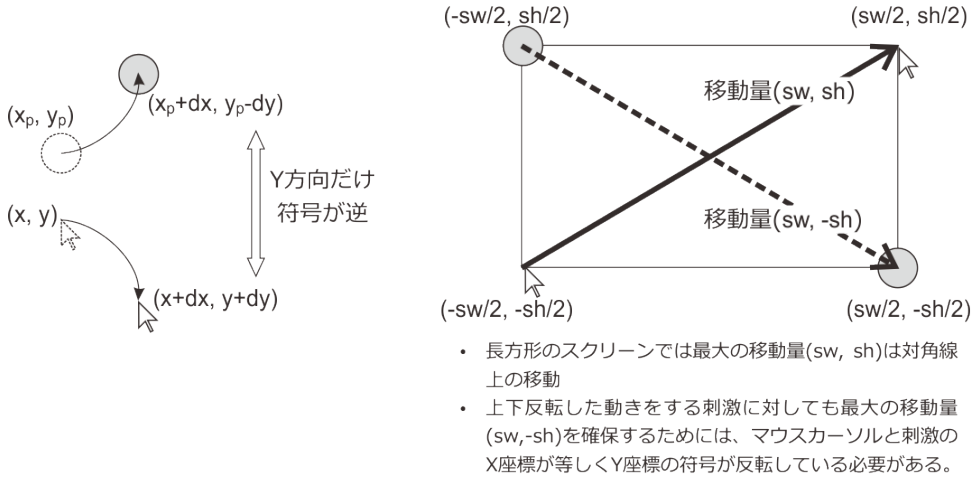


図 8.6　マウスカーソルと上下方向だけ反転して移動するプローブ。マウスカーソルの Y 座標の符号を反転した座標をプローブの座標とすれば、最大の移動量 (対角線上の運動) に対応することが出来ます。ただし sw、sh はスクリーン幅、スクリーン高を表しているとします。

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 8.6 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出来ません。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。

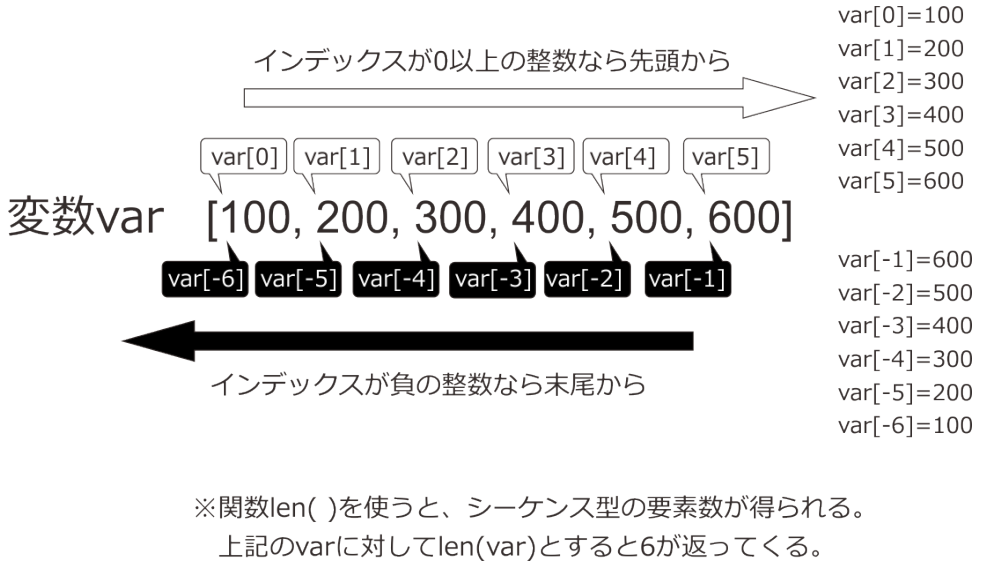


図 8.7　シーケンス型のデータからの要素の取出し。インデックスが 0 以上の整数なら先頭から、負の整数なら末尾から数えます。

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 8.7 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 に代入することが出来ます。

```
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出来ます。

```
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。

もうひとつ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出来ます。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 8.8 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 [ ] を適用することが出来て、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] と書くことも出来ます。

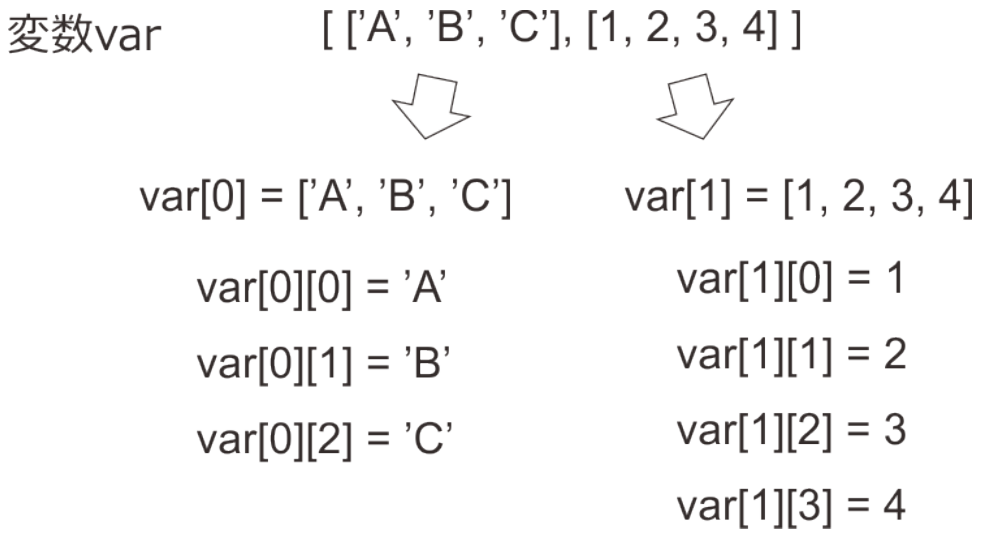


図 8.8　多重シーケンスからの要素の取出し。var[0] とするとリストが得られるので、さらに [ ] 演算子を適用して var[0][0] とすると var[0] の最初の要素を取り出せます。

ここまでの章で、**位置 [x, y] $** や **サイズ [w, h] $** といったプロパティにシーケンス型の一種であるリストをたくさん書いてきました。[0, 0] とか [240,20] とかいう具合に [ ] で括ってある値がリストでしたね。ちょっとややこしいのですが、この [ ] とシーケンス型の要素を取り出す [ ] 演算子の違いをよく理解しておいてくださ

い (図 8.9)。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、[240, 20][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。 [240, 20][0] は「[240, 20] 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 240 です。よろしいでしょうか？

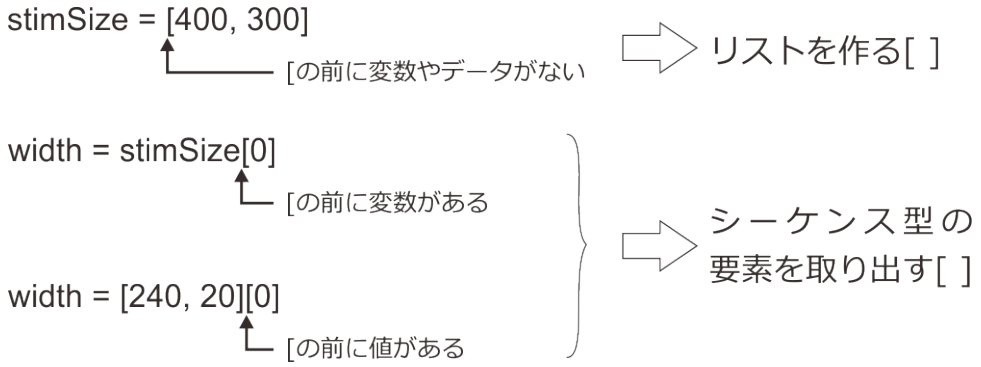


図 8.9　リストを作る [ ] とシーケンス型の要素を取り出す [ ]。

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出来ます。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出来ます。'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。

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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- [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。

8.6 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 15pix 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出来ます。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  < 15**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 [30, 30] 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 15pix 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。

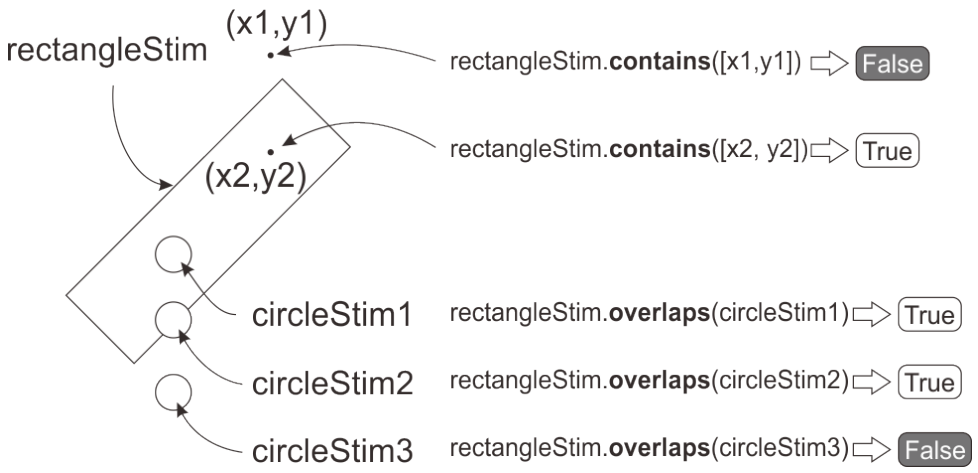


図 8.10　Contains( ) と Overlaps( ) メソッドの例。

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 8.10 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 8.10 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出来ませんので、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、exp08a.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。

8.7 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 8.2 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 8.2 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、exp08a.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ません。

これで 図 8.4 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。

8.8 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 8.3) よりもさらに面倒な計算なので、出来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]):
    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。

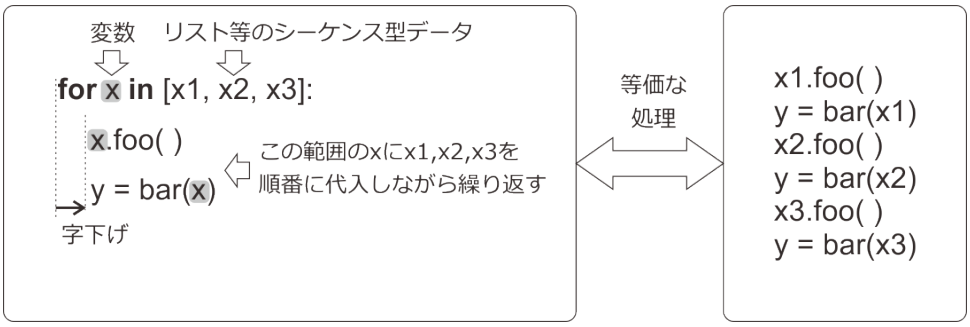


図 8.11　for 文による繰り返し処理。

図 8.11 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 8.11 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 8.11 左の for 文は、 図 8.11 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 8.12 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 8.12 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。

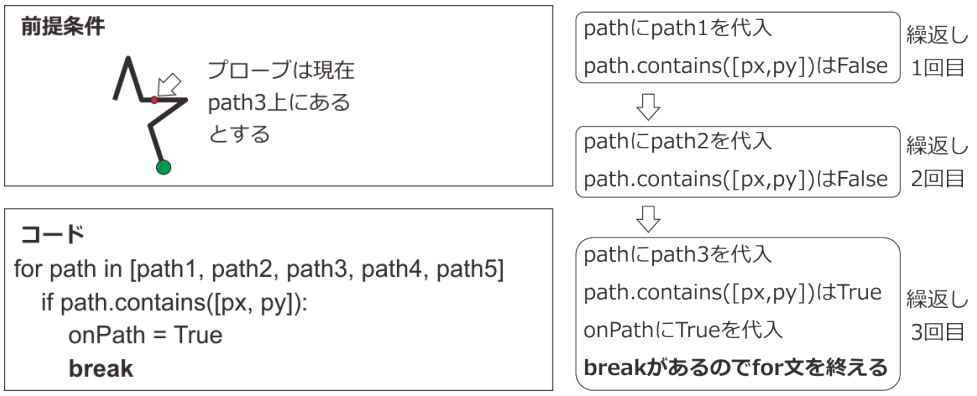


図 8.12　break による for 文の中断。if 文などと組み合わせて使用するのが一般的です。

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出来る。
- 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出来る。

8.9 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 exp08a.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、exp08a.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 8.13 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。

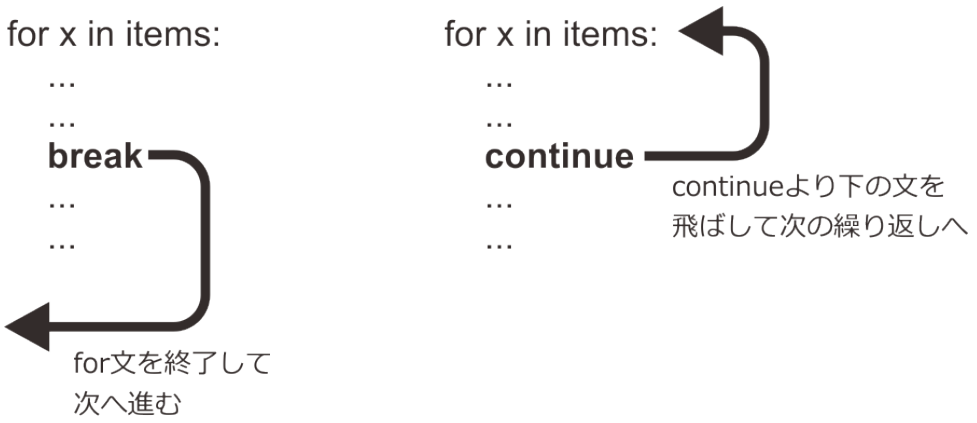


図 8.13　break と continue の違い。残りの要素に対して処理をする必要がない場合は break、必要がある場合は continue を使用します。

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： *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出来るのです。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
        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 の値を得ることが出来ます。

```
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： **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 8.14 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。

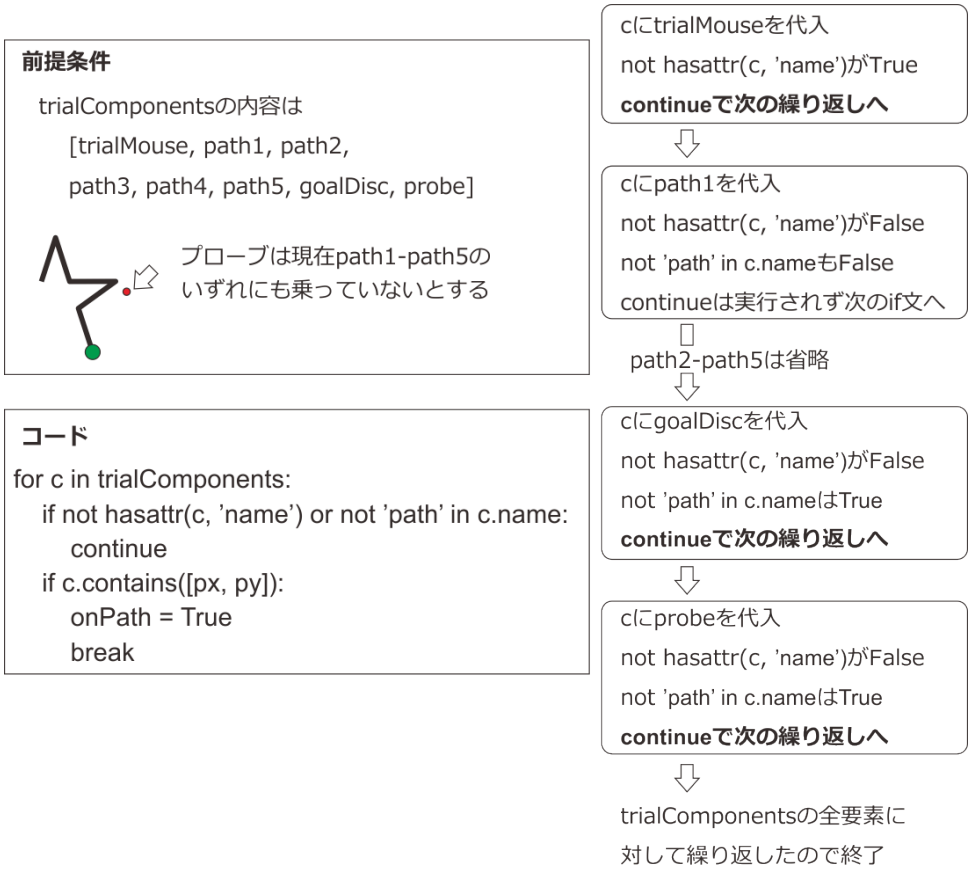


図 8.14　continue による繰り返し処理のスキップ。

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。
- 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。

8.10 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、 **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 8.3 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 8.3　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 8.3 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 8.15 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出来ないデータを引数にすることは出来ません。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( ) で追加することが出来ません。append( ) を使用しましょう。

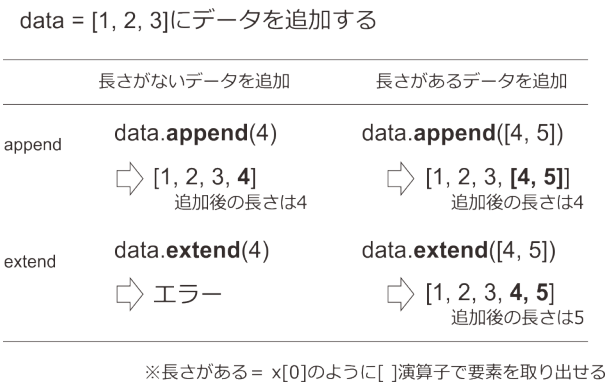


図 8.15　append( ) と extend( ) によるリストへのデータの追加。

それではコードを入力していきます。まず、exp08a.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 8.9 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。

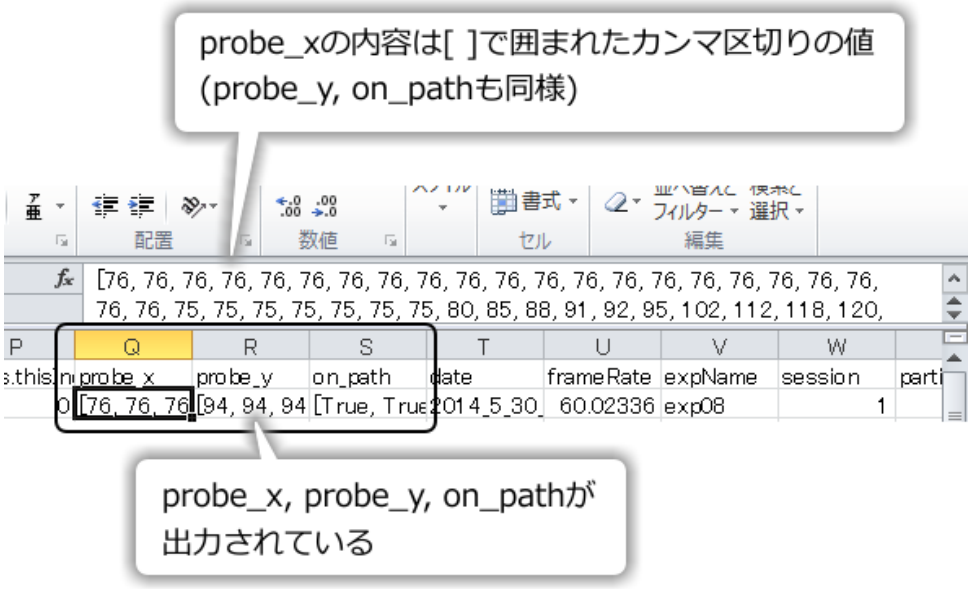


図 8.16　trial-by-trial 記録ファイルにおけるプローブ座標と判定結果の出力。Python のリストと同様に、[ ] で囲まれたカンマ区切りの値として出力されています。

これで作業は完了しました。exp08a.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 8.16 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 8.17 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出来ます。

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。

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出来る。
- 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。

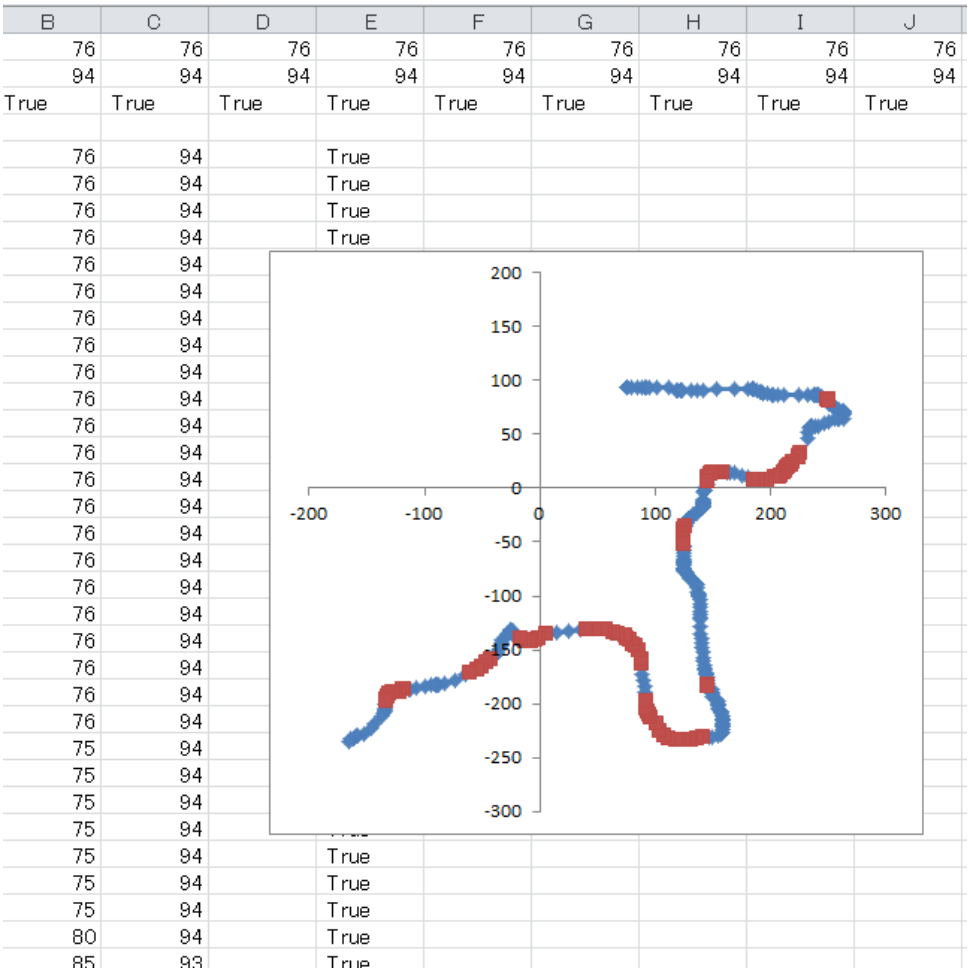


図 8.17　probe_x、probe_y、on_Path を Excel の「データの区切り位置」機能を用いて復元し、軌跡をグラフとしてプロットしたもの。軌跡の青い部分はパス上にプローブの中心が乗っていたことを、赤い部分は乗っていなかったことを示しています。

8.11 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。

exp08a.psyexp を exp08b.psyexp という別名で保存して、そちらを使って作業しましょう。exp08b.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 件に間引くことが出来るのでしょうか。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 8.18)。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。

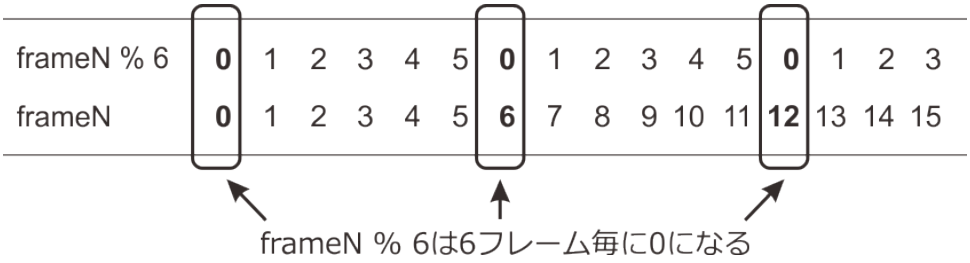


図 8.18　剰余演算子を利用したデータの間引き。

% 演算子のもう一つの重要な応用例を 図 8.19 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、Python の整数同士の除算が余りを切り捨てることを思い出せば (第 5 章)、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。

% 演算子の話はこのくらいにしておいて、exp08b.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

図 8.19　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。

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)
    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。exp08b.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出来る。
- 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出来る。

8.12 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こうというわけです。

ここまでの作業で作成した exp08a.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 8.2 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。PsychoPy

の Mouse クラスには isPressedIn(shape, buttons) という、shape に指定されたオブジェクト上で buttons に指定されたボタンが押されたときに True を返すというメソッドがあり、本実験のような座標の上下反転などを行わない場合は、このメソッドを使用すればボタン風のユーザーインターフェースは実現できます。しかし、「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。

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。

8.13 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出来るはずです。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 exp08a.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。

8.14 この章のトピックス

8.14.1 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。*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。

8.14.2 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出来ます。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
if not 'path' in c.name
    continue
```

path1pos	path1ori	path2pos	path2ori	path3pos	path3ori	path4pos	path4ori	path5pos	path5ori	startPos	goalPos
[176.3,92.7]	0	[197.1,28.6]	-36	[142.7,-139.1]	-108	[88.2,-178.6]	-144	[-88.2,-178.6]	-36	[67.4,92.7]	[-176.3,-242.7]
[142.7,-139.1]	-108	[88.2,-178.6]	-144	[-88.2,-178.6]	-36	[-142.7,-139.1]	-72	[-197.1,28.6]	-144	[109,-35.4]	[-285.3,92.7]
[-88.2,-178.6]	-36	[-142.7,-139.1]	-72	[-197.1,28.6]	-144	[-176.3,92.7]	0	[-33.7,196.4]	-72	[0,-114.6]	[0,300]
[-197.1,28.6]	-144	[-176.3,92.7]	0	[-33.7,196.4]	-72	[33.7,196.4]	-108	[176.3,92.7]	0	[-109,-35.4]	[285.3,92.7]
[-33.7,196.4]	-72	[33.7,196.4]	-108	[176.3,92.7]	0	[197.1,28.6]	-36	[142.7,-139.1]	-108	[-67.4,92.7]	[176.3,-242.7]
[33.7,196.4]	-108	[-33.7,196.4]	-72	[-176.3,92.7]	0	[-197.1,28.6]	-144	[-142.7,-139.1]	-72	[67.4,92.7]	[-176.3,-242.7]
[-176.3,92.7]	0	[-197.1,28.6]	-144	[-142.7,-139.1]	-72	[-88.2,-178.6]	-36	[88.2,-178.6]	-144	[-67.4,92.7]	[176.3,-242.7]
[-142.7,-139.1]	-72	[-88.2,-178.6]	-36	[88.2,-178.6]	-144	[142.7,-139.1]	-108	[197.1,28.6]	-36	[-109,-35.4]	[285.3,92.7]
[88.2,-178.6]	-144	[142.7,-139.1]	-108	[197.1,28.6]	-36	[176.3,92.7]	0	[33.7,196.4]	-108	[0,-114.6]	[0,300]
[197.1,28.6]	-36	[176.3,92.7]	0	[33.7,196.4]	-108	[-33.7,196.4]	-72	[-176.3,92.7]	0	[109,-35.4]	[-285.3,92.7]

図 8.20　本章で用いる条件ファイル

第 9 章

実験の流れを制御しよう―強化スケジュール

9.1 この章の実験の概要

突然ですが、野鳥観察を趣味にしている人が居るとします。この人が、とある湖畔に明け方に訪れた時にずっと探していた野鳥を見つけました。その後も時々同じ湖畔で明け方にその野鳥を見かけることがあって、今では機会があればいそいそと朝早くに家を出てその湖畔へ向かうようになりました。―いきなり何のことかと思われるかも知れませんが、この例は「明け方に湖畔に行く」という行動をすると、「目当ての野鳥が観察できる」という結果が時々生じて、その結果によって「明け方に湖畔に行く」という行動が生じる頻度が高まったと言うことが出来ます。このようにある行動に随伴して生じる出来事によって、その行動の発生頻度が高まることを強化と呼び、行動に随伴して生じる出来事を強化子と呼びます。この例では「湖畔に行く」という行動に対して「目当ての野鳥が観察できる」という強化子が毎回ではなく「時々」生じているのですが、毎回生じるのか、一定間隔で生じるのか、といった強化子の出現スケジュールの違いによって行動の発生頻度の変化パターンが異なることが知られています。強化子の出現スケジュールのことを強化スケジュールと呼びます。

図 9.1 は基本的な強化スケジュールを示しています。ポイントは「強化子が生じるタイミングが時間によって決まっているか、行動回数によって決まっているか」と、「時間や回数が一定であるか、変動するか」です。これらの組み合わせで Fixed Interval (FI)、Fixed Ratio (FR)、Variable Interval (VI)、Variable Response (VR) の 4 種類のスケジュールが出来ます。FI、VI において強化子が得られるまでに必要な時間を「強化時間」、FR、VR において強化子が得られるまでに必要な反応回数を「強化回数」と呼ぶことにします。FI、VI では強化時間が経過したら自動的に強化子が得られるのではなく、強化時間が経過した後に行われた最初の行動によって得られる点に注意してください。

この章では、Code コンポーネントを活用してこれらの強化スケジュールを Builder で実現します。 図 9.2 に実験の手続きを示します。実験参加者の課題は、キーボードのスペースキーを押して出来るだけ早く指定された点数の「得点」を獲得することです (例えば 20 点)。スクリーンには実験が開始してからの時刻と現在の得点が表示されていて、参加者は強化スケジュールで定められた条件を満たした状態でスペースキーを押すと、1 点を獲得することができます。得点を獲得すると同時に 2000Hz の音が 0.2 秒間鳴り、スクリーン上の時刻と得点の表示の背後に赤い長方形が 1 秒間表示されます。指定された得点に到達したら、その時点で実験は終了します。

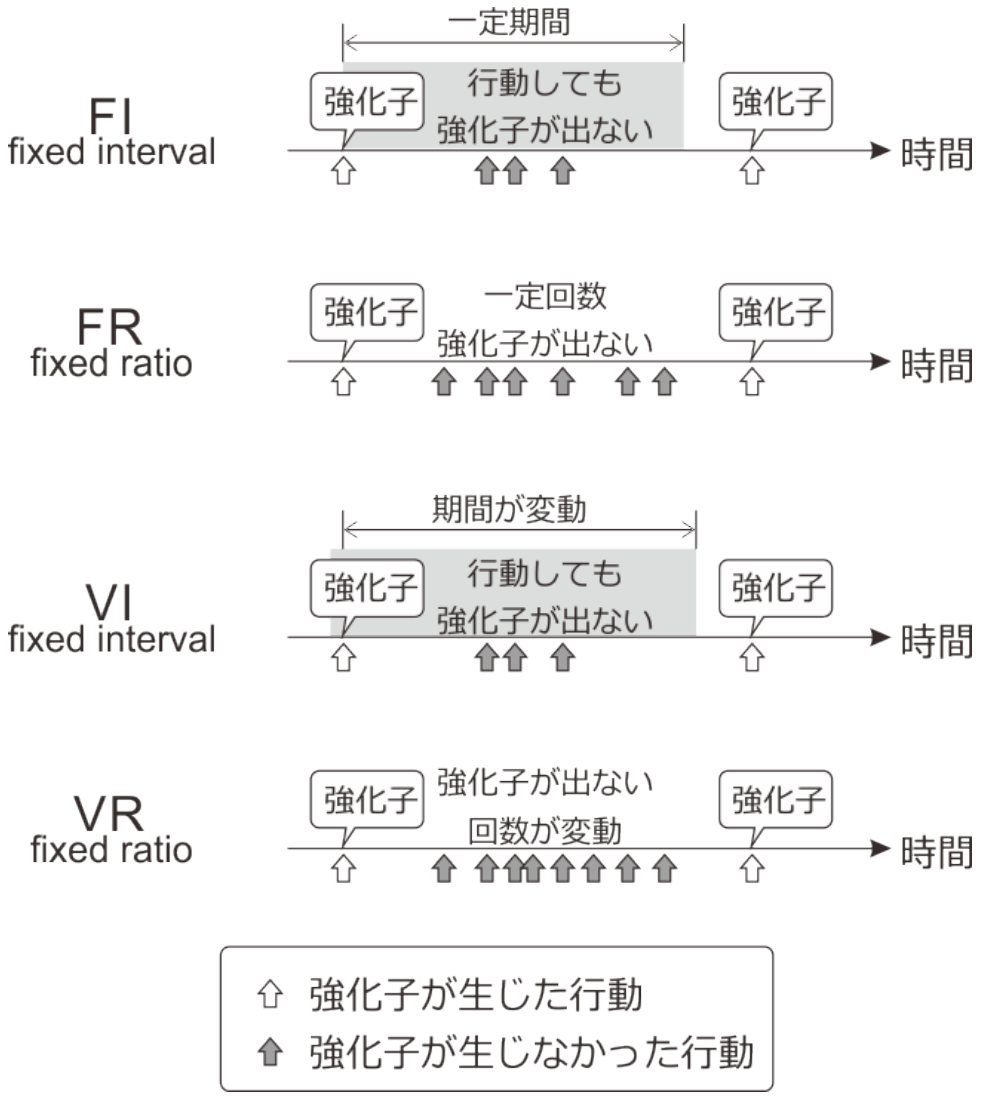


図 9.1　基本的な強化スケジュール。

今回の実験では単位として norm を用いて、経過時刻および得点の表示用テキストの大きさ (**文字の高さ $**) は 0.1、経過時刻の位置は [0, 0]、得点の位置は [0, -0.2] とします。また、得点獲得時の赤い長方形は大きさ [0.5, 0.5]、位置 [0, -0.1] とします。

得点を獲得できる条件として、FI、FR、VI、VR の 4 種類のスケジュールを実行時に選択するようにします。いろいろな作成方法が考えられますが、ここでは「FI は強化時間が 1 種類しかない VI」と考えてみましょう。そうすると、FI は VI 用の実験に条件が 1 種類しか定義されていない条件ファイルを与えることで実現できます。 図 9.2 の 1. のように実験開始時の実験情報ダイアログに条件ファイルを指定する condition という項目と、各条件の繰り返し回数を指定する nReps という項目を用意しておいて、「条件 1 種類の条件ファイルを condition に指定して、nReps を 20 に」すれば 20 点獲得で終了する FI です。「条件 5 種類の条件ファイルを condition に指定して、nReps を 4 に」すれば、5 × 4=20 点獲得で終了する VI です。同様に、「FR は強化回数が 1 種類しかない VR」と考えれば、VR 用の実験を用意すれば FR に対応できます。結論として、VI 用と VR 用の実験を作成するだけで 4 種類のスケジュールに対応できます。

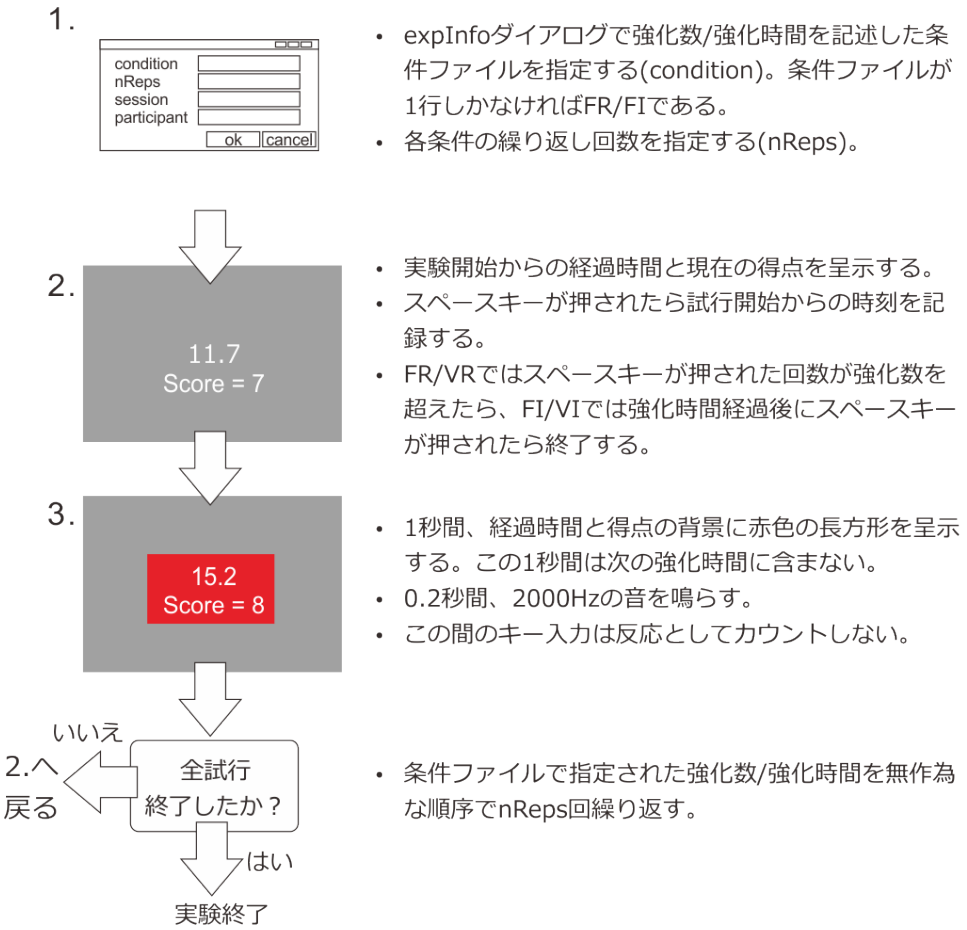


図 9.2　実験の手続き。

以上が実験の概要です。この章では、以上の実験を土台として、一定時間経過したら実験が終了したり、参加者の反応によって次の課題が変化したりといった高度な実験の流れの制御を学びたいと思います。実験の作成に入る前に、この章で初登場の Sound コンポーネントについて解説しておかなければいけません。ついでですので、Movie コンポーネントの使い方についても簡単に説明しておきます。

9.2 Sound コンポーネントと Movie コンポーネント

図 9.3 に Sound コンポーネントと Movie コンポーネントのアイコン及びプロパティ設定ダイアログを示します。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 **音** と **ボリューム $** です。 **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出来ます。また、2000 という具合に正の数値を入力すると、その周波数の音がなります。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 を用いるのが確実です。 **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。 **開始** および **終了** で定められた時間が音声ファイルの時間より短い場合は、音声ファイルの再生が途中で終了します。

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 **バックエンド** と **動画ファイル** です。 **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。 **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出来ます。動画ファイルの元の解像度のまま再生する場合は **サイズ [w, h] $** は空白でも構いません。Sound コンポーネントと同様に、 **開始** および **終了** で定められた時間が動画ファイルの時間より短い場合は、動画ファイルの再生が途中で終了します。 **Routine を終了** をチェックしておくと、動画再生が終了すると同時にルーチンが終了します。 **バックエンド** については、とりあえず「avbin」のままにしておくのが良いと思います。詳しくは *Movie* **コンポーネントのバックエンド** をご覧ください。

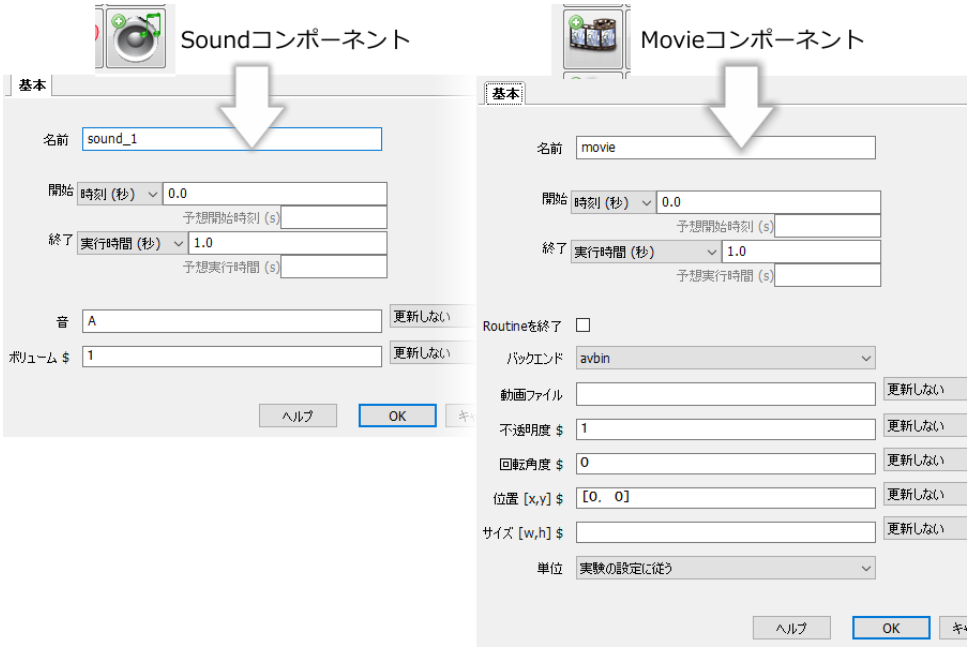


図 9.3　Sound コンポーネントと Movie コンポーネントのアイコン及びプロパティ設定ダイアログ

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 MPEG4 形式をよく使用しています。いずれのフォーマットを用いるにせよ、動画再生はかなり PC への負担が大きい処理なので、解像度をむやみに上げるべきではありません。例えば動画撮影に使用したカメラの記録フォーマットが 1920 × 1080 の Full HD 画質で、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。先ほど **サイズ [w, h] $** を指定することで動画の拡大縮小が出来ると書きましたが、 **サイズ [w, h] $** を用いる方法は PC に十分な処理能力がある場合のみ利用してください。PC の処理能力は皆さんの実行環境に寄りますので一概には言えませんが、再生させてみて動画がカクカクする場合は確実に処理能力が不足しています。

音声ファイルや動画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、「動画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や動画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 コンポーネントや Movie 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_Sound の再生終了時

にルーチンを強制終了させることが出来ます。

```
if stim_Sound.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、第 2 章で「とりあえず無視」した機能を利用します。各コンポーネントの **開始** および **終了** のプルダウンメニューには 「条件式」 という選択項目があります。これを選択すると、コンポーネントの開始や終了のタイミングを条件式によって指定することが出来ます。第 2 章の時点では条件式について説明していなかったので無視しましたが、この章まで進んだ皆さんならもう説明は不要でしょう。 図 9.4 のように入力することによって、音声ファイルや動画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、終了を設定できます。もちろん 「条件式」 に入力するのは条件式であれば何でも構いませんので、Code コンポーネントを利用すれば多彩な制御が可能となります。ぜひ覚えておいてください。

図 9.4　**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

なお、非常にファイルサイズの大きい音声ファイルや動画ファイルを再生しようとすると、読み込みが間に合わずに正常に再生されない場合があります。このような場合に便利なのが第 2 章からずっと実験の作成手順で trial ルーチンから消去され続けてきた Static コンポーネントです。Static コンポーネントのアイコンはコンポーネントペインの Custom のグループにあります。Static コンポーネントを配置すると、ルーチン上に赤い領域が出現します (図 9.5)。それと同時に、ルーチンに配置されている各コンポーネントのプロパティ設定ダイアログの、更新方法のメニューに「trial の ISI の間に更新」という項目が追加されます。「trial の ISI」の部分は Static コンポーネントを配置したルーチン名と、Static コンポーネントの **名前** によって決まります。instruction というルーチンに preread という名前で Static コンポーネントを配置したのであれば、「instruction

の preread の間に更新」という項目が追加されます。

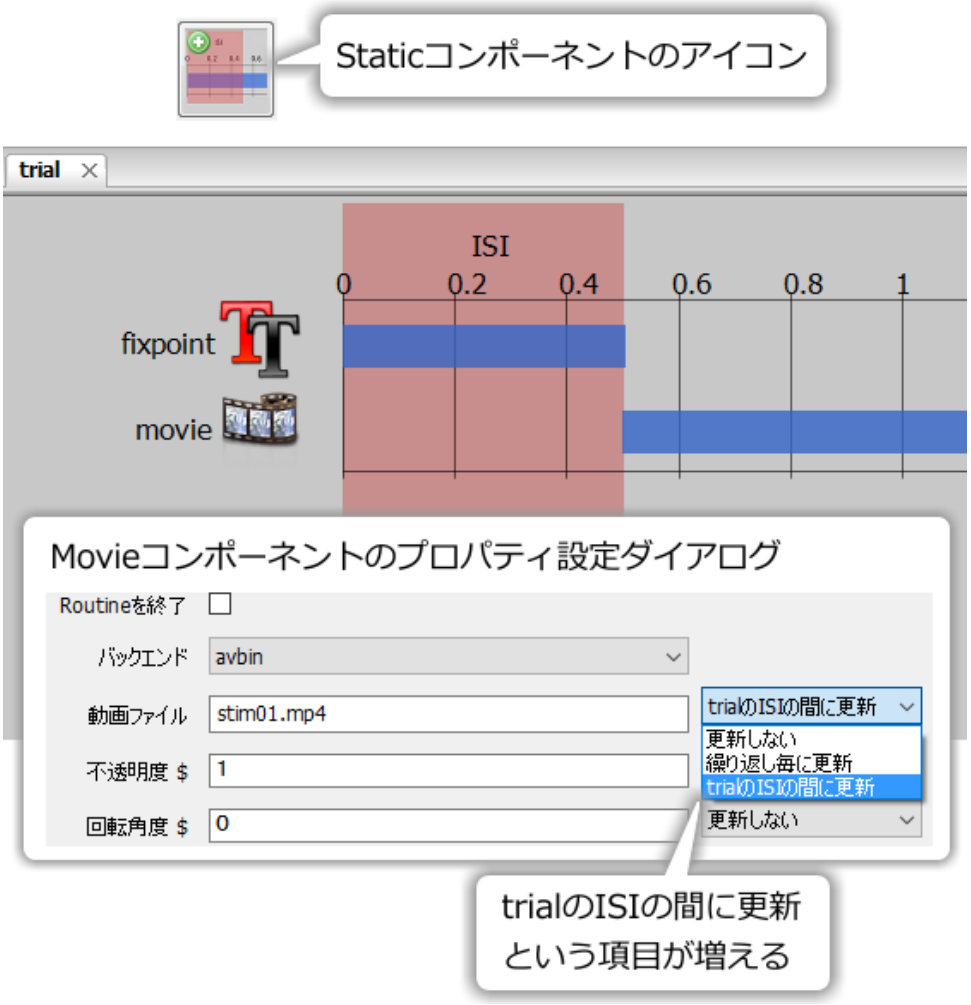


図 9.5　Static コンポーネントを利用して、ファイルサイズの大きい刺激を PC の負荷が小さい時間帯に読み込むことが出来ます。

図 9.5 のように Movie コンポーネントの **動画ファイル** で「…の間に更新」を選択すると、Static コンポーネントで指定されている時間内に動画ファイルの読み込みが行われます。「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。やって出来ない事はないのですが、精度が保障されません。 図 9.5 の fixpoint という Text コンポーネントのように、Static コンポーネントの期間中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。

以上で Sound コンポーネントと Movie コンポーネントの解説は終了ですが、最後にひとつ注意点を挙げておきます。Sound コンポーネントによる音声の再生タイミングはかなり「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。高性能なサウンドモジュールを追加することによって多少改善しますが、視覚－聴覚の相互作用の研究を考えておられる方は Builder で正確な実験をするのはかなり厳しいと思っておいてください。刺激を動画として作成するのもひとつの対策でしょう。

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出来る。
- 指定されたキーコードの音を鳴らすことが出来る。
- 音声のボリュームを指定できる。
- MPEG2 形式の動画ファイルを再生できる。
- 動画ファイルを拡大縮小して再生することが出来る。
- 音声ファイル、動画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、動画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出来る。
- なぜ短時間に複数の 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のかを説明することが出来る。

9.3 FI/VI 実験の作成

Sound コンポーネントの説明が終わったところで、実験の作成に入りましょう。まずは FI/VI 用の実験から作成します。以下の解説では、Builder で新規に実験を作成して以下の作業を行い、exp09vi.psyexp という名前で保存したものとします。

- 実験設定ダイアログ
  - **実験情報ダイアログ** に nReps と condition という項目を追加する。
  - **単位** を norm にする。
- trial ルーチン
  - 最初から Static コンポーネントが配置されている場合は削除する。
  - Text コンポーネントをふたつ配置して、それぞれ **名前** を scoreTrial、clockTrial にする。両方とも **終了** を空白にする。

    - scoreTrial の **位置 [x, y] $** を [0.0, -0.2] にする。 **文字列** に$'Score:'+str(score) と入力し、「繰り返し毎に更新」に設定する。
    - clockTrial の **文字列** に—-と入力しておく。
  - 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。
    - **名前** を key_resp_Trial にする。
    - **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 $** に'space' と入力する。
    - **記録** を「全てのキー」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。
    - **実験開始時** に score=0 と入力する。
    - **Routine 終了時** に score+=1 と入力する。
- reinforcement 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。
  - 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。
    - **名前** を backgroundRect にする。
    - **終了** が「実行時間 (秒)」で 1.0(=初期値) であることを確認する。
    - **塗りつぶしの色** 、 **輪郭線の色** を red にする。
    - **位置 [x, y] $** を [0.0, -0.1] にする。
  - Text コンポーネントをふたつ配置して、それぞれ **名前** を scoreRF、clockRF にする。両方とも **終了** が「実行時間 (秒)」で 1.0(=初期値) であることを確認する。また、両方とも backgroundRect より上に描画されるようにルーチンペイン上の順序に配慮する。
    - scoreRF の **位置 [x, y] $** を [0.0, -0.2] にする。 **文字列** に$'Score:'+str(score) と入力し、「繰り返し毎に更新」に設定する。
    - clockRF の **文字列** に—-と入力しておく。
  - Sound 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。
    - **名前** を SoundRF にする。
    - **終了** を「実行時間 (秒)」の 0.1 にする。
    - **音** を 2000 に、 **ボリューム $** を 1 にする。

- trials ループ (作成する)
  - trial ルーチンと reinforcement ルーチンを繰り返すように挿入する。
  - **繰り返し回数** $ に expInfo['nReps'] と入力する。
  - **繰り返し条件** に$expInfo['condition'] と入力する。
- exp09fi.xlsx(条件ファイル)
  - interval というパラメータを設定し、値として 10 を入力する。パラメータ名の行を除いて 1 行 1 列の条件ファイルとなる。
- exp09vi.xlsx(条件ファイル)
  - interval というパラメータを設定し、値として 6, 8, 10, 12, 14 を入力する。パラメータ名の行を除いて 5 行 1 列の条件ファイルとなる。

以上の作業に加えて、trial ルーチンの codeTrial の **フレーム毎** に以下のコードを入力してください。

```
if len(key_resp_Trial.rt) > 0 and key_resp_Trial.rt[-1] >= interval:
    continueRoutine = False
```

解説済みのテクニックで作成できる部分は以上です。実はこの時点ですでに FI/VI の実験として使用できる状態まで完成していますので、一度実行してみましょう。実験開始からの時刻が---と表示される以外は、計画通り動作するはずです。条件ファイルに exp09fi.xlsx を指定すれば強化時間 10 秒の FI スケジュールとなりますし、条件ファイルに exp09vi.xlsx を指定すれば強化時間が 6 秒から 14 秒まで変化する VI スケジュールとなります。実験を実行すると作成される記録ファイルの例を 図 9.6 に示します。「xlsx 形式のデータを保存」をチェックしていませんので、作成されるのは trial-by-trial 記録ファイルだけです。内容を確認すると、key_resp_Trial.rt という列に、実験参加者がキーを押した時刻がルーチン開始時を 0 秒として記録されています。このデータを利用すると、実験参加者のキー押し反応の発生頻度がどのような時間経過をたどって変化したかを分析することが可能です。

図 9.6　trial-by-trial 記録ファイルを確認すると、key_resp_Trial.rt にキーを押した時刻がすべて記録されています。このデータを用いて反応の頻度がどのように変化していくかを分析することが出来ます。

ここまでで使用しているテクニックはすでに解説済みであることはすでに述べましたが、Code コンポーネントの働きが今一つピンとこない人が居るかも知れませんので、念のため補足しておきましょう。復習のつもりで読んでください。まず、 **実験開始時** で得点を保持する変数 score の値を 0 に初期化しています。そして、**フレーム毎** では以下の if 文を実行しています。

```
if len(key_resp_Trial.rt) > 0 and key_resp_Trial.rt[-1] >= interval:
    continueRoutine = False
```

この if 文は、実験参加者の最新のキー押し反応が強化時間経過後に押されていればルーチンを終了するという動作を実現しています。詳しく見ていきましょう。まず、and 演算子の左側の len(key_resp_Trial.rt)>0 という式ですが、key_resp_Trial.rt はキーが押された時刻を格納しているデータ属性でした (第 6 章)。len( ) はシーケンス型の要素数を返す関数ですから (第 8 章)、一回でも反応があってキー押しが記録されていれば and 演算子の左辺の式は True になります。続いて and 演算子の右辺を見ましょう。右辺は左辺の式が True でなければ評価されませんので (第 8 章)、必ず 1 個以上の要素が key_resp_Trial.rt に含まれています。ですから、key_resp_Trial.rt[-1] が必ず存在しています。interval は条件ファイルで定義されている変数ですから、右辺の式は記録済みの最後のキー押し反応の時刻が interval の値以上である時に True となります。左辺、右辺の両方の式が True であれば、continueRoutine を False にしてルーチンを終了します。このような方法を採ることによって、ルーチンで行われたすべてのキー押し反応の時刻を記録しつつ、強化時間を過ぎて反応が起こったら直ちにルーチンを終了することが可能となります。そして、ルーチンが終了したということは得点を得たという事ですから、 **Routine 終了時** で score += 1 を実行するわけです。解説を読んでもよくわからなかった方は、第 6 章から第 8 章にかけてしっかりと復習してください。

9.4 Global Clock を用いて実験開始からの経過時間を得よう

すでに前節で FI/VI スケジュールの実験が完成してしまって、この章は一体この後何を学ぶの？と疑問に思われているかたも多いかもしれません。この章の次の目標は、一定時間が経過したら実験を中断するという動作を実装することです。そのための布石として、まずは実験開始からの経過時間をスクリーン上に表示させてみましょう。

Builder には、実験を制御するためにいくつかの「時計」が用意されています。「時計」には実験全体の経過時間を計測するためのグローバルクロックと、各ルーチンが始まってからの経過時間を計測するためのルーチンクロックがあります。これらの時計の実体は、psychopy.core.Clock というクラスのインスタンスで、グローバルクロックは globalClock という変数に格納されています。ルーチンクロックは” ルーチン名 +Clock” という名前の変数に格納されています。ルーチン名が trial なら、ルーチンクロックは trialClock です。

psychopy.core.Clock は今まで紹介してきた PsychoPy のクラスと同様にいくつかのデータ属性とメソッドを持ちますが、クロックの働きを変更するメソッド等を実行すると Builder で作成した実験の動作に悪い影響が出る恐れがありますので、ここでは現在の時間を得るメソッド getTime( ) だけを紹介しておきます。getTime( ) メソッドは、現在の時間を小数で返します。時間の単位は秒です。globalClock.getTime( ) とすれば、実験開始からの経過時間が得られますし、trialClock.getTime( ) とすると trial ルーチン開始からの経過時間が得られます。なお、第 5 章において、ルーチン開始からの経過時刻を保持する t という内部変数が出てきましたが、実はこの t はルーチン実行中に各フレームの処理の最初で t = trialCock.getTime( ) という具合にルーチンクロックの getTime( ) メソッドの戻り値を格納することで得られています。

さて、実験開始からの経過時間を得る方法がわかりましたので、さっそく exp09vi.psyexp を改造して経過時間をスクリーン上に標示してみましょう。exp09vi.psyexp を開いて、trial ルーチンの clockTrial のプロパティ設定ダイアログを開いて、 **文字列** に$globalClock.getTime() と入力して「フレーム毎に更新する」に設定し

ましょう。同様に、reinforcement ルーチンの clockRF の **文字列** にも$globalClock.getTime() と入力して「フレーム毎に更新する」に設定しましょう。変更が終わったら保存して、実行してください。確かに先ほどまで—-と表示されていた部分に経過時刻が表示されていますが、 図 9.7 に示すように小数点以下の桁数がやたらと多くて非常に見難くなってしまいました。小数点以下はせいぜい 1 桁もあれば充分でしょう。次の節では、小数点 1 桁までの表示を実現する方法を解説します。

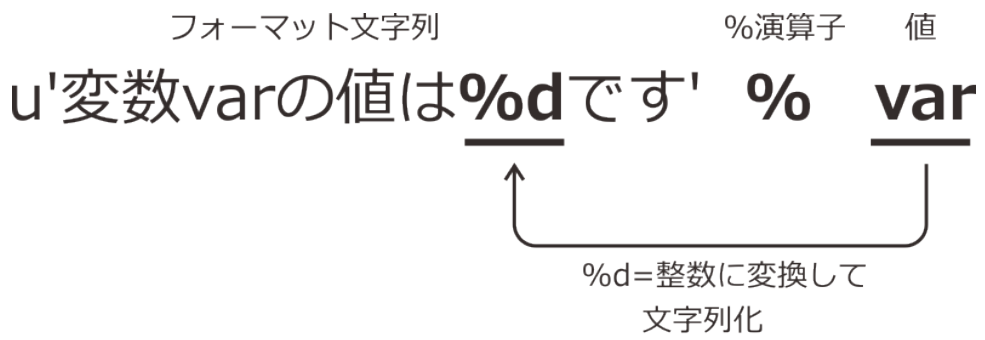
図 9.7　現在の経過時間をそのまま描画すると、小数点以下の桁数が多すぎて非常に見難くなります。

チェックリスト

- 実験が開始してからの経過時間を得るコードを書くことが出来る。

9.5 文字列に対する % 演算子を使って経過時間の表示を整えよう

前節で問題となった経過時間の表示ですが、これは Python が小数を文字列として表示する際に標準の動作としてあのように小数点以下の桁を表示してしまうことが原因で生じます。Python にはプログラマが小数点以下何桁程度を表示したいと考えているかを知る方法がありませんので、Python の標準動作が気に入らない場合はプログラマが明示的に表示方法を指定してやる必要があります。ここで重要な役割を果たすのが文字列に対する % 演算子です。% 演算子は左辺に文字列、右辺に数値や文字列等のデータまたはタプルと呼ばれるシーケンス型の一種をとり、右辺の値を左辺の文字列へ指定された書式で埋め込みます。左辺の文字列をフォーマット文字列と呼びます。C++ 言語など、Python 以外にも類似の機能を持っている言語はたくさんありますので、他のプログラミング言語をご存じのかたは「ああ、あれか」と思われるのではないかと思います。しかし、初めての方には何のことやらさっぱりわからないと思いますので、ていねいに見ていきましょう。他の言語をご存じの方も復習のつもりでご覧ください。



varの値	%演算子の適用結果
6	変数varの値は6です
-1	変数varの値は-1です
7.3	変数varの値は7です
0.9	変数varの値は0です
'foo'	(数値以外が指定されるとエラー)

図 9.8　% 演算子による数値の文字列への埋め込み。% 演算子の左辺の文字列に含まれている %d という部分に、% 演算子の右辺の数値を整数に変換して埋め込みます。%d の d は整数に変換するという「変換型」を指定する文字です。

表 9.1　フォーマット文字列における主な変換型

変換型	意味
d または i	符号付き 10 進数。小数点以下は切り捨てられます。
o	符号なし 8 進数
u	符号なし 10 進数。小数点以下は切り捨てられます。
x	符号なし 16 進数 (小文字) 例: 2000 → '7d0'
X	符号なし 16 進数 (大文字) 例: 2000 → '7D0'
e	指数表記の浮動小数点数 (小文字) 例: 2000 → ' 2.000000e+03'
E	指数表記の浮動小数点数 (大文字) 例: 2000 → ' 2.000000E+03'
f または F	10 進浮動小数点数
s	文字列
%	% という文字

図 9.8 は、変数 var に格納されている数値を整数に変換して文字列を生成する例です。% 演算子の左辺の文字列の中に含まれた %d という部分に、変数 var の値が整数として挿入されます。演算の結果は必ず文字列になる点に注意してください。% があちこちに出てきて非常に説明しにくいのですが、% 演算子の左辺に含まれている %d に注目してください。%d の d は「変換型」を示す文字列で、この文字列を変更することでさまざまな書式を指定することが出来ます。 表 9.1 に主な変換型を示します。だいたいお分かり頂けると思いますが、最後の % がわかりにくいかもしれません。変換型としての % は、変換後の文字列に % という文字そのものを含みたいときに使います。例えば「あなたの正答率は 60% でした」という文字列の 60 の部分に変数 correctRatio の値を埋め込みたい場合は、以下のように記述します。得られる文字列では %% が % に変換さ

れます。

```
u'あなたの正答率は%d%%でした' % correctRatio
```

変換後の文字列で、数値の桁数を指定したい場合は % と変換型の間にフラグ文字列と呼ばれる文字列を挿入します。主なフラグ文字列を 表 9.2 に示します。 表 9.2 では数値で例を示していますが、桁数の指定は文字列を埋め込む場合でも使えます。今回の場合、実験開始からの経過時間を小数点 1 桁まで表示したいので、小数を表示する変換型である f と、小数点以下の桁数を指定するフラグを組み合わせればうまくいきます。具体的には、以下の式で目的を達成できます。

```
'%.1f' % globalClock.getTime( )
```

表 9.2　主なフラグ文字列

フラグ	意味
正の整数	桁数を指定します。 例: u’ 値は %7d です’ % 17 → u’ 値は　　17 です’
0+ 正の整数	桁数を指定します。足りない桁は 0 で埋められます。 例: u’ 値は %07d です’ % 17 → u’ 値は 0000017 です’
負の整数	桁数を指定し、左詰めで文字列化します。 例: u’ 値は %-7d です’ % 17 → u’ 値は 17　　です’
小数	小数点の前の値は全体の、後の値は小数点以下の桁数を指定します。小数点前の値は省略することが出来ます。値は四捨五入されます。 例: u’ 値は %.3f です’ % 3.14159 → u’ 値は 3.142 です’ 例: u’ 値は %07.3f です’ % 3.14159 → u’ 値は 0003.14 です’

exp09vi.psyexp を開いて、この式を trial ルーチンの clockTrial と reinforcement ルーチンの clockRF の **文字列** に入力してください。もちろん先頭に$を付け忘れないようにしてください。変更したら保存して実行してみましょう。経過時間の表示が小数点 1 桁までになったはずです。

この % 演算子を使った文字列への値の埋め込みは、教示文や刺激文などを作る時にも便利です。今までは + 演算子と str( ) 関数を駆使して埋め込みを行っていましたが、埋め込む値が複数個ある場合は % 演算子を使っ

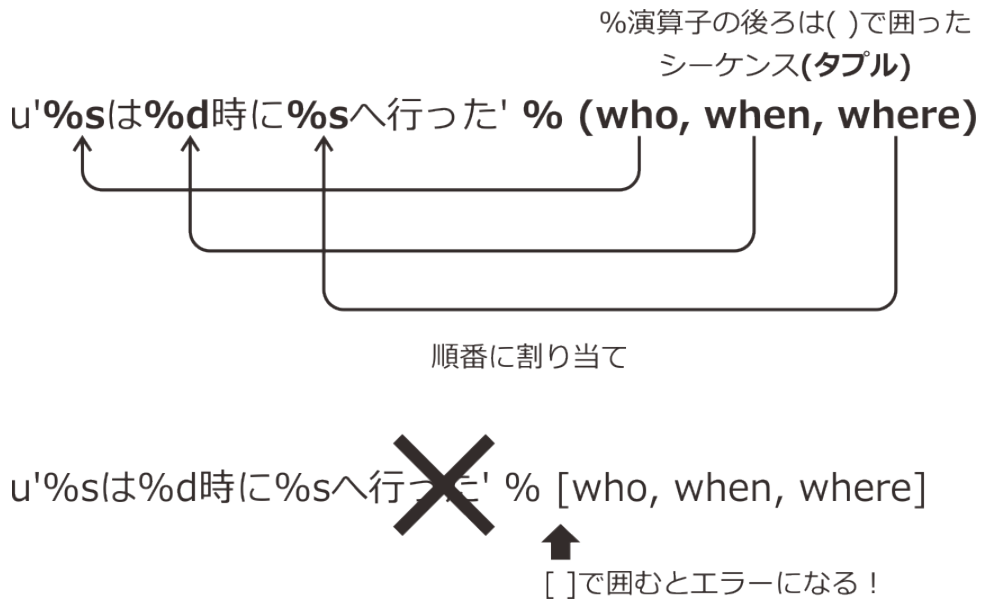


図 9.9　% 演算子による複数の値の埋め込み。フォーマット文字列に出現する変換型の個数と、% 演算子の右辺のシーケンスの要素数が一致している必要があります。右辺のシーケンスは ( ) で囲んだ「タプル」でなければいけません。[ ] で囲んだリストを右辺に置くとエラーになります。

た方が簡潔に記述できます。複数の値を埋め込む例を 図 9.9 に示します。% 演算子の右辺に埋め込みたい値を並べたシーケンスを置くのですが、このシーケンスは「タプル」と呼ばれる ( ) で囲まれたものでなければいけません。今までの章で出てきた「リスト」は [ ] で囲みましたが、% 演算子の右辺にリストを置くとエラーになります。タプルについて詳しくは **リストとタプル** をご覧ください。左辺のフォーマット文字列に含まれる変換型の個数と右辺のタプルの要素数は一致している必要があります。n 番目の変換型の位置へ、タプルの n 番目の要素が埋め込まれます。 図 9.9 の式では、変数 who に u’ 太郎’、when に 10、where に u’ ホームセンター’ が格納されているとすると、u’ 太郎は 10 時にホームセンターへ行った’ という文字列が得られます。ここで %d と対応している when に u’ 公園’ のように整数に変換できない値が格納されていた場合はエラーとなります。埋め込む値の個数が多い時には本当に % 演算子は便利なので、ぜひマスターしておいてください。

さて、これで 図 9.2 に示した実験手続のうち、FI/VI に関する実験が完成しました。ここからさらにステップアップする前に、exp09vi.psyexp を改造して FR/VR スケジュールの実験を作成しておきましょう。

チェックリスト

- ひとつの数値を 8 進数、10 進数、16 進数整数の書式で文字列に埋め込むことが出来る。
- ひとつの浮動小数点数を 10 進数表記と指数表記の書式で文字列に埋め込むことが出来る。
- 数値を文字列に埋め込む際に、指定した桁数の右寄せで埋め込むことが出来る。桁が足りない時に 0 で埋めることが出来る。
- 数値を文字列に埋め込む際に、指定した桁数の右寄せ埋めで込むことが出来る。桁が足りない時に空白文字で埋めることが出来る。
- 数値を文字列に埋め込む際に、指定した桁数の左寄せで埋め込むことが出来る。

- 浮動小数点数を文字列に埋め込む時に、整数部の桁数と小数点以下の桁数を指定することが出来る。
- 複数の値をひとつの文字列にそれぞれ書式を指定して埋め込むことが出来る。

9.6 FR/VR 実験を作成しよう

この節では、exp09vi.psyexp をベースにして、FR/VR スケジュールの実験を作成します。具体的には、exp09vi.psyexp を exp09vr.psyexp という別名で保存した後、exp09vr.psyexp に以下の変更を加えます。

- ratio というパラメータを定義する条件ファイル exp09fr.xlsx と exp09vr.xlsx を作成する。exp09fr.xlsx では ratio は 10 のみ、exp09vr.xlsx では ratio は 2、6、10、14、18 の 5 種類の値をとるようにする。
- trial ルーチンにおいて、ルーチン開始後に ratio の回数だけスペースキーを押したらルーチンを終了する。

これまでに解説してきたことだけで出来る変更なので、腕試しをしたい人は以下のヒントを見ずに作業してみてください。

さて、以下は「まだちょっと自力では難しいかな」という方向けのヒントです。条件ファイルの作成は、exp09fi.xlsx と exp09vi.xlsx を少し修正するだけです。exp09fi.xlsx のパラメータ名 interval を ratio に書き換えれば exp09fr.xlsx になります。exp09vi.xlsx のパラメータ名 interval を ratio に書き換えて、値を 2、6、10、14、18 の 5 種類に修正すれば exp09vr.xlsx になります。

続いて exp09vr.psyexp の変更です。trial ルーチンの終了条件を変更するだけですので、変更点は trial ルーチンの Code コンポーネント (codeTrial) です。trial ルーチンが開始されてから実験参加者がスペースキーを押した時刻が key_resp_Trial.rt にリストとして格納されているのですから、このリストの長さが変数 ratio 以上になればルーチンを終了すればよいのです。例えば以下のようなコードを書けばよいでしょう。

```
if len(key_resp_Trial.rt) >= ratio:
    continueRoutine = False
```

少なくとも筆者の経験上、Keyboard コンポーネントの **検出するキー $** にキー名がひとつしかない場合は一度に 1 回しかキー押しはカウントされないので、if 文の条件式は >= ではなく == でも恐らく正常に動作すると思います。しかし、一度に 2 回以上キー押しがカウントされてしまうことが万一起こったら == では永遠にルーチンが終了しなくなってしまいますので、上記のコードでは念のために >= としています。

以上の変更で FR/VR スケジュールへの対応が出来ました。次は再び exp09vi.psyexp をベースにして、実験開始後一定時間が経過したら実験を終了するように改造しましょう。

9.7 条件を満たしたら実験を強制終了するようにしよう

Builder は、基本的に条件ファイルで定められたすべてのパラメータを、ループで指定された回数だけ必ず繰り返すように設計されています。言い換えると、指定された試行数を実行するまで実験は終了しません。しか

し、心理学の実験の中には、「直近の 20 試行の正答率が 80% 以上になれば終了」とか、「実験開始から 20 分経過したら終了」といった具合に、試行数以外の条件で終了する実験もあります。このような実験は Builder との相性が悪いのですが、Code コンポーネントを使うと実現することが出来ます。

この節では、「試行数以外の条件で終了する実験」の例として、実験開始から指定された時間が経過したら実験を終了する実験を exp09vi.psyexp をベースに作成してみましょう。exp09vi.psyexp を exp09vi_time_limited_1.psyexp という別名で保存して、以後の作業は exp09vi_time_limited_1.psyexp に対しておこないます。

すでに前節までの作業で、グローバルクロックを用いて実験開始からの経過時間を得る方法は解説しました。先ほど「Code コンポーネントを使えば実現できる」と言ったのですから、exp09vi_time_limited_1.psyexp のどこかに

```
if globalClock.getTime( ) > 制限時間を保持している変数:
    実験を終了させる
```

というコードを書けばいいということは想像がつくと思います。実行中の実験を終了させると言えば思い出されるのが、実験設定ダイアログで [Enable Escape] をチェックしていれば ESC キーを押すことで実験を強制終了できる機能です (第 2 章)。この機能は、Builder の内部で psychopy.core.quit( ) という関数を呼び出すことで実現されています。この関数は名前通り、すべての PsychoPy のウィンドウを破棄して実験を終了します。Builder 内部ではこの関数を core.quit( ) という名前で呼び出すように import が行われているので、

```
if globalClock.getTime( ) > limitTime:
    core.quit( )
```

と書けば、実験を終了できます。なお、制限時間を保持している変数を limitTime としました。

残る問題は、このコードをどの時点で実行すればよいかということです。いろいろな候補が考えられますが、重要なポイントは「指定された時間が経過したら、課題遂行中であろうが直ちに終了してほしい」のか、「指定された時間が経過したら、現在遂行中の課題を通常通り終えてから終了してほしい」のかの違いです。前者であれば、Code コンポーネントの **フレーム毎** に上記のコードを記入して、フレーム毎に経過時間をチェックするべきです。後者であれば **Routine 終了時** か **Routine 開始時** にコードを記入するべきです。今回の実験では、interval に大きな値を指定した時には特に、指定時間が経過してから trial ルーチンが終了するまで時間がかかりますから、trial ルーチンの途中で直ちに終了するべきでしょう。reinforcement ルーチンはわずか 1 秒しかありませんし、強化子が途中で途切れるのは不適切だと思われますので、reinforcement ルーチンは途中で中断しないことにしましょう。そうすると、コードを挿入すべきなのは trial ルーチンの **フレーム毎** ということになります。

trial ルーチンにはすでに Code コンポーネントを配置済みですので、プロパティ設定ダイアログを開いて **フレーム毎** に以下のコードを追加してください。追加する位置は入力済みのコードの前でも後でもどちらでも構いません。

```
if globalClock.getTime( ) > limitTime:
    core.quit( )
```

あとは変数 limitTime の準備です。ここでは実験情報ダイアログから limitTime の値を指定出来るようにしま

しょう。実験設定ダイアログを開いて、**実験情報ダイアログ** に time limit (s) という項目を追加してください。そして、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再び開いて、 **実験開始時** に以下のコードを追加します。やはり入力済みのコードの前でも後でもどちらでも構いません。

```
limitTime = float(expInfo['time limit (s)'])
```

以上で変更は終了です。exp09vi_time_limited_1.psyexp を保存して実行してみてください。実験情報ダイアログに time limit (s) という項目がありますので、実験の終了時間を秒で指定しましょう。とりあえず動作確認のために 30 秒くらいの値を指定するのがお勧めです。条件ファイルと繰り返し回数も忘れずに指定して実験を実行しましょう。30 秒経過した時点で実験が自動的に終了するはずです。trial-by-trial 記録ファイルを開いて、強化まで進んだ試行についてはキー押しの記録が残っていることを確認してください。当然、中断された試行については保存処理なしにいきなり実験が終了していますので、キー押し記録は残りません。

中断された試行のキー押し記録も残したい場合は、psychopy.core.quit( ) 以外の方法で実験を終了させる必要があります。また、制限時間が経過したら実験全体を終了させてしまうのではなく、現在の課題を終了させて次の課題を実行したいという場合にも、psychopy.core.quit( ) は使えません。これらの場合にはどのような方法が有効でしょうか。勘のいい人は、psychopy.core.quit( ) の代わりに continueRoutine = False とすればいいじゃないかと思われるかもしれません。確かに、trial ルーチンと reinforcement ルーチンに Code コンポーネントを置いて、 **フレーム毎** で以下のコードを実行すれば、グローバルクロックの値が limitTime を超えた以後は trial ルーチンも reinforcement ルーチンも一瞬で終了してしまいます。この方法ならルーチン終了の処理もすべて通常通り行われますから中断された試行のキー押し記録も残りますし、終了させたい課題にだけこのコードを挿入しておけば、「現在の課題を終了させて次の課題を実行する」ことも可能です。

```
if globalClock.getTime( ) > limitTime:
    continueRoutine = False
```

この方法を自力で思いついた方は、前章までの内容をとてもよく理解しておられると思います。ですが、残念なことに、この方法にはひとつ問題があります。どのような問題が生じるのか、実際に試してみましょう。

exp09vi_time_limited_1.psyexp を開いて、今度は exp09vi_time_limited_2.psyexp という名前で保存してください。以後、exp09vi_time_limited_2.psyexp に対して作業をするものとします。保存したらさっそく exp09vi_time_limited_2.psyexp を開いて、trial ルーチンの codeTrial の **フレーム毎** の core.quit( ) を continueRoutine = False に書き換えましょう。書き換え後の **フレーム毎** は以下のようになっているはずです (ふたつの if 文の順番が逆でも構いません)。

```
if len(key_resp_Trial.rt) > 0 and key_resp_Trial.rt[-1] >= interval:
    continueRoutine = False
if globalClock.getTime( ) > limitTime:
    continueRoutine = False
```

続いて reinforcement 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。 **名前** は codeRF としておきましょう。codeRF の **フレーム毎** に以下のコードを追加します。

```
if globalClock.getTime() > limitTime:
    continueRoutine = False
```

これで作業は完了ですが、trial ルーチンと reinforcement ルーチンの実行を中断した後に別の課題を行えることを確認するために、trials ループの後にルーチンをひとつ置きましょう。以下の作業を行ってください。

- feedback ルーチン (作成する)
  - フローの trials ループの後ろ (つまりフローの最後) 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。
    - **名前** を textFB にする。
    - **終了** を「実行時間 (秒)」に設定し、5 と入力する。
    - **文字列** を$feedbackMessage にし、「繰り返し毎に更新」に設定する。
  - Code コンポーネントをひとつ配置し、以下のように設定する。
    - **名前** を codeFB にする。

さらに、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。

```
if globalClock.getTime() > limitTime:
    feedbackMessage = 'Time out'
else:
    feedbackMessage = 'Completed'
```

作業が終わったら、exp09vi_time_limited_2.psyexp を保存して、実行してください。条件ファイルに exp09fi.xlsx を指定して、nReps を 100、time limit (s) を 5 にしてみましょう。実行したら、スペースキーを押さずに制限時間の 5 秒が過ぎるのを待ってください。いかがでしょうか。10 秒経過した後、PC からピピピピーッと音がなってからスクリーンに Time out と表示されたのではないかと思います。何が起きたのかおわかりでしょうか？

グローバルクロックが limitTime を超えたら continueRoutine = False を実行するというコードを挿入することによって、確かに「trial ルーチンと reinforcement ルーチンを (nReps で指定された)100 回繰り返さずに」feedback ルーチンへ進むことが出来ました。しかし、いくら一瞬で終了するとはいえ trial ルーチンと reinforcement ルーチンは実行されているので、reinforcement ルーチンに含まれていた Sound コンポーネントの音が鳴り響いたのです。これでは「trial ルーチンと reinforcement ルーチンの実行を中断した」とは言えません。これらのルーチンを一瞬たりとも実行させずに feedback ルーチンへ進まなければなりません。

この難題を解決するためには、Builder の仕組みにさらに踏み込まなければなりません。Builder はルーチンを実行している時、実際にはどのようなコードを実行しているのでしょうか。そして、continueRoutine という変数の値を False にするとルーチンが終了するというのはいったいどういう仕組みによるものなのでしょうか。これらの点を理解していただくためには、if 文、for 文と並ぶ Python の重要構文である while 文を覚えていただく必要があります。

while 文とは、for 文と同様に繰り返し処理を行うための文です。for 文では、与えられたシーケンス型データに含まれる要素を先頭から取り出して順番に処理をしていきました。あらかじめ繰り返し処理の対象が決まっている時には for 文はとても便利なのですが、心理学実験でよくある「キーが押されるまでスクリーンに刺激を描画し続ける」といった処理では、いつまで描画を繰り返せばいいのか実際に実行してみるまでわかりませ

ん。このような「ある条件が満たされるまで同じ処理を繰り返す」という処理を行いたいときに用いるのが while 文です。

図 9.10 に while 文の概要を示します。while 文は条件式と組み合わせて使用し、条件式が True である間、処理を繰り返します。繰り返す範囲は for 文や if 文と同様に字下げで示します。注意しないといけないのは、while 文と組み合わせる条件式に誤って絶対に False にならない式を書いてしまうと、Python インタプリタ自体が正常に動作している限り永遠に処理が終了しないという点です。ただし、for 文と同様に break と continue を使うことが出来ますので、繰り返し中に break を実行すれば条件式に関わらず繰り返しを中断することは出来ます。なお、while 文には if 文のように else を伴わせることができますが、その時の動作については *while* **文に伴う** *else* を参照してください。

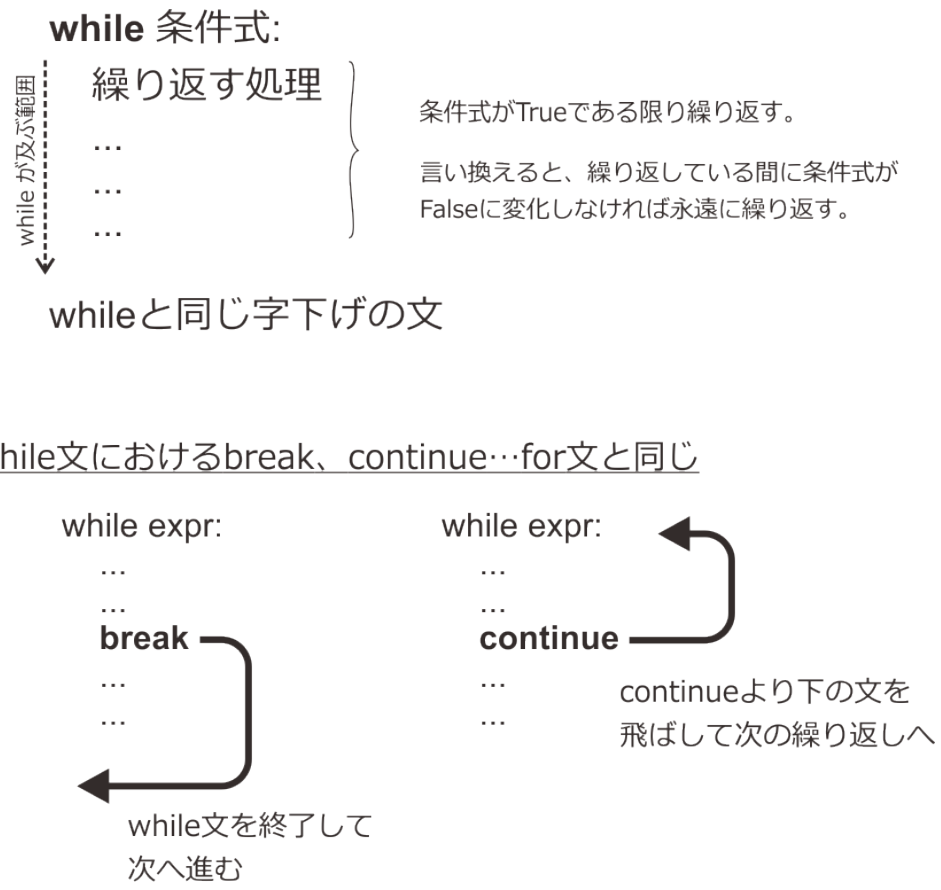


図 9.10　while 文。条件式が True である限り処理を繰り返します。for 文と同様に break や continue を使うことが出来ます。

図 9.11 は、Builder が実験のフローをどのように Python のコードへ変換するかを示しています。フローにおけるループは for 文に変換されます。条件ファイルや **Loop の種類** 、 **繰り返し回数 $** などに基づいて各試行で用いられるパラメータ一覧のリストが作成され、for 文へ渡されます。この for 文を実行することで、パラメータを変更しながら指定された回数の繰り返しが実行されます。一方、各ルーチンは while 文に変換されます。 図 9.11 は Code コンポーネントの解説も含んでいて複雑になっていますので、以下に while 文の部分を抜粋して示します。

```
continueRoutine = True
while continueRoutine and ルーチン継続に関する条件:
    各コンポーネントのフレーム毎の処理
```

ルーチンが開始される直前に、continueRoutine という変数に True が代入されます。そして while 文によってルーチン内の各コンポーネントのフレーム毎の処理を繰り返します。while 文の条件式には、continueRoutine とその他の条件の論理積 (and) が渡されます。その他の条件というのは、例えばルーチンが 5.0 秒で終了するように各コンポーネントの **終了** が設定されていた場合には「ルーチン開始からまだ 5.0 秒経過していない」という条件が入ります。continueRoutine が False の場合、他にどのような条件が指定されていても while 文の条件式は False になるので、continueRoutine = False を実行すると while 文の繰り返しが終了します。これが今まで呪文のように continueRoutine = False と書いていた時に実際に生じていたことなのです。

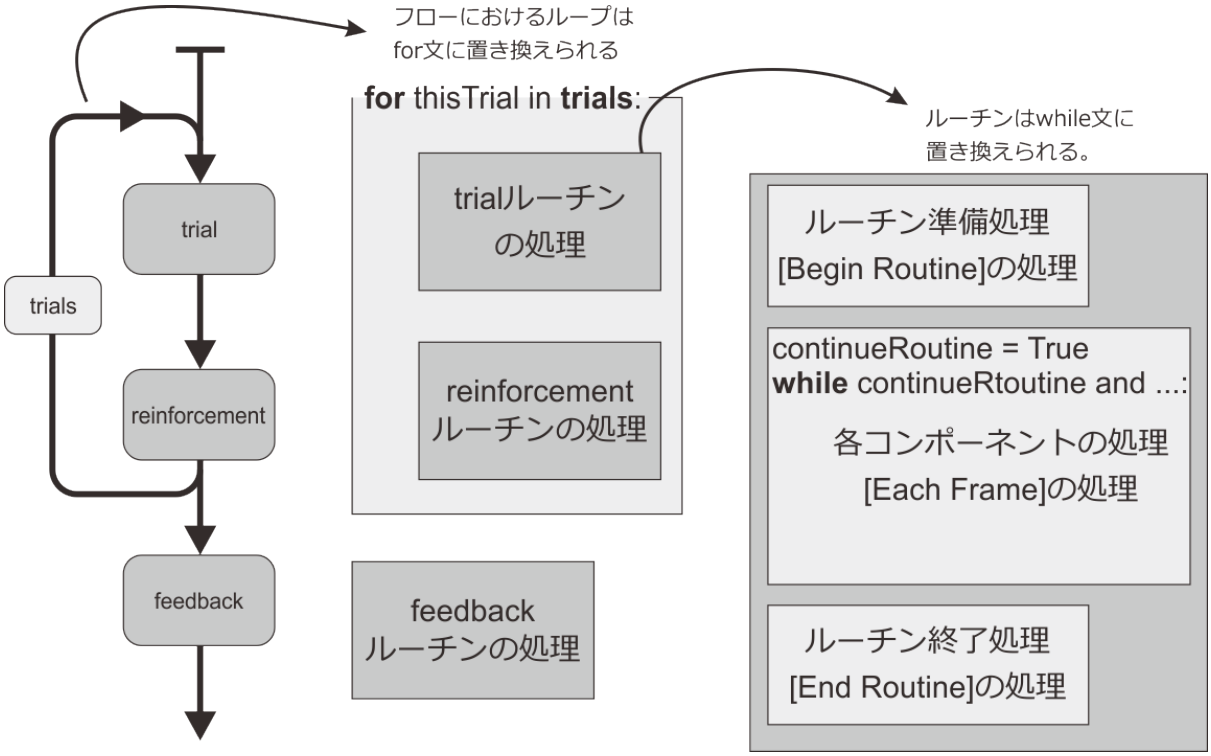


図 9.11　Builder によるフローから Python コードへの変換。フローにおけるループは for 文に、ルーチンのフレーム毎の処理は while 文に置き換えられます。

以上を踏まえて改めて 図 9.11 をご覧ください。 図 9.11 では、Code コンポーネントで **Routine 開始時** 、 **フレーム毎** 、 **Routine 終了時** にコードを記入した時にどの位置にコードが挿入されるかを示しています。これをご覧になったら、continueRoutine = False を **フレーム毎** に書かなければ while 文を終了させることが出来ないのがおわかりいただけるかと思います。

さて、なぜこんな Builder の裏側の世界まで足を踏み入れているのかと言えば、何とかしてルーチンを一瞬たりとも実行させずに中断させたいのでした。continueRoutine には while 文が実行される直前に True が代入されます。ですから Code コンポーネントの **Routine 開始時** に continueRoutine = False と書いても実行直前に True に書き換えられてしまい、ルーチン (=while 文) の実行を止めることが出来ません。一度でも while 文が実行されると Sound コンポーネントが実行されてしまうので、いくら **フレーム毎** に continueRoutine = False

と書いても音が鳴ってしまいます。これが exp09vi_time_limited_2.psyexp で生じていた問題です。では、どうすればいいでしょうか？ ここで思い出していただきたいのが break です。break が実行されると、break 以降に処理があろうと直ちに繰り返しが中断されるのでした。ということは、ルーチンが実行された後、どのコンポーネントの処理よりも先に break を実行すれば、実質的にルーチンを実行しなかったのと同じ結果になるはずです。さっそく試してみましょう。

exp09vi_time_limited_2.psyexp を、exp09vi_time_limited_3.psyexp という別名で保存してください。以下の作業は exp09vi_time_limited_3.psyexp に対して行うものとします。別名で保存したら、trial ルーチンを開いて codeTrial の **フレーム毎** を確認してください。globalClock.getTime( ) > limitTime だったら continueRoutine = False するというコードが入力されているはずですが、この continueRoutine = False を break に書き換えてください。書き換えた後の **フレーム毎** は以下のようになっているはずです (ふたつの if 文の順番が逆でも構いません)。

```
if len(key_resp_Trial.rt) > 0 and key_resp_Trial.rt[-1] >= interval:
    continueRoutine = False
if globalClock.getTime( ) > limitTime:
    break
```

同様に、reinforcement ルーチンの codeRF の **フレーム毎** も以下のように break に書き換えてください。

```
if globalClock.getTime() > limitTime:
    break
```

そして、次の作業が重要です。これらの修正したコードが、同一ルーチン内の他のコンポーネントの処理が行われる前に実行されるようにする必要があります。そのためには、codeTrial や codeRF がルーチンペインの一番上に並んでいる必要があります (図 9.12)。trial ルーチンと reinforcement ルーチンを開いて、codeTrial と codeRF がそれぞれ一番上になるように並び替えましょう。

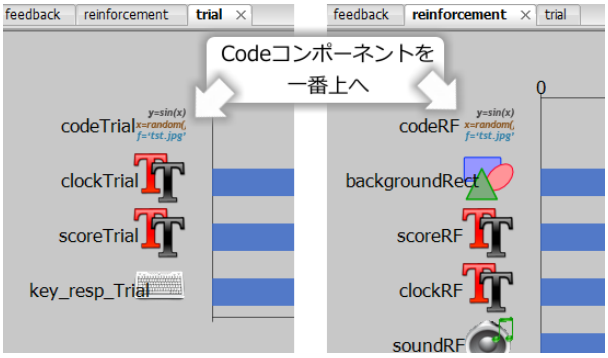


図 9.12　trial ルーチンと reinforcement ルーチンの Code コンポーネントを一番上へ配置してください。

作業が終わったら、exp09vi_time_limited_3.psyexp を保存して実行しましょう。先ほどと同様に、条件ファイルに exp09fi.xlsx を指定して、nReps を 100 にしてください。時間内に得点が得られた試行のデータも欲しいので、time limit (s) は 30 にしてください。実行したら、スペースキーを押して得点を獲得しながら 30 秒経過するのを待ちましょう。30 秒経過したら実験が中断されてスクリーンに Time out と表示されますが、今度は exp09vi_time_limited_2.psyexp の時のように音が鳴らなかったはずです。

psychopy.core.quit( ) で終了した場合と異なり、作成された trial-by-trial 記録ファイルには、 図 9.13 のように中断された試行もすべて記録されています。break で中断された試行ではキー押しが記録されませんので、key_resp_Trial.keys と key_resp_Trial.rt はいずれも None と出力されています。かなり苦戦しましたが、これでなんとか目的を達成することが出来ました。

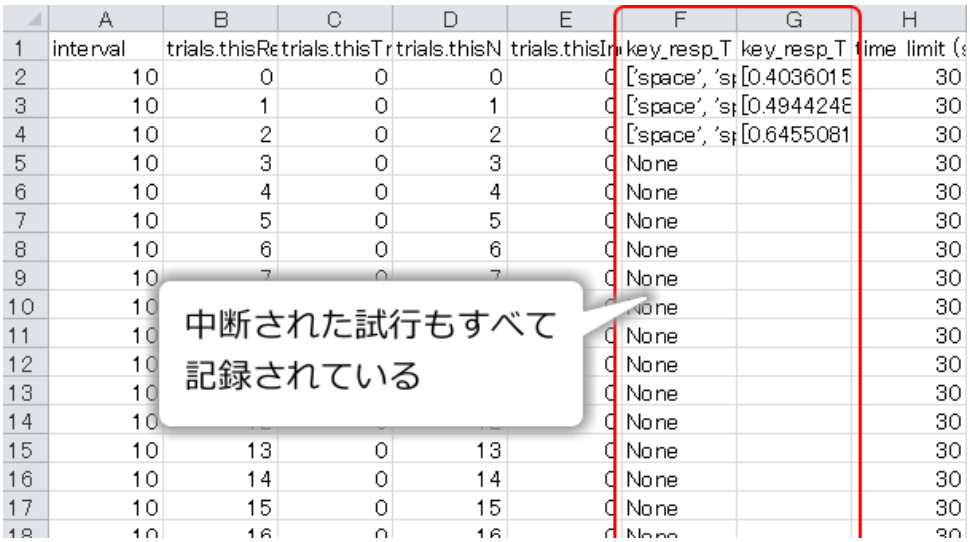


図 9.13　psychopy.core.quit( ) で中断した場合と異なり、ルーチンを break で中断した場合は中断された試行もすべて記録されています。

なお、皆さんの中には、exp09vi_time_limited_3.psyexp を実行した時に、30 秒経過してから Time out と表示されるまでに少し「30.0」という表示のまま PC が停止したかのような動作をしたことが気になった方がいるかも知れません。これは、Builder が残りの全て実際に開始して、即 break して、結果を trial-by-trial 記録ファイルに書きこむという作業を繰り返しているために生じる現象です。break された試行もすべて記録に残すべきか否かはどちらが良いか一概には言えませんが、この待ち時間が問題になる実験ももしかしたらあるかも知れません。この節のテクニックを応用することで待ち時間をなくすことも可能ですが、これは練習問題としておきましょう。

チェックリスト

- 条件式が True である間処理を繰り返す Python のコードを書くことが出来る。
- Code コンポーネントを用いて、ある条件を満たしたときに実行が中断される実験を作ることが出来る。
- Code コンポーネントを用いて、ある条件を満たしたときに実行がスキップされるルーチンを作ることが出来る。

9.8 繰り返し回数を変更して並立スケジュールを実現しよう

Builder に対する不満として非常によく聞くのが、フローを分岐させることが出来ないというものです。例えば、スクリーン上に複数の選択肢が提示されて、実験参加者がひとつを選択すると、それに応じた課題が行わ

れるといった実験を現在の Builder は想定していません。しかし、Code コンポーネントを利用するとフロー上では分岐できなくても動作上は分岐する実験を作成することが可能です。

具体的な例題が無いと話がしづらいので、 図 9.14 のような並列スケジュールの実験を考えてみましょう。exp09vi.psyexp の前にスクリーンをひとつ挿入して、そこで「カーソルキーの左右を押して課題を選択してください」と教示します。参加者は適切なタイミングでスペースキーを押すと得点が得られること、出来るだけ速く 50 点を得るように努力することだけが告げられていて、左右の課題がそれぞれどのようなスケジュールであるのか知らされていません。左右いずれかのキーを押すと、押したキーに応じたスケジュールで課題が始まります。

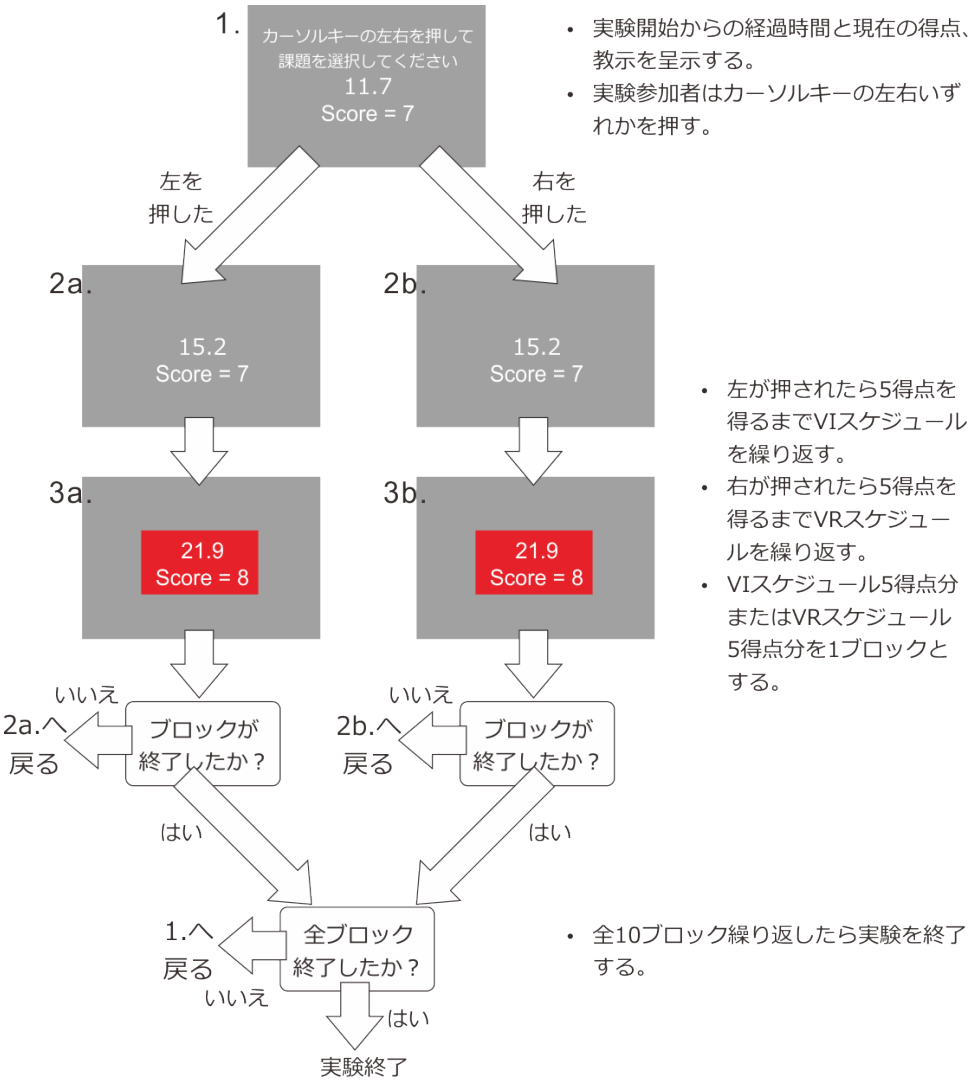


図 9.14　並立スケジュール。1. のスクリーンで実験参加者がカーソルキーの左右どちらを押すかによって次のブロックの課題が変化します。

並列スケジュールの実験の作成において、

- 二つの課題がどちらも VI
- 左キーの場合は強化時間が 2 秒、11 秒、20 秒、29 秒、38 秒の 5 種類

- 右キーの場合は強化時間が 12 秒、16 秒、20 秒、24 秒、28 秒の 5 種類

といった具合に、左右の課題の違いがパラメータの違いだけであれば、実現はあまり難しくありません。ルーチンとループは共通のものを利用して、条件ファイルを押されたキーに応じて変更すれば実現できます。これは練習問題にしておきます。厄介なのは

- 左キーの場合は VI で強化時間が 2 秒、6 秒、10 秒、14 秒、18 秒の 5 種類
- 右キーの場合は VR で強化回数が 12 回、16 回、20 回、24 回、28 回の 5 種類

といった具合に、押されたキーによって実験内容が異なる場合です。前節の break を駆使する方法でも実現可能ですが、この節ではループの実行そのものを省略する方法を紹介します。

この節の実験では、前節までのような時間制限がないので、exp09vi.psyexp をベースに改造するのがよいでしょう。exp09vi.psyexp を開いて、exp09concurrent.psyexp という別名で保存してください。保存したら、exp09concurrent.psyexp に対して以下の作業を行ってください。かなり複雑なフローになりますので、完成後のフローを 図 9.15 に示しておきます。

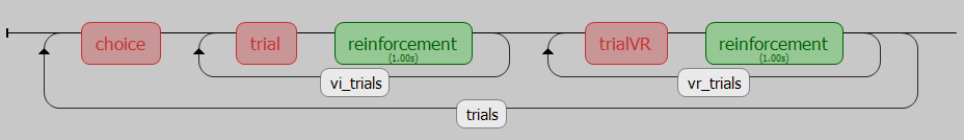


図 9.15　exp09concurrent.psyexp のフロー。

- 実験設定ダイアログ
  - **実験情報ダイアログ** から nReps と condition を削除する。
- trials ループ (vi_trials ループに名称変更)
  - **名前** を vi_trials にする。
  - **繰り返し回数 $** を nRepsVI にする。
  - **繰り返し条件** を$condition にする。
- trialVR ルーチン (作成する)
  - vi_trials ループの直後に挿入する。
  - Text コンポーネントをふたつ配置して、それぞれ **名前** を scoreTrialVR、clockTrialVR にする。両方とも **終了** を空白にする。
    - scoreTrialVR の **位置 [x, y] $** を [0.0, -0.2] にする。 **文字列** に$'Score:'+str(score) と入力し、「繰り返し毎に更新」に設定する。
    - clockTrialVR の **文字列** に$'%.1f' % globalClock.getTime( ) と入力し、「フレーム毎に更新する」に設定する。
  - 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。

        * **名前** を key_resp_TrialVR にする。
        * **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 $** に'space' と入力する。
        * **記録** を「全てのキー」にする。
    - Code コンポーネントをひとつ配置し、 **名前** を codeTrialVR にする。
        * **Routine 終了時** に score+=1 と入力する。
- reinforcement ルーチン
    - trialVR ルーチンの直後に挿入する。フロー内に 2 個の reinforcement ルーチンが配置されることになる。
- vr_trials ループ (作成する)
    - trialVR ルーチンと、その後ろにある reinforcement ルーチンを繰り返すように挿入する。
    - **繰り返し回数 $** を nRepsVR に、 **繰り返し条件** を$condition にする。
- choice ルーチン (作成する)
    - vi_trials ループの前に挿入する。
    - Text コンポーネントを 3 つ配置して、 **名前** をそれぞれ textInstruction、scoreChoice、clockChoice とする。すべて **終了** を空白にする。以下のように設定する。
        * textInstruction の **位置 [x, y] $** を [0.0, 0.2] にする。 **文字列** に「カーソルキーの左右を押して課題を選択してください」と入力する。
        * scoreChoice の **位置 [x, y] $** を [0.0, -0.2] にする。 **文字列** に$'Score:'+str(score) と入力し、「繰り返し毎に更新」に設定する。
        * clockChoice の **文字列** に$'%.1f' % globalClock.getTime( ) と入力し、「フレーム毎に更新する」に設定する。
    - 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。
        * **名前** を key_resp_Choice にする。
        * **終了** を空白にする。
        * **検出するキー $** を'left','right' にする。
        * **記録** を「なし」にする。
    - Code コンポーネントを配置し、 **名前** を codeChoice にしておく。現時点ではコードは入力しない。
- blocks ループ (作成する)

  - フロー全体を繰り返すように挿入する。すなわち、choice ルーチンの前から vr_trials ループの終わりまでを繰り替えす。
  - **名前** を blocks にする。
  - **繰り返し回数 $** を 10 にする。

- exp09concurrentVI.xlsx(条件ファイル)
  - exp09vi.xlsx をコピーして、interval の値を 2, 6, 10, 14, 18 にする。
- exp09concurrentVR.xlsx(条件ファイル)
  - exp09vr.xlsx をコピーして、ratio の値を 12, 16, 20, 24, 28 にする。

さらに、trialVR ルーチンの codeTrialVR の **フレーム毎** に以下のコードを入力してください。

```
if len(key_resp_TrialVR.rt) >= ratio:  continueRoutine = False
```

準備はよろしいでしょうか。それでは最後の仕上げ、Choice ルーチンの Code コンポーネント (codeChoice) にコードを入力しましょう。codeChoice では、以下の処理を行います。

- key_resp_Choice で押されたキーが'left' であれば、以下の変数を設定する。
  - nRepsVI = 1
  - nRepsVR = 0
  - condition = 'exp09concurrentVI.xlsx'
- key_resp_Choice で押されたキーが'right' であれば、以下の変数を設定する。
  - nRepsVI = 0
  - nRepsVR = 1
  - condition = 'exp09concurrentVR.xlsx'

もうすでに何をしようとしているかお分かり頂けたと思います。VI スケジュールを行うループ vi_trials の繰り返し回数は、nRespVI という変数で決まります。nRepsVI に 0 を代入すれば、繰り返し回数が 0 回となって一度も VI スケジュールの試行が行われないというわけです。同様に、nRepsVR に 0 を代入すれば、VR スケジュールの試行は行われません。押されたキーに従って変数に異なる値を設定するのはすでに 6 章で、 **記録** が「なし」の時に押されたキーを取得する方法は 7 章で学びましたから、新しい作業は何もありません。以下のコードを codeChoice の **Routine 終了時** に入力してください。

```
if 'left' in theseKeys:
    nRepsVI = 1
    nRepsVR = 0
    condition = 'exp09concurrentVI.xlsx'
elif 'right' in theseKeys:
    nRepsVI = 0
```

```
nRepsVR = 1
condition = 'exp09concurrentVR.xlsx'
```

入力したら、exp09concurrent.psyexp を保存して実行してみましょう。選択画面が出てきたら教示に従ってカーソルキーの左右いずれかを押し、左を押せば VI スケジュール、右を押せば VR スケジュールの課題が 5 得点分ずつ実行されることを確認してください。実行後に作成される trial-by-trial 記録ファイルの例を 図 9.16 に示します。繰り返し回数を 0 に設定することによって飛ばされたループに該当するセルは、空白になります。どのような順番で課題を選択したか、それぞれの課題においてキーは何回押したか、いつ押したかといった情報を全て trial-by-trial 記録ファイルから読み取ることが出来ます。

さて、ずいぶん解説が長くなってしまいましたので、この章もそろそろおしまいにしたいと思います。この章で紹介したテクニックを用いると、現状の Builder でもフローが分岐する実験を作成することが出来ます。次章はいよいよ最終章です。次章では無作為化に関する話題を取り上げます。

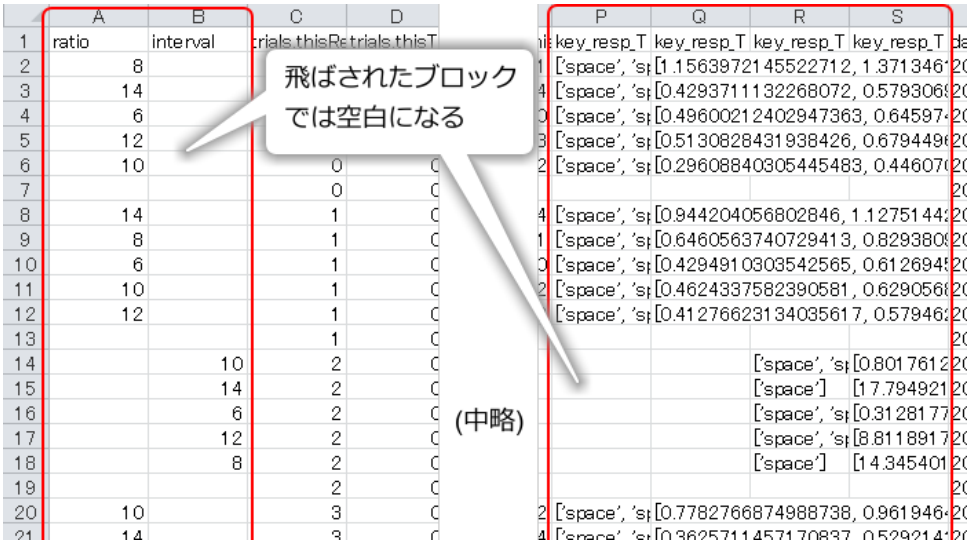


図 9.16　exp09concurrent.psyexp を実行した時に作成される trial-by-trial 記録ファイルの例。繰り返し回数を 0 に設定することによって飛ばされたループに該当するパラメータやキー押し記録のセルは、空白になります。

チェックリスト

- ある条件に当てはまる時にループを実行しない実験を作成することが出来る。
- ループを実行しない実験を作成した時の trial-by-trial 記録ファイルから、ループを実行しなかった時の記録を判別できる。

9.9 練習問題：さまざまなフロー制御をマスターしよう

この章の練習問題は以下の 3 問です。第 2 問と第 3 問は本文中で「練習問題」としたものです。

- 問 1：exp09vi.psyexp をベースとして、フローの先頭に教示画面を挿入してください。そして、trial

ルーチンや reinforcement ルーチンで表示される経過時間を実験開始からの時間 (globalClock が示す時間) ではなく、教示画面を終了して実際に強化スケジュールが開始してからの時間を表示してください。

  - 教示文は各自に任せます。
  - 教示画面でスペースキーを押したら実験が始まるようにしてください。

- 問 2：exp09vi.psyexp をベースとして、 図 9.14 の並列スケジュールの実験を以下の条件で作成してください。
  - 左キーが押された場合は強化時間が 2 秒、11 秒、20 秒、29 秒、38 秒の 5 種類の VI スケジュール
  - 右キーが押された場合は強化時間が 12 秒、16 秒、20 秒、24 秒、28 秒の 5 種類の VI スケジュール
  - 左キーと右キーのスケジュールの実行には同一のループおよびルーチンを使用し、条件ファイルの切り替えによってスケジュールの違いに対応する。
- 問 3：exp09vi_time_limited_3.psyexp をベースとして、time limit (s) で指定された制限時間を過ぎてしまったときに、スキップされた試行のデータが trial-by-trial 記録ファイルに出力されないようにする。ただし、制限時間によって中断された試行の、中断直前までのキー押しは trial-by-trial 記録ファイルに出力されているようにすること。

9.10 この章のトピックス

9.10.1 Movie コンポーネントのバックエンド

バックエンドにはいろいろな意味がありますが、動画再生に利用するライブラリのことを指しています。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。

PsychoPy が動画再生をサポートした当初は、avbin というライブラリが利用されていました。ところがこの avbin は実験の実行時にうまく読み込めないことがあるなどのトラブルが多く、動画再生機能を全く利用していない実験にまで 図 9.17 のようなエラーメッセージが表示されることがよくありました。動画再生を利用していないのであれば実験は正常に動作するのでこのエラーは無視して良いのですが、そのことを知らなくて「なぜエラーメッセージが出ているんだろう、実験に何か問題があるのだろうか」と頭を悩ませる人が後を絶ちませんでした。

avbin を利用した動画刺激クラスは psyhcopy.visual.MovieStim という名称ですが、MovieStim の問題を解消するために新たに OpenCV というライブラリを通じて VLC media player というソフトウェアの動画再生機能を利用する新しい動画刺激クラス MovieStim2 が導入されました。確かに MovieStim に比べていろいろと改善された点が多いのですが、VLC media player をインストールしていないと利用でいないという問題を抱え込んでしましました。厄介なことに、PsychoPy は VLC media player を見つけられない時に「MovieStim2 を import 出来ないから利用できない」というメッセージを表示するため、VLC media player をイントールしないといけないということが非常にわかりにくいのです。

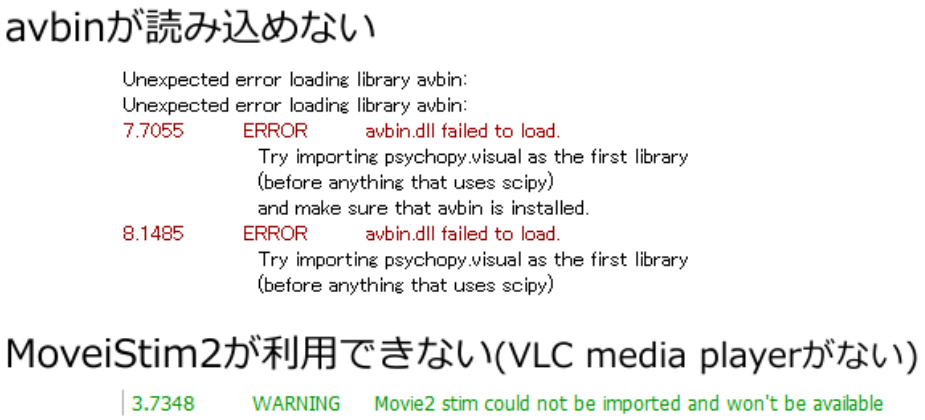


図 9.17　動画再生バックエンドに関するエラーメッセージ

以上を踏まえたうえで、Movie コンポーネントの **バックエンド** について解説します。 **バックエンド** は「avbin」と「opencv」を選ぶことができますが、「avbin」を選ぶと MovieStim がバックエンドとして使用されます。「opencv」ならば MovieStim2 が使用されます。VLC media player のことを知らずに「opencv」を選ぶと「MovieStim2 が利用できない」とエラーメッセージが出てきて、その解消方法が「VLC media player をインストールする」だなんて普通わかるはずがないと筆者は思うのですが、仕方がありません。利用する際はご注意ください。

9.10.2 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、一応その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。タプルはリストと非常によく似ているのですが、なぜ「リスト」と「タプル」という類似したデータ型が用意されているかを第 5 章の時点で解説すると却って混乱すると考えたからです。

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出来ます。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。

なお、リストをはじめとする各種シーケンス型データをタプルに変換したいときには tuple( ) という関数を用います。以下のコードを実行すると、変数 data_tupel には (1, -7, 'psychology') というタプルが格納されます。

```
data = [1, -7, 'psychology']
data_tuple = tuple(data)
```

タプルをリストに変換したいときには list( ) という関数を用います。これらの関数を覚えておくと役に立つかもしれません。

9.10.3 while 文に伴う else

Python では while 文に else を伴わせることができます。if 文に伴う else は他の多くのプログラミング言語で使用することができますので、他言語でのプログラミング経験がある人はすぐにわかったと思うのですが、while 文に伴う else は C 言語などには無いので戸惑われるかたもいるかもしれません。

if 文の else は、if 文の条件式が False だった時に行う処理を指定するためのものでした。while 文の else も同様に、while 文の条件式が False だった時に行う処理を指定します。以下の while 文を考えてみましょう。

```
x = y = 0
while x<10:
    x += 1
else:
    y = x
```

最初に x と y に 0 が代入され、while 文で x+=1 を繰り返します。x=10 になった時点で while 文の条件式が False になるので、else で記述されている y = x が実行されます。結果として、x と y の値は 10 となります。続いて以下の式を考えてみましょう。while 文で繰り返す処理の中に「x の値が 5 であれば break する」という処理を追加しています。

```
x = y = 0
while x<10:
    x += 1
    if x == 5:
        break
else:
    y = x
```

この場合、while 文を繰り返しているうちに x の値が 5 になって break が実行されます。while 文の条件式である x<10 が False になったわけではないので、else の処理は実行されません。結果として、x の値は 5 になり、y の値は 0 のままになります。

第 10 章

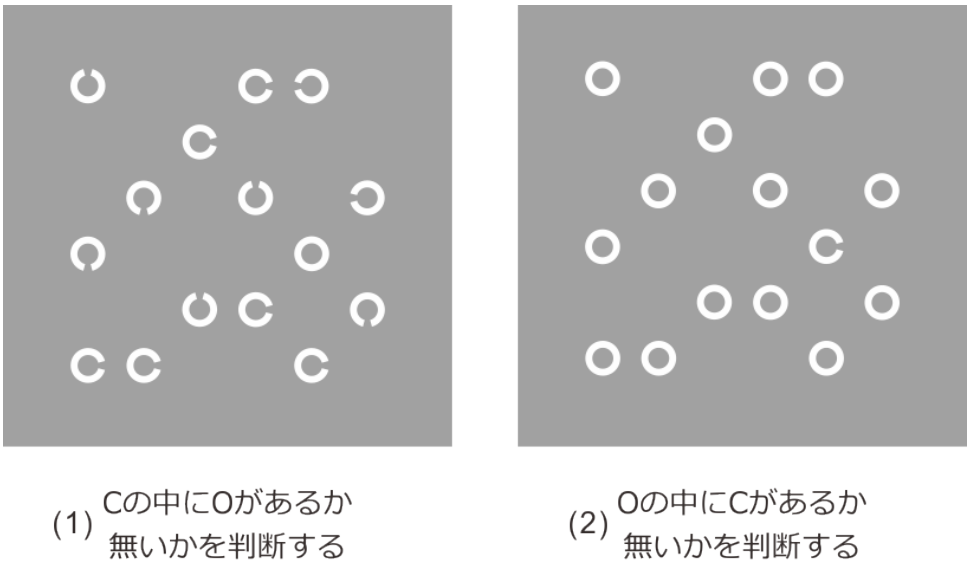
無作為化しよう―視覚探索

10.1 この章の実験の概要

いよいよ本書も最後の章です。前章では、Builder に対する不満として「条件分岐が出来ない」という点を挙げて、Code コンポーネントで解決を試みました。この章では、「十数個の視覚刺激を描画するのが面倒」、「すべてのパラメータを明示的に条件ファイルで与えるのが面倒」という問題を取り上げたいと思います。

この章の例題として挙げる実験は、視覚探索課題です。 図 10.1 に視覚探索課題の例を示します。 図 10.1 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、出来るだけ速く正確に、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。容易に予想出来る事ですが、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 10.1 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 10.1 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 10.2 に具体的な手続きを示します。実験を実行すると、準備が出来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出来るだけ速く、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



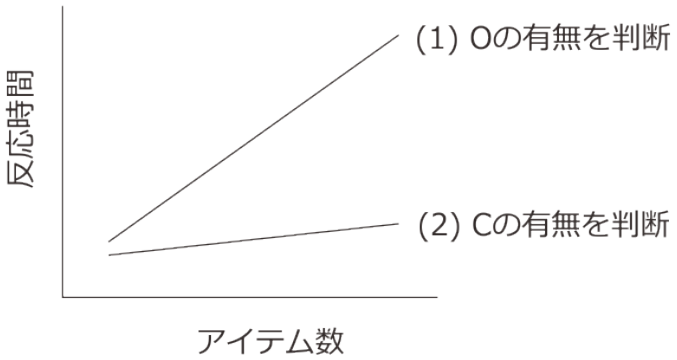


図 10.1　視覚探索の非対称性。スクリーン上に提示されているアイテム数が多いほど判断に時間がかかりますが、C のなかから O を探すより、O の中から C を探す方がアイテム数増加に伴って反応時間が急激に増加します。

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 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 10.3 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。

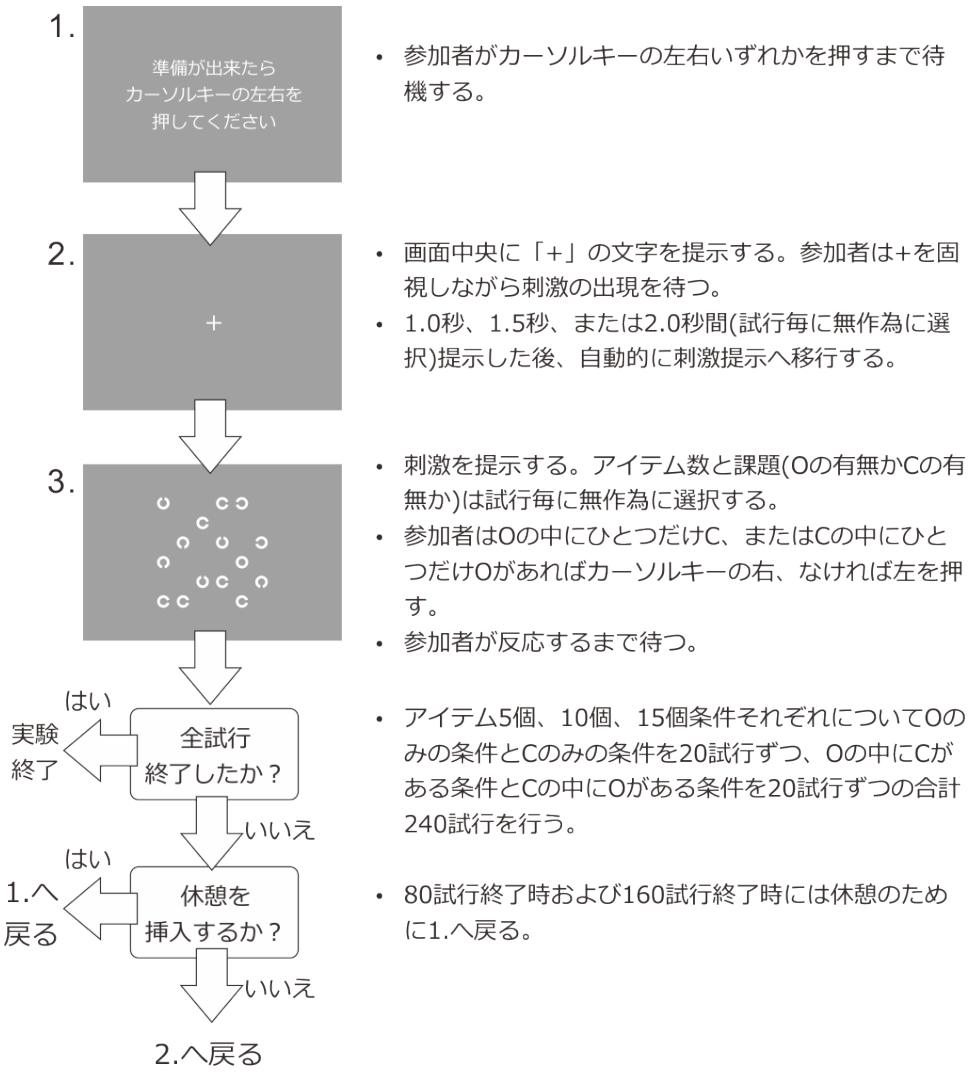


図 10.2　実験の手続き。

以上が実験の概要です。Builder が苦手とするポイント、出来る事なら 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、敢えて箇条書きにすると以下の 4 点が挙げられます。

1. 独立に位置や形状が変化するアイテムが最大 15 個も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、画面上に刺激が大量に存在すると、ルーチン上に必要な個数のコンポーネントをひとつひとつ並べてパラメータを設定していかなければなりません。とても面倒な作業です。試行毎にアイテム数が異なる点も、真面目に作成しようとしたら第 9 章のテクニックを用いてアイテム数 5 個のルーチン、10 個のルーチン、15 個のルーチンを使い分けなければいけません。大変な手間です。もっとも「真面目に作

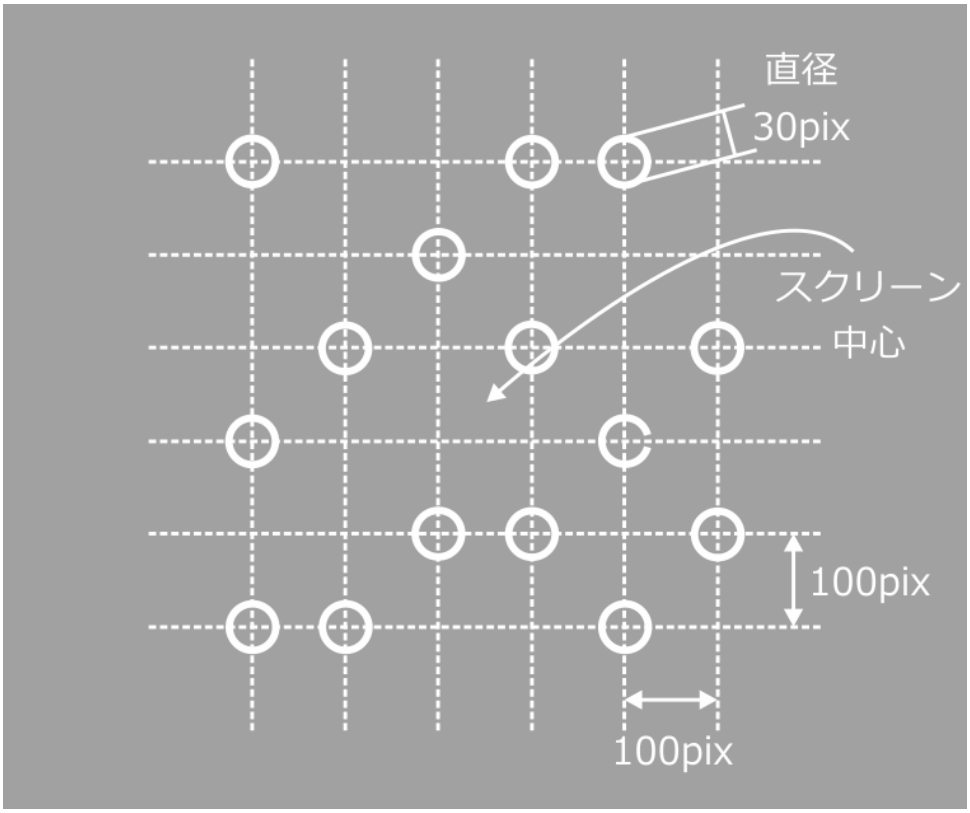


図 10.3　刺激の配置。アイテムの位置は仮想的な 6 × 6 のグリッド上から試行毎に無作為に選択されます。

成しようとしたら」と断り書きを入れるということは抜け道があるのですが、それはこの後の解説で触れます。

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出来ません。

なぜ今回の実験では、今までの章と同じ考え方ではうまくいかないのでしょうか。今までの章では、「無作為」という言葉を使う時に、それは「順番が無作為」というだけでどのパラメータ（の組み合わせ）を何試行行うかが決まっていました。ところが、今回の実験では、パラメータそのものを試行毎に無作為に決めようというのです。結果として全試行を通じて 3 種類の注視時間が選ばれた回数は均等にならないでしょうが、本当に「無作為に」有限回数の選択をしたのであれば、むしろ回数が均等にならないことがある方が普通です。この種の「無作為さ」を実現する方法は Builder には用意されていませんので、Code コンポーネントの力を借りる必要があります。

この章の課題を Builder で実現することの厄介さを何となく感じていただけましたでしょうか。筆者の個人的な考えでは、Builder に慣れていない初級から中級の方がこの章の実験を Builder で作成するのであれば「すべての刺激を画像としてあらかじめ用意する」という方法を検討すべきだと思います。 図 10.2 および 図 10.3 に基づいて数百から数千枚程度の刺激画像ファイルを作成しておいて、その中から 240 枚の画像を各条件の試行数を満たすように無作為に抽出した条件ファイルを複数個準備しておいて、参加者毎にことなる条件ファイルを使用して実験を行うのです。しかし、本書は Builder のさまざまなテクニックを紹介することが目的なので、Builder 上で刺激を作成して実験を行う方法を考えたいと思います。

10.2 実験の作成

実験の作成に入りましょう。まず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。Builder で新規に実験を作成して以下の作業を行い、exp10proto.psyexp という名前で保存してください。

- 実験設定ダイアログ
  - 「xlsx 形式のデータを保存」をチェックする。
  - **単位** を pix にする。
- trial ルーチン
  - 最初から配置されている Static コンポーネントの **名前** が ISI、 **開始** と **終了** がそれぞれ 0.0 と 0.5 になっていることを確認する (いずれも初期値)。
  - Code コンポーネントをひとつ配置して、 **名前** を code_trial 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。
    - **名前** を fixpoint にする。
    - **終了** を delay にする。
    - **文字の高さ $** を 24 にする。
    - **文字列** に + と入力する。
  - 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。
    - **名前** を key_resp_trial にする。
    - **開始** を delay に、 **終了** を空白にする。
    - **Routine を終了** がチェックされていることを確認する。
    - **検出するキー $** を'right', 'left' とする。
    - **正答を記録** をチェックし、 **正答** に$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。

        * **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、「trial の ISI の間に更新」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。
        * **位置 [x, y] $** に pos[0] と入力し、「繰り返し毎に更新」に設定する。
        * **サイズ [w, h] $** に [30, 30] と入力する。

- 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。
        * **名前** を textRest にする。
        * **終了** を空白にする。
        * **文字の高さ $** を 24 にする。
        * **文字列** に「準備が出来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。
    - 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。
        * **名前** を key_resp_rest にする。
        * **終了** を空白にする。
        * **Routine を終了** がチェックされていることを確認する。
        * **検出するキー $** を'right', 'left' とする。
        * **記録** を「なし」にする。
- 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。
    - **繰り返し条件** に exp10cnd.xlsx と入力する。先に exp10cnd.xlsx を作成してから「選択…」ボタンをクリックして選択するとよい。
- exp10fi.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 30 × 30pix の PNG ファイルを作成し、o.png という名前にする。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、円の中心右側の 10pix 分を消して切れ目を入れる。

最後に、rest ルーチンに配置してある rest_code の **フレーム毎** に以下のコードを入力しておいてください。これは、条件数である 12 の倍数ではない「80 試行毎の休憩」を実現するためのコードです。第 9 章で紹介した break によるルーチンのスキップが利用されています。

```
if trials.thisN % 80 != 0:
    break
```

第 9 章の解説が済んでいなかったら、この 80 試行毎に休憩を挿入する方法も「難問」として挙げなければいけないところでした。念のためコードについても解説しておきますと、trials.thisN は 0 から始まって trials ループが 1 回繰り返されるたびに 1 増加します。trials.thisN を 80 で割った剰余が 0 でない時は break するので、剰余が 0 となる実験開始直後、80 試行終了後、160 試行終了後のみ rest ルーチンがスキップされずに実行されます。よろしいでしょうか。それでは作業を進めましょう。まずは trial ルーチンに 1 個だけ配置してある Image コンポーネントを 15 個まで増やす作業です。

10.3 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

今回の実験では、最大 15 個のアイテムをスクリーン上に提示するために、trial ルーチンに 15 個の Image コンポーネントを配置しなければいけません。前節の作業で Image コンポーネントを配置して **名前** に image00、**回転角度 $** に ori[0]、 **位置 [x, y] $** に pos[0] と入力しましたが、Image コンポーネントを追加してこれらの値を image01、ori[1]、pos[1] にし、さらに続いて Image コンポーネントを追加して image02、ori[2]、pos[2]、… としていって、image14、ori[14]、pos[14] に到達するまで作業を繰り返さなければいけません。もちろん、**開始** や **終了** 、 **サイズ [w, h] $** の値の設定や、「trial の ISI の間に更新」と「繰り返し毎に更新」の設定も忘れずに行わなければいけません。これはかなり面倒です。作業自体も面倒ですが、どれかひとつのコンポーネントで **サイズ [w, h] $** の値を設定し忘れたりした場合に、ルーチンペイン上にずらっと並んだ Image コンポーネントのどれを修正したらいいか探し出すのはうんざりするほど面倒です。この章では、もう最終章ということで「反則技」を紹介しておこうと思います。

ここで解説する方法を用いるには、UTF-8 の文字コードと LF の改行コードのテキストファイルの編集に対応しているテキストエディタが必要です。UTF-8 と LF の組み合わせは Linux 系 OS では標準的なので、Ubuntu などの Linux で作業している方は、標準でインストールされているテキストエディタ (gedit など) で問題なく編集できます。Microsoft Windows や MacOS X ではオープンソースのテキストエディタをインストールする

ことで編集が可能になります。非常に多くのエディタがありますが、筆者が愛用しているのは以下のエディタです。

- サクラエディタ (Windows 向け) http://sakura-editor.sourceforge.net/
- mi (MacOS X 向け) http://www.mimikaki.net/

以下の解説では、サクラエディタの画面を例に用います。まず、作業に失敗した時のためにやり直しが出来るように、exp10proto.psyexp のコピーを作成して exp10.psyexp という名前にしておきましょう。以後の作業は、exp10.psyexp に対して行うものとします。

exp10.psyexp をテキストエディタで開くと、 図 10.4 のように XML 形式というフォーマットで記述された実験の内容が表示されます。XML 形式については *XML* **形式による実験の表現** を参照してください。

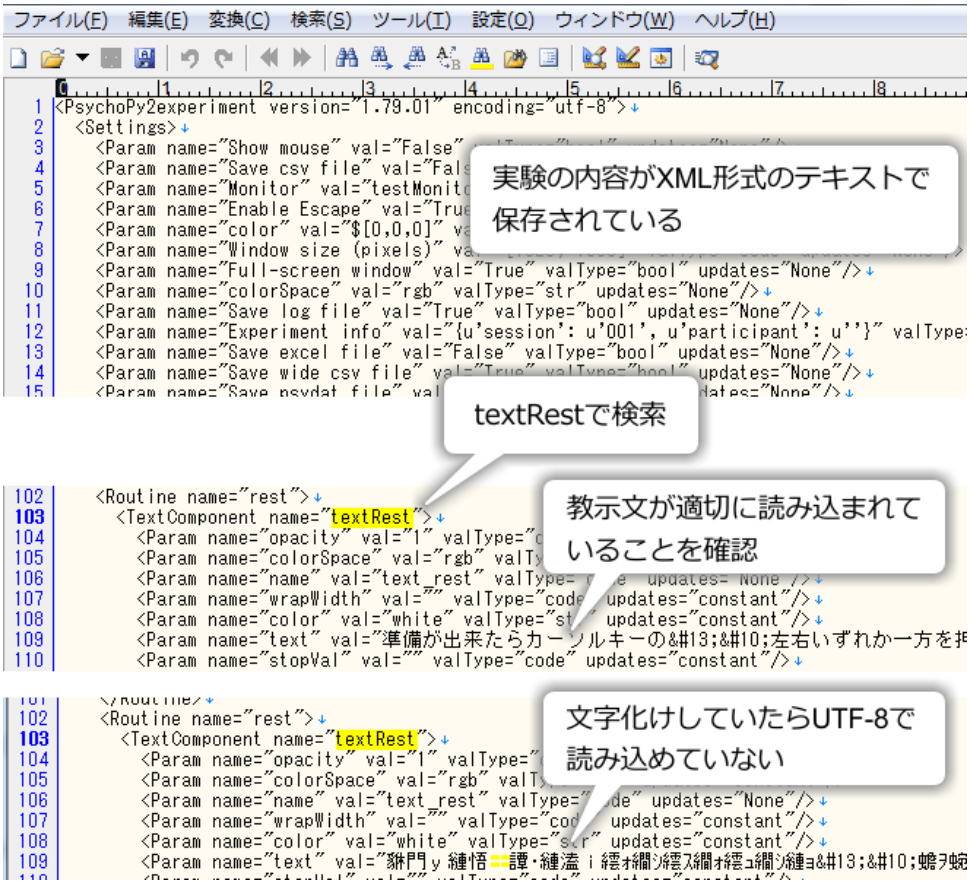


図 10.4　exp10proto.psyexp をテキストエディタで開いた様子。textRest という文字列が見つかった数行下に教示文が適切に表示されていれば、正しい文字コード (UTF-8) で読み込めています。

textRest という文字列を検索すると、rest ルーチンに配置した textRest の設定位置へ移動できます。textRest という文字列が見つかった数行下に教示文が適切に表示されていれば、正しい文字コード (UTF-8) で読み込めています。 図 10.4 の一番下のように意味不明な記号や文字が並んでいる場合は、UTF-8 で読み込めていません。このまま編集作業を続けると教示が失われてしまいますので、ファイルを開き直してください。どうしても UTF-8 で開けない場合は、一旦 Builder に戻って日本語などの非 ASCII 文字を全て削除すれば文字化けは解消されます。exp10.psyexp の場合は textRest の **文字列** に日本語の文字が入力されているので、ここを一旦

空白にしておけば文字化けが解消されます。テキストエディタでの作業を終えてから、教示文を入力し直しましょう。

無事にテキストエディタで exp10.psyexp を開くことが出来たら、image00 という文字列で検索してください。以下のような部分が見つかるはずです。

```
<ImageComponent name="image00">
    <Param name="opacity" val="1" valType="code"…(略)
     (中略)
    <Param name="name" val="image00" valType="code" …(略)
     (中略)
    <Param name="pos" val="pos[0]" valType="code"…(略)
     (中略)
    <Param name="ori" val="ori[0]" valType="code"…(略)
     (中略)
    <Param name="image" val="$imagefile[0]" valType="str"…(略)
     (中略)
</ImageComponent>
```

これが trial ルーチンに配置した Image コンポーネントの設定です。XML をご存じない方でも、なんとなく Builder 上での Image コンポーネントのプロパティ設定画面との対応が想像できるのではないでしょうか。この部分をコピー＆ペーストして image00、pos[0]、ori[0]、imagefile[0] を書き換えれば、image01、image02、image03…を trial ルーチンに追加することが出来ます。

<ImageComponent name=”image00”>という行から、</ImageComponent>という行までを選択してコピーしてください。exp10.psyexp では Image コンポーネントはひとつしか配置されていないので間違えようがありませんが、同種類のコンポーネントが複数個配置されている psyexp ファイルでこのテクニックを使う場合のために、コピー範囲の判断方法を説明しておきます。psyexp ファイルでは、コンポーネントやルーチンなどの要素が定義されている範囲が字下げで判断できるようになっています (図 10.5)。Python の for 文や if 文の文法と似ていますが、Python と違って最初に見つけた同じ字下げの行を要素に含む点に注意してください。コピーしたら、作業用に新しいテキストファイルを開いて 14 回繰り返して貼りつけてください。貼りつけたら exp10tmp.txt という名前で保存しておきましょう。

exp10tmp.txt を保存したら、exp10tmp.txt のパラメータの設定を上から順番に書き換えていきましょう (図 10.6)。ちょっと面倒ですが、Builder 上で 14 個の Image コンポーネントを追加することを考えたら楽なものだと思って頑張ってください。

- image00 を image01、image02、…、image14 に書き換える。ひとつの Image コンポーネントに付き image00 は二カ所あるので注意すること。
- pos[0] を pos[1]、pos[2]、…、pos[14] に書き換える。
- ori[0] を ori[1]、ori[2]、…、ori[14] に書き換える。
- imagefile[0] を imagefile[1]、imagefile[2]、…、imagefile[14] に書き換える。

書き換えが終了したら、exp10tmp.txt を元の exp10.psyexp に貼りつけます。貼り付ける場所は、image00 の定義のすぐ後ろです (図 10.7)。貼りつけたら exp10.psyexp を保存して、Builder から開いてみてください。

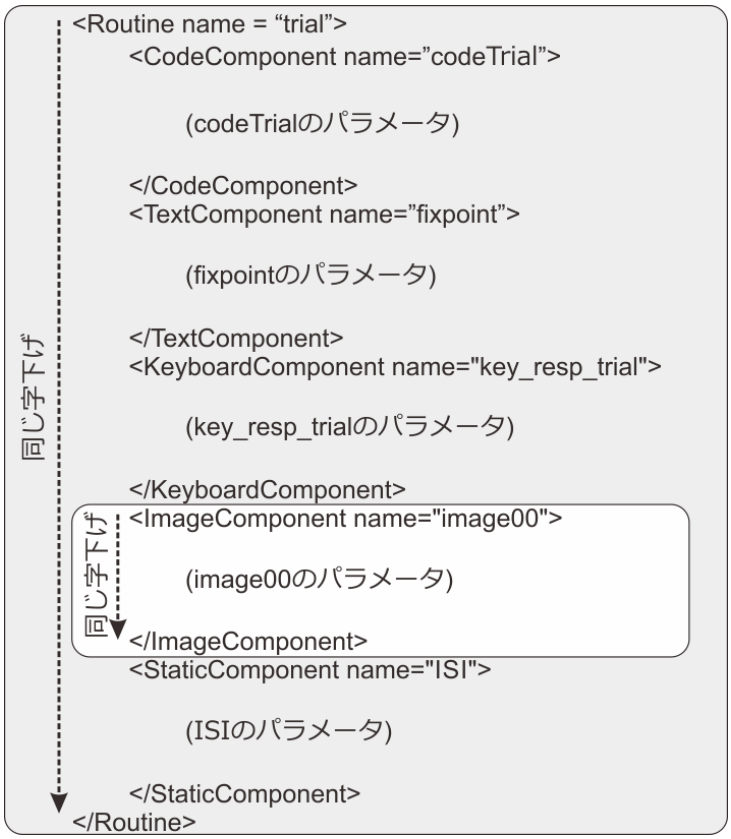


図 10.5　コピー範囲の判断。白い四角形の部分がコピーする範囲です。

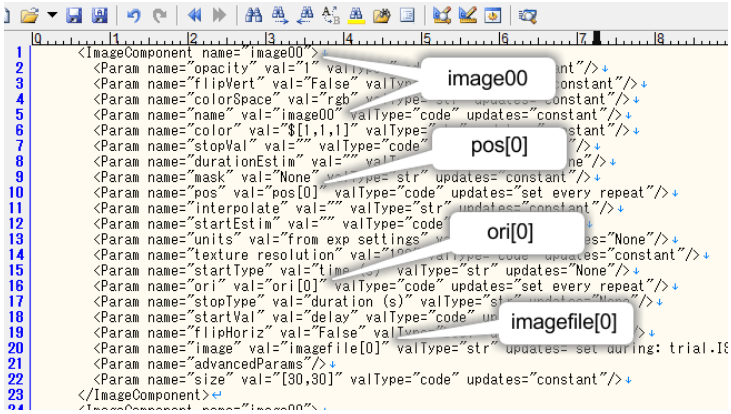


図 10.6　Image コンポーネントの設定を書き換えます。

図 10.8 のように、trial ルーチンに image00 から image14 までの 15 個の Image コンポーネントが配置されているはずです。適当にいずれかの Image コンポーネントをクリックして、image03 ならパラメータが pos[3]、ori[3]、imagefile[3] といった具合に **名前** の番号とパラメータに含まれるリストのインデックスの値が一致することを確認してください。成功したら、もう exp10tmp.txt は削除していただいても構いません。失敗した場合は、exp10tmp.txt に誤りがないか、exp10.psyexp へのコピー位置が正しいかをよく確認して作業をやりなおしてください。

これで 15 個の Image コンポーネントを追加するという問題をクリアすることが出来ました。恐らく Python

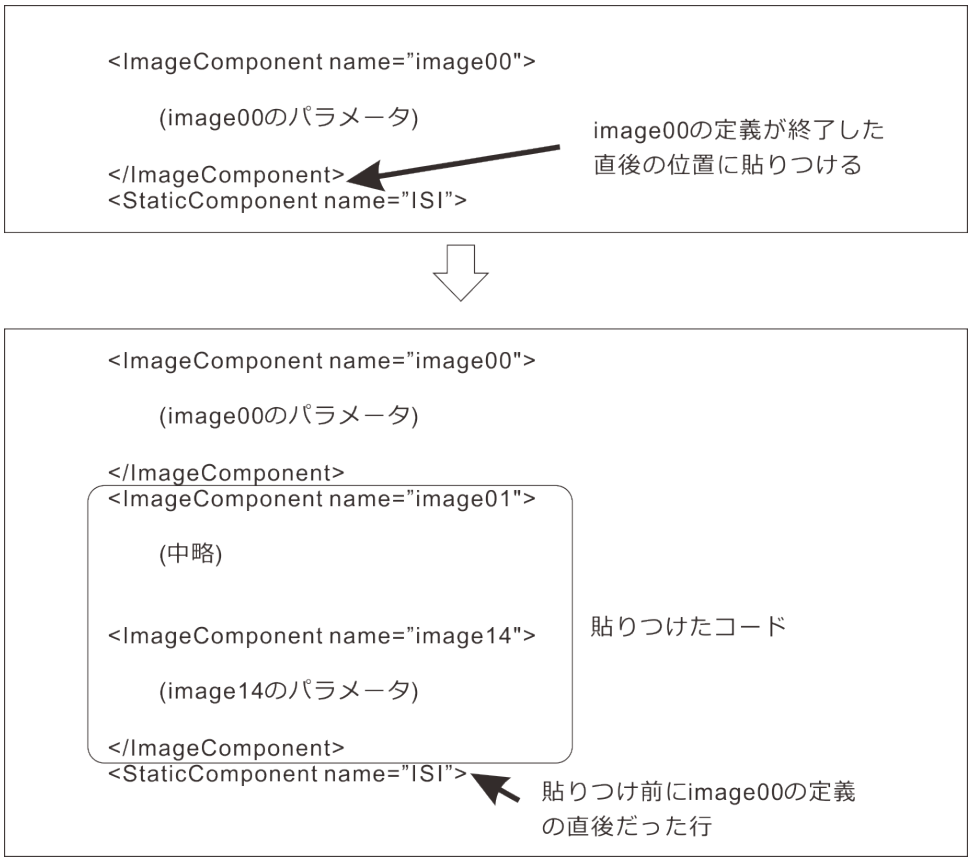


図 10.7　編集したコードの貼り付け。元ファイルの image00 の定義の直後へ挿入するように貼りつけます。

のスクリプトを書ける方は「ここまでするくらいなら全部 Coder でコードを書いた方がいい」と思われるかも知れませんが、psyexp ファイルの仕組みを知っておくといろいろと便利なこともあります。例えば第 3 章で「PsychoPy 1.84.0 以前の Builder にはルーチン名を変更する機能がない」と書きましたが、psyexp ファイルを直接編集すれば簡単にルーチン名の変更ができます。詳しくは *PsychoPy 1.84.0* **より前の** *PsychoPy* **でルーチン名を変更する** をご覧ください。

それでは続いて、試行毎に無作為にアイテムの位置と固視点の提示時間を変更する方法を考えましょう。

チェックリスト

- テキストエディタを用いて適切な改行コード、文字コードで psyexp ファイルを開くことが出来る。
- psyexp ファイルをテキストエディタで開いて **名前** の値を検索して、コンポーネントのパラメータの定義を探し出すことが出来る。
- psyexp ファイルをテキストエディタで編集して、コンポーネントのパラメータを変更することが出来る。
- psyexp ファイルをテキストエディタで編集して、Builder で配置したコンポーネントをコピーして個数を増やすことが出来る。

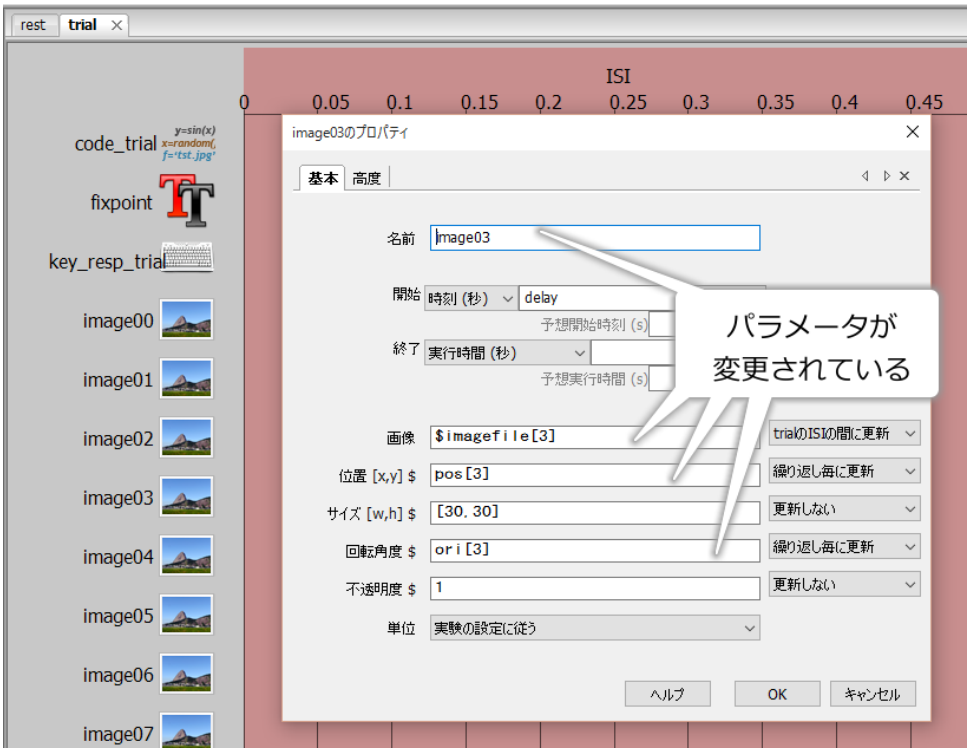


図 10.8　編集後の exp10.psyexp を Builder で開くとテキストエディタで追加した Image コンポーネントが trial ルーチンに表示されます。

10.4 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

これで必要なコンポーネントをすべてルーチン上に配置することが出来ましたので、コードを入力していきましょう。以下の処理をコードで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 10.3 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 10.1) 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出来ます。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出来ます。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 *numpy.ndarray* **型について** を参照してください。

表 10.1　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。

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、randint( ) が便利です。delay の値は 1.0、1.5、2.0 の 3 通りです。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
```

exp10.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。これで delay の値については解決しました。

あまりにもあっさり解決してしまったので、無作為に値を選択する方法についてもう少し考えてみましょう。今回の delay は randint( ) の戻り値から簡単な計算で得ることが出来ましたが、「u’ 一致’、u’ 不一致’ のいずれか一方を無作為に選ぶ」という具合に計算で得ることが出来ない値から選択しないといけない場合はどうすればいいでしょうか。この章まで学んできた人ならピンと来るかもしれません。選択肢のリストを作成して、リストのインデックスに randint( ) の戻り値を使えばよいのです。変数 tasktype に u’ 一致’ または u’ 不一致’ という文字列のいずれかを無作為に選んで設定するのであれば、例えば以下のようなコードで実現することが出来るでしょう。

```
tasklist = [u'一致', u'不一致']
tasktype = tasklist[ randint(0, 2) ]
```

変数 tasklist は実験の最初に一回だけ作成すればよいので、Builder でこのコードを使用する場合は tasklist = [u' 一致', u' 不一致'] は **実験開始時** に書いておいて、tasktype = tasklist[randint(0, 2)] を該当するルーチンの **Routine 開始時** に書くとよいでしょう。この方法はもちろん今回の delay のように、数値をひとつ選択する場合にも使えます。

```
delaylist = [1.0, 1.5, 2.0]
delay = delaylist[ randint(0, 3) ]
```

最初に紹介した、計算によって delay の値を得る方法の場合は、 **Routine 開始時** にだけコードを入力すれば済みますが、数か月後に自分が作成した実験を読み直さないといけなくなったときなどに少々わかりにくいかもしれません。リストを用意する方法は、 **実験開始時** と **Routine 開始時** にコードが分散するという欠点がありますが、後で読み返す時には「ああ、delaylist から値を一つ無作為に取り出しているんだな」という事がわかりやすいかも知れません。どちらの方法を使っていただいても結構です。以上でこの節の解説は終わりですが、最後にひとつ補足しておきます。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( ) とは全く別の関数です。「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。気を付けてください。

チェックリスト

- 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出来る。(low、high は整数)
- 整数の一様乱数を用いて、試行毎に複数の値のリスト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出来る。

10.5 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、もし要素数が 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出来ます。引数が 1 個の場合は、0 から引数より 1 小さい整数までを並べたリストが得られます。例えば 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 までの整数を並べたリストが得られます。例えば 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。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]
```

この range( ) と for 文を組み合わせると、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、range( ) と for 文を組み合わせて 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出来ます。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')
    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。

変数の準備が出来たので、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** に入力して下さい。

```
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 10.9 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 P」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。

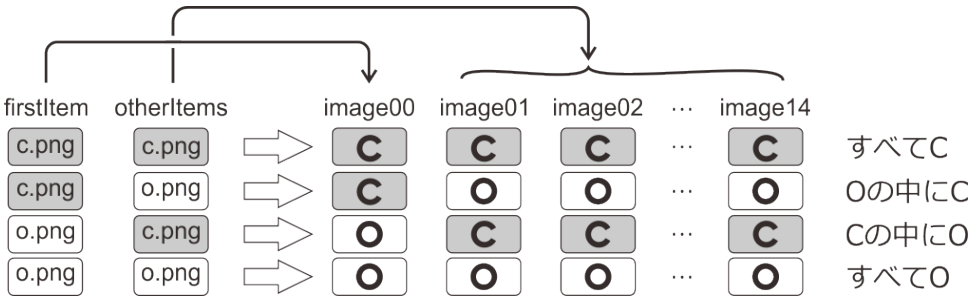


図 10.9　firstItem と otherItems のパラメータ値と刺激条件との対応。

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 の更新が出来たので、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) までの整数を並べたリストを作成することができる。
- range( ) を用いて、m から n (n>m) までの整数を並べたリストを作成することができる。
- range( ) を用いて、m から n まで、s の間隔で整数を並べたリス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。

10.6 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 10.3 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 100pix で一番左下の座標が [-250, -250] 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 10.10 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([100*pos_x-250, 100*pos_y-250])
```

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 10.1 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出来ます。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

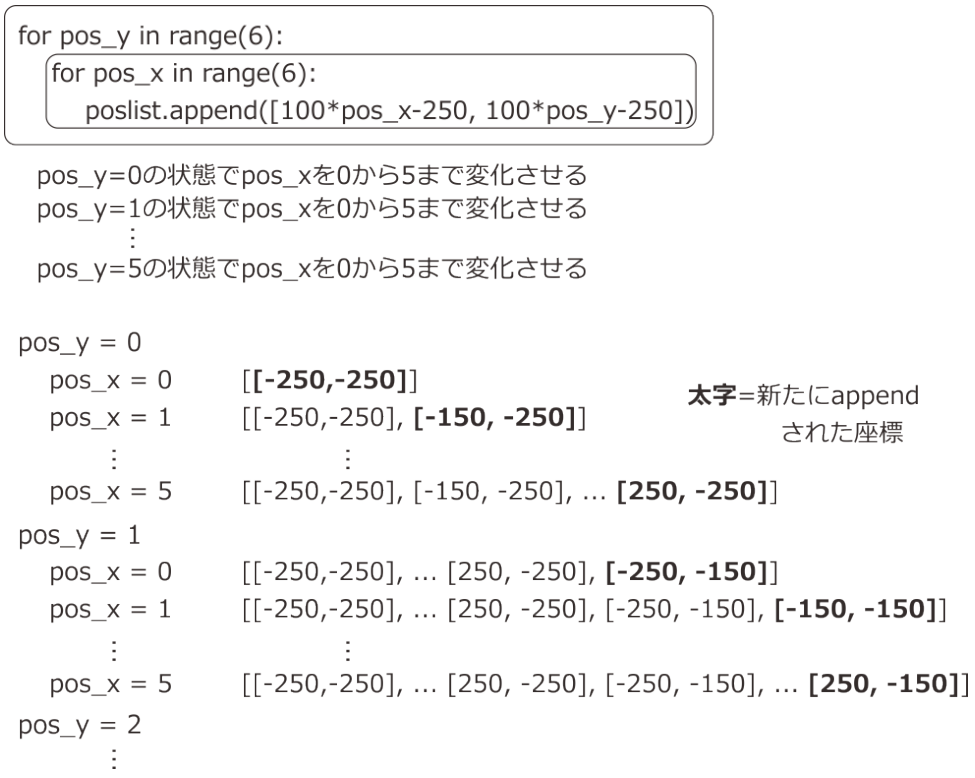


図 10.10　多重 for 文による座標値リストの作成。

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出来るはずです。

```
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、以下のように ori と imagefile の設定を合わせて行うことが出来ます。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 = 0.5*(randint(0,3))+1
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 exp10.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出来る。
- 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出来る。

10.7 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、刺激の単位が norm になっているのに pix のつもりで **位置 [x, y] $** に [500,0] 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大きく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。

```
for i in range(15):
    if i<numItems:
        pos[i] = poslist[i]
    else:
        pos[i] = [10000, 10000]
```

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 0、1、2、3、4 番目の合計 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 0 から n-1 番目までのアイテムに 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 番目までの numItems+1 個のアイテムに poslist の値が設定されてしまいます。

今回の例では、スクリーン外に置いて描画しないようにしたいアイテムに [10000, 10000] という位置を指定しています。時代と共にモニターの高解像度化が進んでいますが、スクリーン中央から右に 10000 ピクセル、上に 10000 ピクセルの位置の刺激が描画出来るモニターが登場するのはまだまだ先のことでしょう。何より、今回の実験は単位を pix にして作成していますので、[10000, 10000] が描画範囲に含まれるほどの高解像度モニターでこの章の実験を実行すると、刺激が小さすぎてまともな実験にならないでしょう。

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 8 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 = 0.5*(randint(0,3))+1
shuffle(poslist)
for i in range(15):
    ori[i] = 90*randint(0,4)
    if i<numItems:
        pos[i] = poslist[i]
    else:
        pos[i] = [10000, 10000]
    if i==0:
        imagefile[i] = firstItem
    else:
        imagefile[i] = otherItems
```

exp10.psyexp に上記の変更を加えたら、exp10.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出来ました。

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出来ると思います。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 displayed Pos に実際に提示に利用された位置をまとめることが出来るでしょう。

```
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 8 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 10.11 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出来ます (図 10.11 例 2)。正と負のインデックスを混ぜて使うことも出来ます。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 10.11 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 10.11 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出来ます。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出来る。
- 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出来る。
- 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出来る。
- 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出来る。

10.8 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

exp10.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。

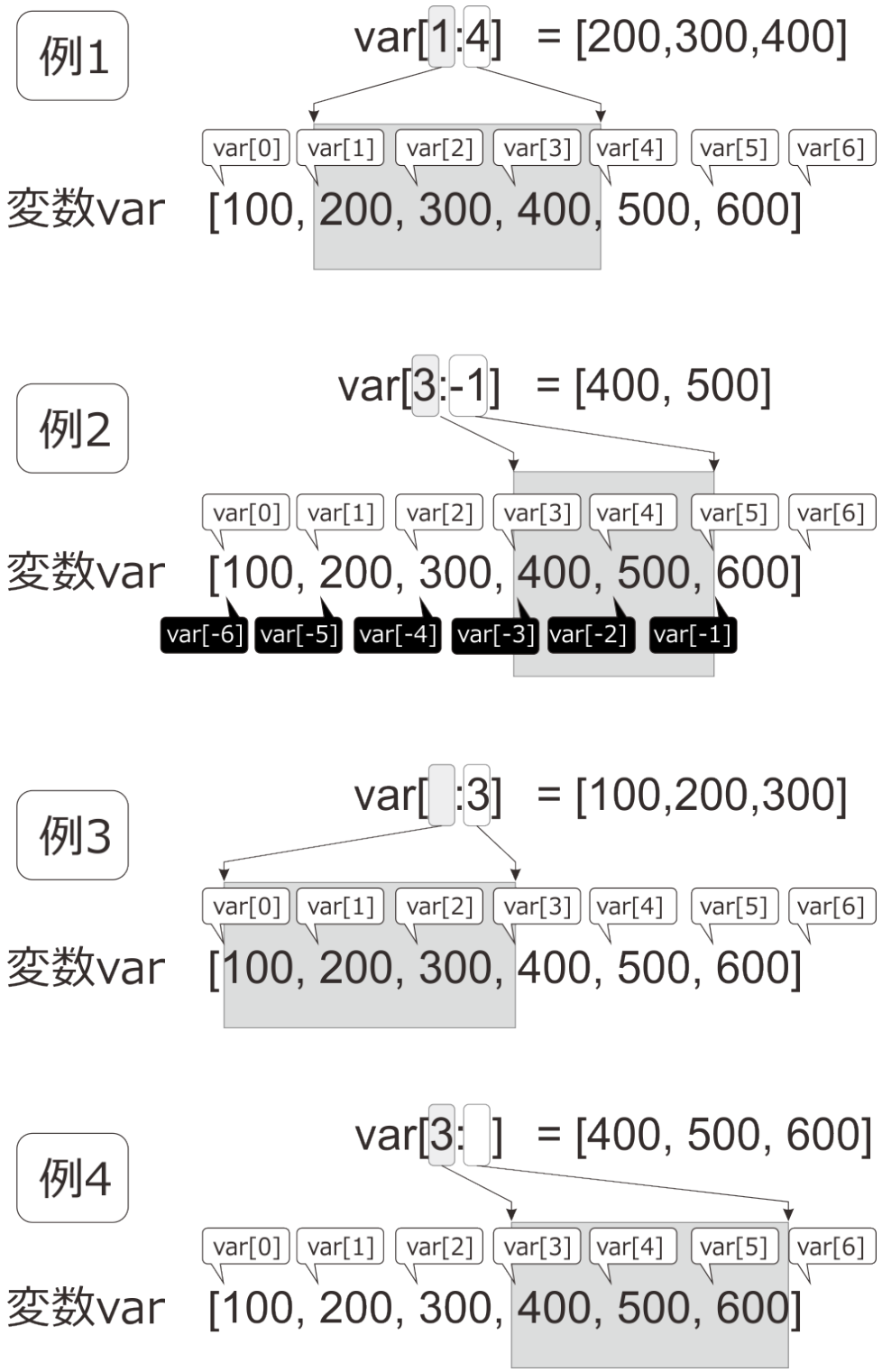


図 10.11　スライスによるリスト要素の抽出。var[a:b] と書くと、変数 var のインデックス a からインデックス b の間にある要素を抜き出します。a が省略されたときは先頭が、b が省略されたときは末尾が指定されたものとします。

- 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-15、-5、5 または 15pix 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。

10.9 この章のトピックス

10.9.1 XML 形式による実験の表現

XML とは Extensible Markup Language の略で、マークアップ言語と呼ばれる言語のひとつです。タグと呼ばれる記号を用いて文書やデータの構造を記述することが出来ます。インターネットの web ページ等を作成したことがある人は HTML をご存知のことと思います。XML のタグは HTML と似ていますが、HTML と異なり自由にタグを定義して使用することが出来ます。

XML の詳細については文献が大量にありますのでそちらを参照していただくとして、psyexp ファイルを読むのに最小限必要なことだけを解説します。XML 文書において、半角のアングルブラケット (山括弧：< >) で囲まれた文字列を「タグ」と呼びます。例えば<Routine>はタグで、Routine というのがタグの名前です。タグは必ず「開始タグ」と「終了タグ」を組み合わせて使用します。Routin の開始タグは<Routine>、終了タグは</Routine>といった具合に、終了タグにはタグ名の前に/が付きます。なお、<Routine/>という具合にタグ名の最後に/が付いているものを「空要素タグ」と呼びます。空要素タグについては後で説明します。

タグの中に、タグ名に続いて Python における変数の代入のような記述が続けて書かれている場合があります。具体的には<Routine name=”trial”>といった具合です。この例において、name=”trial” を Routine タグの「属性」と呼びます。name が属性の名前で、”trial” がその値です。

タグは、開始タグと終了タグの間に他のタグを含むことが出来ます。以下の例では、Routine タグの間に CodeComponent というタグが含まれています。この例において、Routine タグは CodeComponent タグの「親」、CodeComponent タグは Routine タグの「子」と呼びます。XML 文書では、このようにタグを入れ子構造にして、さまざまなデータや文書の構造を記述します。なお、Builder が作る XML ファイルは Python のコードのように字下げされていますが、Python と異なり字下げは必須ではありません。

```
<Routine name="trial">
  <CodeComponent name="code_trial">
  </CodeComponent>
</Routine>
```

exp10proto.psyexp の Routine タグを眺めていると、その要素としてルーチン内に配置したコンポーネントに対応するタグが並んでいることがわかると思います。さらにコンポーネントに対応するタグの要素を確認すると、先ほど述べた「空要素タグ」が見つかります。以下はその例です。

<Param name=”opacity” val=”1” valType=”code” updates=”constant”/>

空要素タグは、子となるタグを持ちません。空要素タグは他のタグを挟み込む必要がないので、単独で使用します。

以上の点を踏まえたうえで、 図 10.12 をご覧ください。 図 10.12 は psyexp ファイルの構造を示しています。Builder の実験は PsychoPy2experiment というタグで表現されます。PsychoPy2experiment は Settings、

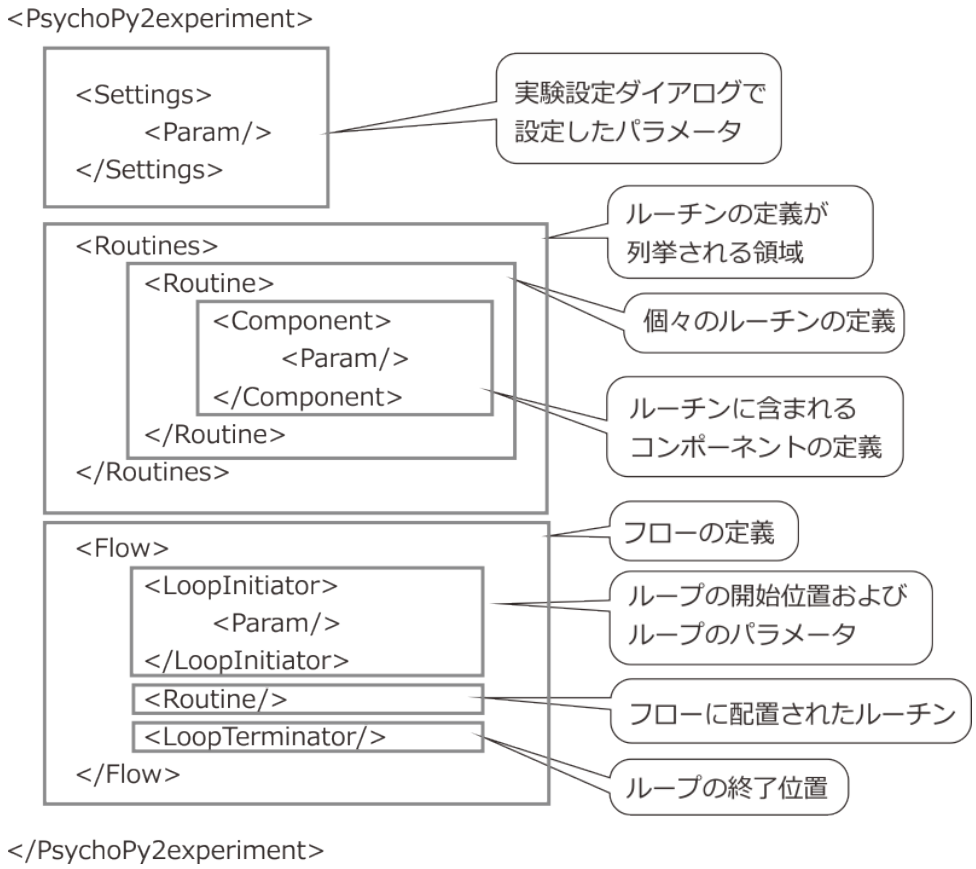


図 10.12　psyexp ファイルの構造

Routines、Flow という要素を持ちます。Settings には実験設定ダイアログで設定したパラメータが記述されています。Routines は、実験で使用されるルーチンの定義である Routine を要素として持っています。 図 10.12 ではひとつしか Routine が描かれていませんが、実験で n 個のルーチンを定義していれば n 個の Routine がここに列挙されます。

個々の Routine は、対応するルーチンに配置されているコンポーネントを定義するタグを要素として持っています。 図 10.12 では Components という名前のタグとして書いてありますが、すでに 図 10.4 や 図 10.5 で見たように、Image コンポーネントに対応するタグは ImageComponent、Keyboard コンポーネントに対応するタグは KeyboardComponent という具合に各コンポーネントに対応するタグが用意されています。本文では trial ルーチンに配置されていた ImageComponent をコピーして貼り付けることによって、Image コンポーネントの個数を増やしました。 図 10.5 や 図 10.7 を見て、コピー範囲と貼り付け位置がタグの入れ子構造を壊さないようになっていることを確認してください。

Flow には、実験のフローが XML で表現されています。Flow の子要素として Routine が配置されている場合は、フローの該当する位置にルーチンが配置されていることを示しています。ループの開始点と終了点はそれぞれ LoopInitiator と LoopTerminator というタグで示されています。多重ループの実験などを適当に作成して(あるいは第 4 章で作成した psyexp ファイルを持ってきて)psyexp ファイルの中を確認すると、フローとこれらのタグの関係がよくわかります。

10.9.2 PsychoPy 1.84.0 より前の PsychoPy でルーチン名を変更する

本文中で述べたとおり、バージョン 1.84.0 より前の Builder にはルーチン名を変更する機能がありません。しかし、実験を保存した psyexp ファイルを直接テキストエディタで編集すれば変更することは可能です。上級者向けのテクニックですので、初心者の方はとりあえず読み流していただくだけで結構です。

psyexp ファイルは LF を改行コードとするテキストファイルです。LF は一般に Unix で用いられる改行コードで、Microsoft Windows などの他の OS で作業している方は、LF を改行コードとして認識できるテキストエディタを使う必要があります。図 3 34 はこの章で作成した exp03.pyexp を開いた様子です。実験の内容が XML(Extensible Markup Language) という言語を用いて記述されています (詳しくは第 10 章参照)。<Routines>がルーチンの定義開始を示すタグで、この中に<Routine name=”xxx”>という形でルーチン名が定義されています。xxx がルーチン名ですので、この行を探して xxx を書きかえて保存した後に Builder で開けばルーチン名が変更されます。ただし、名前を変更したルーチンがすでにフローに挿入されている場合は、フローの対応する部分も変更しておかないと Builder で開く際にエラーになります。フローの定義開始タグは<Frow>で、終了タグ</Frow>までの間に挿入されたルーチンが<Routine>タグで示されています。この中に名前を変更したいルーチン xxx に対応する<Routine name=”xxx”/>という行がありますので、xxx を新しいルーチン名に書き換えてください。

図 3 34 psyexp ファイルの内容。2. と 4. の name=”trial” を name=”testTrial” に書き換えて Builder で開くと trial ルーチンの名前が testTrial に変化しています。

10.9.3 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]) → array([6,5])

asarray([5,3]) * 4 → 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出来ます。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。

第 11 章

付録

11.1 コンポーネントのプロパティ

PsychoPy Builder 1.82.02 で利用できるコンポーネントのプロパティを表にまとめました。最初に各コンポーネントに共通しているプロパティについて解説し、続いて個々のコンポーネントのプロパティを列挙していきます。

なお、日本語のプロパティ名の後ろに括弧書きで PsychoPy の言語設定を英語にした時のプロパティ名を示します。psychopy-users メーリングリストで質問したり、海外の web サイトを検索するときは英語のプロパティ名を使用した方がよいでしょう。

11.1.1 共通プロパティ (一部のコンポーネントを除く)

名前に関するプロパティ

名前 (Name) Builder 内でコンポーネントを識別するための名前を指定します。Python の変数名として解釈でき、かつ Builder の内部で使用されていない文字列でなければなりません。Pyhton の変数名として使用できるのは以下の条件を満たす文字列です。

- 1 文字目が英文字かアンダーバー (_)
- 2 文字目以降が英数字かアンダーバー
- Python の予約語ではない。

大文字と小文字は区別されますので、Foo と foo は別の変数名です。また、1 文字目がアンダーバーの変数名は Python では特別な意味を持つので使用しない方がよいでしょう。

時間制御に関するプロパティ

開始 (Start) コンポーネントが有効になるタイミングを指定します。以下の値から選択します。

- 時刻 (秒) (time(s)) … ルーチン開始からの経過時間 (秒)

- フレーム数 (frame N) … ルーチン開始からのフレーム数
- 条件式 (condition) … 条件式

予想開始時刻 (Expected start (s)) **開始** の値に変数や条件式が指定された時に、ルーチンペイン上でバーを表示するために用います。

終了 (Stop) コンポーネントが終了するタイミングを指定します。以下の値から選択します。

- 実行時間 (秒) (duration(s)) … 有効になってからの経過時間 (秒)
- 実行時間 (フレーム数) (duration(frames)) … 有効になってからのフレーム数
- 時刻 (秒) (time(s)) … ルーチン開始からの経過時間 (秒)
- フレーム数 (frame N) … ルーチン開始からのフレーム数
- 条件式 (condition) … 条件式

予想実行時間 (Expected duration (s)) **終了** の値に変数や条件式が指定された時に、ルーチンペイン上でバーを表示するために用います。

色に関するプロパティ

色 (Color) web/X11 Color name による色名、16 進数表記の web カラー、または色空間内の座標値で色を指定します。座標値で指定する場合は $ が必要になりますのでご注意ください。web/X11 Color name は 図 2.20 、 16 進数表記の web カラーについては *16* **進数と色表現** を参照してください。

色空間 (Color space) **色** で色空間の座標値を指定するときに適用される色空間を指定します。第 2 章を参照してください。

単位に関するプロパティ

単位 (Units) 各パラメータの値に適用する単位を選択します。以下の値から選びます。 **位置と大きさを指定しよう** および *PsychoPy* **における視角の計算について** を参照してください。

- 実験の設定に従う (from exp settings) … 実験の設定に従います。
- cm … 視角を用います。モニターの寸法が適切に設定されている必要があります。
- pix … ピクセルを用います。
- norm … スクリーンの幅と高さを ±1 に換算した単位を用います。
- deg … 視角を用います。視角の計算には近似式を用います。モニターの寸法と視距離が適切に設定されている必要があります。
- degflat … deg と同様ですが、視角を正確に計算します。
- degflatpos … deg と同様ですが、刺激の位置に関してのみ視角を正確に計算します。

11.1.2 Patch コンポーネントのプロパティ

画像ファイルや縞模様などを描画します。このコンポーネントが使用している PsychoPy の PatchStim が廃止 (deprecated) となっており、過去にこのコンポーネントを使用して作成された実験をサポートするために残されているものと思われます。新しく実験を作成する時には Image コンポーネントまたは Grating コンポーネントを使うことをお勧めします。

Patch コンポーネントのプロパティは Image コンポーネントおよび Grating コンポーネントの対応するプロパティを参照してください。

11.1.3 Text コンポーネントのプロパティ

文字列を画面に表示するコンポーネントです。Unicode 対応で日本語の文字列もそのまま表示できます。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

色 (Color), 色空間 (ColorSpace) 「色に関するプロパティ」を参照してください。

フォント (Font) フォント名を指定します。

文字の高さ $ (Letter height $) 文字の縦方向の大きさを正の数値で指定します。**単位** の設定に従います。

位置 [x, y] $ (Position [x, y] $) 刺激の中心の X 座標と Y 座標を表す値を指定します。**単位** の設定に従います。

文字列 (Text) 表示する文字列を指定します。改行も表示に反映されます。

反転 (Flip (mirror)) 文字列を反転して表示します。horiz なら水平方向、vert なら垂直方向に反転します。反転したくない場合は空欄にしておきます。

不透明度 (Opacity $) 刺激の透明度を 0.0〜1.0 の実数で指定します。0 が透明、1 が不透明です。

回転角度 $ (Orientation $) 刺激の回転角度を数値で指定します。単位は deg で、時計回りが正の方向です。

単位 (Units) 「単位に関するプロパティ」を参照してください。

折り返し幅 $ (Wrap width $) 正の実数 文字列の折り返し幅を指定します。表示する文字列の幅がこの値より長い場合は自動的に折り返されます。

11.1.4 Image コンポーネントのプロパティ

画像ファイルを画面に表示するコンポーネントです。上下や左右に反転して提示したり、拡大縮小をしながら提示したりすることができます。JPEG、PNG、TIFF、GIF をはじめとするさまざまな画像を表示できます。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

画像 (Image) 画像ファイル名を指定します。Grating コンポーネントと同様に sin と記入して縞模様を描いたりすることも出来ますが、縞模様を描く場合は Grating コンポーネントを使うべきです。

位置 [x, y] $ (Position [x, y] $) 刺激の中心の X 座標と Y 座標を表す値を指定します。**単位** の設定に従います。

サイズ [w, h] $ (Size [w, h] $) 幅と高さを指定します。数値が一つだけの場合は幅と高さに同一の値を指定したものとみなされます。数値が二つの場合は順番に幅、高さの値とみなされます。 **単位** の定義に従います。

回転角度 $ (Orientation $) 刺激の回転角度を数値で指定します。単位は deg で、時計回りが正の方向です。

不透明度 (Opacity $) 刺激の透明度を 0.0〜1.0 の実数で指定します。0 が透明、1 が不透明です。

単位 (Units) 「単位に関するプロパティ」を参照してください。

色 (Color), 色空間 (ColorSpace) 「色に関するプロパティ」を参照してください。

水平に反転 (Flip horizontally) 画像を左右反転します。

垂直に反転 (Flip vertically) 画像を上下反転します。

補間 (Interpolate) 第 4 章の Grating コンポーネントの解説を参照してください。

マスク (Mask) circle, gauss, raisedCos, 画像ファイル名, None のいずれかを指定します。 *Grating* **コンポーネント** を参照してください。

テクスチャの解像度 (Texture resolution $) 32, 64, 128, 256, 512 から選択します。 *Grating* **コンポーネント** を参照してください。

11.1.5 Movie コンポーネントのプロパティ

動画ファイルを再生するコンポーネントです。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

Routine を終了 (Force end of Routine) 動画再生が終了した時に強制的にルーチンを終了します。

バックエンド (backend) 動画再生に用いるライブラリを avbin または opencv のいずれかから選択します。 *Movie* **コンポーネントのバックエンド** を参照してください。

動画ファイル (Movie file) 再生する動画のファイル名を指定します。

不透明度 $ (Opacity $) 刺激の透明度を 0.0〜1.0 の実数で指定します。0 が透明、1 が不透明です。

回転角度 $ (Orientation $) 刺激の回転角度を数値で指定します。単位は deg で、時計回りが正の方向です。

位置 [x, y] $ (Position [x, y] $) 刺激の中心の X 座標と Y 座標を表す値を指定します。**単位** の設定に従います。

サイズ [w, h] $ (Size [w, h] $) 幅と高さを指定します。数値が一つだけの場合は幅と高さに同一の値を指定したものとみなされます。数値が二つの場合は順番に幅、高さの値とみなされます。 **単位** の定義に従います。

単位 (Units) 「単位に関するプロパティ」を参照してください。

11.1.6 Aperture コンポーネントのプロパティ

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。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

サイズ $ (Size $) 正の実数 「穴」の直径をしめす値を指定します。他のコンポーネントの **サイズ [w, h] $** と異なり幅と高さを別々の値にすることはできません。

位置 [x, y] $ (Position [x, y] $) 刺激の中心の X 座標と Y 座標を表す値を指定します。**単位** の設定に従います。

単位 (Units) 「単位に関するプロパティ」を参照してください。

11.1.7 Grating コンポーネントのプロパティ

縞模様が描かれた長方形を描くコンポーネントです。縞模様の向きや幅をパラメータで調整できます。また、マスクと呼ばれる機能を使って長方形を丸く切り抜いたり、周辺を丸くぼかしたりすることが出来ます。これらのパラメータを組み合わせると視知覚の実験でよく用いられる Gabor パッチなどを簡単に描くことが出来ます。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

色 (Color), 色空間 (ColorSpace) 「色に関するプロパティ」を参照してください。第 4 章の Grating コンポーネントの解説も参照してください。

不透明度 $ (Opacity $) 刺激の透明度を 0.0〜1.0 の実数で指定します。0 が透明、1 が不透明です。

回転角度 $ (Orientation $) 刺激の回転角度を数値で指定します。単位は deg で、時計回りが正の方向です。

位置 [x, y] $ (Position [x, y] $) 刺激の中心の X 座標と Y 座標を表す値を指定します。**単位** の設定に従います。

サイズ [w, h] $ (Size [w, h] $) 幅と高さを指定します。数値が一つだけの場合は幅と高さに同一の値を指定したものとみなされます。数値が二つの場合は順番に幅、高さの値とみなされます。 **単位** の定義に従います。

単位 (Units) 「単位に関するプロパティ」を参照してください。

テクスチャ (Texture) 描画する縞模様の形状を sin, sqr, saw, tri, sinXsin, sqrXsqr, gauss, radRamp, raisedCos, 画像ファイル名, None のいずれかで指定します。Sin は正弦波、sqr は矩形波、saw はノコギリ波、tri は三角波です。波の一周期分の画像ファイルを指定することによって、任意の波形の縞模様を描くことも出来ます。sinXsin、sqrXsqr は水平方向と垂直方向の正弦波／矩形波の重ねあわせです。radRamp と raisedCos は中心から周辺に向かって輝度が変化する円形の模様を描きます。None を指定すると **色** で指定された色で塗りつぶした長方形が描かれます。 *Grating* **コンポーネントの テクスチャ プロパティについて** を参照してください。

マスク (Mask) circle, gauss, raisedCos, 画像ファイル名, None のいずれかを指定します。 *Grating* **コンポーネント** を参照してください。

補間 (Interpolate) 第 4 章の Grating コンポーネントの解説を参照してください。

位相 (周期に対する比)$ (Phase (in cycles) $) 描画する波形の位相を指定します。縞模様をずらしたい時に指定します。

空間周波数 $ (Spatial Frequency $) 実数または実数を二つ並べたシーケンスを用いて空間周波数（波形の繰り返し回数）を指定します。単位が cm、deg の場合はそれぞれ 1cm、1deg あたりの繰り返し回数を表します。pix は「指定された値× Size の幅」の回数繰り返す波が描かれます。norm の場合は刺激の幅に対する繰り返し回数を表します。通常は値を一つ指定すればよいですが、波形を画像ファイルとして読み込んでいて水平垂直の両方向に繰り返したい場合は水平方向と垂直方向の繰り返し回数を並べたリストを指定します。 *Grating* **コンポーネント** を参照してください。

テクスチャの解像度 (Texture resolution $) 32, 64, 128, 256, 512 から選択します。 *Grating* **コンポーネント** を参照してください。

11.1.8 Dots コンポーネントのプロパティ

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。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

色 (Color), 色空間 (ColorSpace) 「色に関するプロパティ」を参照してください。

フィールドの位置 $ (Field position $) 実数を二つ並べたリストで刺激の中心の X 座標と Y 座標を表す値を指定します。 **単位** に従います。

フィールドの形状 (Field shape) フィールドの形状を指定します。「楕円」(circle) または「長方形」(square) を指定できます。

フィールドの大きさ (Field size $) 実数または実数を二つ並べたシーケンスでフィールドの幅と高さを指定します。数値が一つだけの場合は幅と高さに同一の値を指定したものとみなされます。数値が二つの場合は順番に幅、高さの値とみなされます。 **単位** に従います。

コヒーレンス (Coherence $) 0.0〜1.0 の実数でターゲットの割合を指定します。例えば 0.8 ならば 80% の点がターゲットで、20% の点がノイズです。

方向 $ (Direction $) ターゲットが運動する方向を実数で指定します。単位は deg で時計回りが正の方向です。

ドットの寿命 $ (Dot lifetime $) 点が消滅するまでのフレーム数を整数で指定します。例えば 3 であれば個々の点は 3 フレーム描画されると消滅して次のフレームでは新たな場所に出現します。-1 を指定すると点がフィールド外に出るまで消滅しません。

ドットの大きさ $ (Dot size $) 点の大きさを正の実数で指定します。単位は pix です。 **単位** の影響を受けないので注意してください。

ドット数 $ (Number of dots $) 運動する点の個数を正の整数で指定します。

ノイズドット (Noise dots) ノイズの運動を指定します。「一定方向」(direction) であればノイズは一定方向に動き続けます。「ランダム位置」(position) であればフレーム毎にランダムな位置に出現します。「ランダムウォーク」(walk) であればフレーム毎に運動方向が変化します。

シグナルドット (Signal dots) 「同一方向」(same) を指定すると、ターゲットは消滅するまでその方向に動き続けます。「異なる方向」(different) を指定すると、フレーム毎にランダムにターゲットとなる点が入れ替わります。

速度 $ (Speed $) 点が 1 フレーム毎に移動する距離を正の実数で指定します。 **単位** に従います。

11.1.9 Polygon コンポーネントのプロパティ

正多角形を描画します。輪郭線と塗りつぶしの色を別々に指定できます。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), **終了 (Stop)** 「時間制御に関するプロパティ」を参照してください。

頂点数 (N Vertices) 2 以上の整数を指定します。3 以上であれば、指定された値の個数の頂点を持つ正多角形が描かれます。十分に大きい値を指定することで円を描くことも出来ます。

輪郭線の幅 $ (Line width $) 正の実数で線の幅を指定します。単位は pix です。 **単位** の影響を受けないので注意してください。

不透明度 $ (Opacity $) 刺激の透明度を 0.0〜1.0 の実数で指定します。0 が透明、1 が不透明です。

回転角度 $ (Orientation $) 刺激の回転角度を数値で指定します。単位は deg で、時計回りが正の方向です。

位置 [x, y] $ (Position [x, y] $) 刺激の中心の X 座標と Y 座標を表す値を指定します。**単位** の設定に従います。

サイズ [w, h] $ (Size [w, h] $) 幅と高さを指定します。数値が一つだけの場合は幅と高さに同一の値を指定したものとみなされます。数値が二つの場合は順番に幅、高さの値とみなされます。 **単位** の定義に従います。

単位 (Units) 「単位に関するプロパティ」を参照してください。

塗りつぶしの色 (Fill color) 塗りつぶし色を指定します。None を指定すると塗りつぶされません。色の指定については「色に関するプロパティ」を参照してください。

塗りつぶしの色空間 (Fill color space) 塗りつぶし色の色空間を指定します。「色に関するプロパティ」を参照してください。

補間 (Interpolate) 第 4 章の Grating コンポーネントの解説を参照してください。グラフィックチップのドライバに対する描画方法の指示です。通常は標準の値で構いませんが、刺激画像に斜めの隙間などが表示されてしまう場合は変更してみてください。

輪郭線の色 (Line color) 輪郭線の色を指定します。None を指定すると輪郭線が描かれません。色の指定については「色に関するプロパティ」を参照してください。

輪郭線の色空間 (Line color space) 輪郭線の色の色空間を指定します。「色に関するプロパティ」を参照してください。

11.1.10 Sound コンポーネントのプロパティ

音を鳴らします。音は音声ファイル名を指定することが出来るほか、A や Bfl(B ♭)、Csh(C#) のようにキーコードで指定したり、周波数で指定できたりします。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

音 (Sound) 音声ファイル名、A Bsh Csh といったキーコード、または周波数を指定します。

ボリューム $ (Volume $) 音の大きさを 0.0 から 1.0 の値で指定します。

11.1.11 Keyboard コンポーネントのプロパティ

キーボードのキーが押されたことを検出して記録します。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

Routine を終了 (Force end of Routine) キーが押された時点でルーチンを強制終了します。

検出するキー $ (Allowed keys $) キー名をカンマ区切りで並べた文字列で、キー押しを監視するキーを指定します。

記録 (Store) 実験結果ファイルに保存する内容を指定します。「最後のキー」(last key) ならルーチンが終了した時点で最後に押されたキー、「最初のキー」(first key) ならそのルーチンで最初の押されたキー、「すべてのキー」(all keys) ならそのルーチンで押されたすべてのキーを保存します。「なし」(nothing) ならそのルーチンで押されたキーはすべて保存されません。**Routine を終了** を指定している場合は、キーが押された時点でルーチンが終了するので「なし」以外はいずれも同じ結果となります。

正答を記録 (Store correct) そのルーチンにおいて「正しい」キー押し反応がある場合に、押されたキーが正解であったか否かを記録します。

正答 (Correct answer) 正解のキーを指定します。 **正答を記録** がチェックされていない時にはこのプロパティは表示されません。

開始前のキー押しを破棄 (Discard previous) ルーチンが始まるときに、それ以前に押されたキー入力を破棄します。このチェックがついていないと、ルーチンが開始される直前に実験参加者がキーを押してしまった時に誤ってルーチン開始後に押されたキーとして検出される場合があります。通常はチェックしておくべきです。

11.1.12 cedrusButtonBox コンポーネントのプロパティ

Cedrus の反応ボタンボックスを用いて反応を記録するコンポーネントです。特定のハードウェア専用のコンポーネントなので詳細は省略します。

11.1.13 Mouse コンポーネントのプロパティ

マウスのボタン押しを検出、記録したり、マウスカーソルの座標を記録したりします。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

ボタン押しで Routine を終了 (End Routine on press) マウスのボタンがクリックされた時点でルーチンを強制終了します。

マウスの状態を保存 (Save mouse state) マウスの状態を保存するタイミングを指定します。第 8 章の解説を参照してください。

時刻の基準 (Time relative to) 保存される時刻の起点を指定します。routine であればルーチンが開始された時刻、「実験開始時」(experiment) であれば実験が開始された時刻が起点となります。

11.1.14 RatingScale コンポーネントのプロパティ

レーティングスケールを表示し、反応を記録するコンポーネントです。非常に複雑なコンポーネントなので *RatingScale* **コンポーネント** を参照してください。

11.1.15 Microphone コンポーネントのプロパティ

マイクから音声を記録します。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

ステレオ (Stereo) マイクがステレオか否かを指定します。

11.1.16 ioLabsButtonBox コンポーネントのプロパティ

ioLabs の反応ボタンボックスを用いて反応を記録するコンポーネントです。特定のハードウェア専用のコンポーネントなので詳細は省略します。

11.1.17 Static コンポーネントのプロパティ

スクリーンを更新する必要がない時間帯に次の刺激などの準備をするために使用します。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

カスタムコード $ (Custom code $) ツールチップによると更新終了時にここに挿入されたコードが実行されるということですが、バージョン 1.82.02 では無視されるようです。

11.1.18 Code コンポーネントのプロパティ

実験へ Python のコードを挿入するためのコンポーネントです。

名前 (Name) 「名前に関するプロパティ」を参照してください。

実験開始時 (Begin Experiment) 実験開始時にここに記入されたコードが実行されます。

Routine 開始時 (Begin Routine) ルーチン開始時にここに記入されたコードが実行されます。

フレーム毎 (Each Frame) ルーチン中でフレームが更新されるたびにここに記入されたコードが実行されます。

Routine 終了時 (End Routine) ルーチン終了時にここに記入されたコードが実行されます。

実験終了時 (End Experiment) 実験終了時にここに記入されたコードが実行されます。

11.1.19 Unknown コンポーネントのプロパティ

新しいバージョンの Builder で追加されたコンポーネントを使った実験を古いバージョンの Builder で開いた時など、使用中の Builder で利用できないコンポーネントが実験に含まれてる場合があります。このような時、利用できないコンポーネントが Unknown コンポーネントとして表示されます。

実験を作成する際にこのコンポーネントを配置しても何も起きません。

名前 (Name) 「名前に関するプロパティ」を参照してください。

11.1.20 ParallelOut コンポーネントのプロパティ

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。

名前 (Name) 「名前に関するプロパティ」を参照してください。

開始 (Start), 終了 (Stop) 「時間制御に関するプロパティ」を参照してください。

ポートアドレス (Port ddress) パラレルポートのアドレスを指定します。PsychoPy 設定ダイアログの「一般」タブの「パラレルポート」に列挙されている値および LabJack U3 の中から選択します。

開始時データ $ (Start data $) コンポーネントの開始時刻に送信する値を指定します。None なら値を送信しません。

終了時データ $ (Stop data $) コンポーネントの終了に送信する値を指定します。None なら値を送信しません。

スクリーンに同期 (Sync to screen) データの送信と画面の更新を同期させるか否かを指定します。

11.2 予約語

11.2.1 Python の予約語 (Python 2.7)

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 Python2.7 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。

and	as	assert	break	class	continue	def
del	elif	else	except	exec	finally	for
from	global	if	import	in	is	lambda
not	or	pass	print	raise	return	try
while	with	yield				

11.2.2 Builder の予約語 (1.82.02)

Builder (バージョン 1.82.02) の内部で予約語として登録されているもののうち、Python2.7 の予約語でないものを以下に示します。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。

__builtins__	__doc__	__file__	__name__	__package__	abs
all	any	apply	basestring	bin	bool
buffer	bytearray	bytes	callable	chr	classmethod
clear	cmp	coerce	compile	complex	copy
copyright	credits	delattr	dict	dir	divmod
enumerate	eval	execfile	exit	file	filter
float	format	fromkeys	frozenset	get	getattr
globals	has_key	hasattr	hash	help	hex
id	input	int	intern	isinstance	issubclass
items	iter	iteritems	iterkeys	itervalues	keys
len	license	list	locals	long	map
max	memoryview	min	next	object	oct
open	ord	pop	popitem	pow	property
quit	range	raw_input	reduce	reload	
repr	reversed	round	set	setattr	setdefault
slice	sorted	staticmethod	str	sum	super
tuple	type	unichr	unicode	update	values
vars	viewitems	viewkeys	viewvalues	xrange	zip
False	None	True			

以下の語は Builder 内で使用されているモジュール名および定数名です。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。

core	data	event	gui	logging
microphone	misc	os	psychopy	sound
visual	FINISHED	FOREVER	NOT_STARTED	PAUSED
PLAYING	PSYCHOPY_USERAGENT	STARTED	STOPPED	

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	cos	deg2rad	linspace	log	log10
normal	np	pi	rad2deg	randint	random	shuffle
sin	sqrt	std	tan			

11.2.3 Builder の内部変数 (1.82.02)

以下の語は Builder(バージョン 1.82.02) の内部変数として予約されています。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。

予約語	概要
_thisDir	Builder を実行するときにカレントフォルダを psyexp ファイル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
currentLoop	本書では解説しなかった loopType の staircase、interleaved staircases を使用したときに使用されます。
dlg	expInfo ダイアログを作成するために使用します。
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
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。
expName	実験設定ダイアログの [Experiment name] 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 9 章
key_resp	予約されています。
KeyResponse	予約されています。
level	本書では解説しなかった loopType の staircase、interleaved staircases を使用したときに使用されます。
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	本書では解説しなかった loopType の interleaved staircases を使用したときに使用されます。
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。

次のページに続く

表 11.1 – 前のページからの続き

予約語	概要
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。
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 [Name] 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 8 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 9 章

11.3 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
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)
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)
```

11.4 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。

11.4.1 アプリケーション タブ

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。

大きいアイコン ノート PC など、低解像度のモニターを使用しているときはチェックを外すとスペースを節約できます。

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。「前回使用時のウィンドウを開く」、「Builder」、「Coder」、「Builder と Coder を開く」から選びます。

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。空欄にしておくと、OS の言語の設定を利用しようとします。

11.4.2 Builder タブ

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。

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。

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。ロケールが日本語の場合、Foo コンポーネントと表示されているコンポーネントの名前は FooComponent と書く必要があります。

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。

フローを上に表示 チェックするとフローペインが上に、コンポーネントペインが左にレイアウトされます。

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。

11.4.3 Coder タブ

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出来たフォントを使用します。

コメント用フォント コメント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出来たフォントを使用します。

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出来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。

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。

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。

改行文字の変換 ファイルの改行文字を指定します。「unix」にすると改行文字が LF に、「dos」にすると CR+LF に変換されます。「改行コードを変更しない」にすると元ファイルの改行文字を維持します。

11.4.4 一般 タブ

ウィンドウ描画ライブラリ ウィンドウの描画に使うライブラリを指定します。pyglet と pygame が選択できますが、1.82.02 現在で pygame は Builder の機能を十分に活用できないので pyglet のままにしておくべきです。

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 出来ます。

オーディオライブラリ 音声刺激の再生に使用するライブラリを列挙します。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。

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。

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。

11.4.5 ネットワーク タブ

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。

11.4.6 キー設定 タブ

Builder および Coder で利用できるショートカットキーを編集することが出来ます。個々の項目については省略します。

11.5 チェックリスト一覧

11.5.1 第 1 章

- PsychoPy を起動できる。
- Coder のウィンドウから Builder のウィンドウを開くことが出来る。
- Builder のウィンドウから Coder のウィンドウを開くことが出来る。
- 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。

11.5.2 第 2 章

- Polygon コンポーネントを用いて多角形を表示できる。
- 多角形の頂点数を変更できる。
- 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。
- pix を単位に指定して、 **位置 [x, y] $** と **サイズ [w, h] $** に適切な値を入力し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出来る。
- 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。
- **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出来る。
- 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出来る。
- 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、紫にすることが出来る。
- **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、青色にすることが出来る。
- 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輪郭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輪郭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出来る。
- 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出来る。
- 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出来る。
- 刺激の表示終了時刻を定めずに表示させることが出来る。
- 実行中の実験を強制的に終了させることが出来る。
- foo_lastrun.py (foo は psyexp 実験ファイル名) の役割を説明することが出来る。
- data ディレクトリの役割を説明することが出来る。
- 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 で開くことが出来る。
- psyexp ファイルを別の名前で保存することが出来る。
- 実験設定ダイアログを開くことが出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- モニターセンターを開くことが出来る。
- モニターセンターに新しいモニターを登録することが出来る。
- 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出来る。

11.5.3 第 3 章

- 試行中にキー押しが有効となる時間帯を指定できる。
- Text 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出来る。

- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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 $** に値を設定するとどのような効果が得られるか説明できる。
- 条件ファイルを **繰り返し条件** に指定する前に psyexp ファイルを保存すべき理由を説明できる。
- 条件ファイルで定義したパラメータを用いてコンポーネントのプロパティ値を更新できるように設定できる。
- コンポーネントのプロパティ名の最後に$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出来る。
- 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出来る。
- 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出来る。
- trial-by-trial 記録ファイルと xlsx 記録ファイル、summaries 記録ファイルの違いを説明できる。
- 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出来る。
- 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出来る。
- trial-by-trial 記録ファイル、xlsx 記録ファイル、summaries 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出来る。

- trial-by-trial 記録ファイル、xlsx 記録ファイル、summaries 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出来る。
- trial-by-trial 記録ファイル、xlsx 記録ファイル、summaries 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出来る。
- xlsx 記録ファイルおよび summaries 記録ファイルから各条件の試行数を読み取ることが出来る。
- xlsx 記録ファイルおよび summaries 記録ファイルから反応時間の平均値と標準偏差を読み取ることが出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- xlsx 記録ファイルおよび summaries 記録ファイルから各条件の正答率を読み取ることが出来る。
- ルーチンを新たに作成してフローに追加することが出来る。
- 既存のルーチンをフローに追加することが出来る。
- 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出来る。
- 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出来る。
- 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出来る。

11.5.4 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出来る。
- Grating コンポーネントで **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出来る。
- 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、xlsx 記録ファイルまたは summaries 記録ファイルの正答率の値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、削除することが出来る。
- 実験情報ダイアログの項目の初期値を設定することが出来る。
- 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出来る。
- 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出来る。
- 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。

11.5.5 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出来る
- 関数の引数に別の関数の戻り値を使うことが出来る。
- 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出来る。
- python の算術演算子 8 つ挙げてその働きを説明することが出来る。
- 4 ÷ 3 といった整数同士の割り算で、小数点以下を切り捨てない解を得る式の書き方を答えられる。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。

11.5.6 第 6 章

- 画像ファイルをスクリーン上に提示することが出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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。
- N 件法の尺度をスクリーンに提示して反応を記録することが出来る。
- アナログ尺度をスクリーンに提示して反応を記録することが出来る。
- カテゴリカルな尺度をスクリーンに提示して反応を記録することが出来る。
- 複数の文字列を結合した文字列を得る式を書くことが出来る。
- 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出来る。
- 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出来る。
- 条件に応じて処理を分岐させる Python コードを書くことが出来る。
- 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出来る。
- 文字列の一致・不一致に応じて処理を分岐させることが出来る。
- 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出来る。
- 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出来る。
- 変数に値を代入する式を書くことが出来る。
- x+=7、x**=2 といった二項算術演算子と代入演算子を組み合わせた式の働きを説明することが出来る。
- 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。
- 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出来る。

11.5.7 第 7 章

- Polygon コンポーネントを使って線分を描画することが出来る。
- Polygon コンポーネントを使った線分の長さ、角度が決まっている時に、その端点が指定された座標に一致するように **位置 [x, y] $** を指定することが出来る。
- 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出来る。
- 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出来る。
- 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。

11.5.8 第 8 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- [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出来る。
- 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出来る。
- 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出来る。

- 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出来る。
- 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出来る。
- 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。

11.5.9 第 9 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出来る。
- 指定されたキーコードの音を鳴らすことが出来る。
- 音声のボリュームを指定できる。
- MPEG2 形式の動画ファイルを再生できる。
- 動画ファイルを拡大縮小して再生することが出来る。
- 音声ファイル、動画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、動画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出来る。
- なぜ短時間に複数の 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のかを説明することが出来る。
- 実験が開始してからの経過時間を得るコードを書くことが出来る。
- ひとつの数値を 8 進数、10 進数、16 進数整数の書式で文字列に埋め込むことが出来る。
- ひとつの浮動小数点数を 10 進数表記と指数表記の書式で文字列に埋め込むことが出来る。
- 数値を文字列に埋め込む際に、指定した桁数の右寄せで埋め込むことが出来る。桁が足りない時に 0 で埋めることが出来る。
- 数値を文字列に埋め込む際に、指定した桁数の右寄せ埋めで込むことが出来る。桁が足りない時に空白文字で埋めることが出来る。
- 数値を文字列に埋め込む際に、指定した桁数の左寄せで埋め込むことが出来る。
- 浮動小数点数を文字列に埋め込む時に、整数部の桁数と小数点以下の桁数を指定することが出来る。
- 複数の値をひとつの文字列にそれぞれ書式を指定して埋め込むことが出来る。

- 条件式が True である間処理を繰り返す Python のコードを書くことが出来る。
- Code コンポーネントを用いて、ある条件を満たしたときに実行が中断される実験を作ることが出来る。
- Code コンポーネントを用いて、ある条件を満たしたときに実行がスキップされるルーチンを作ることが出来る。
- ある条件に当てはまる時にループを実行しない実験を作成することが出来る。
- ループを実行しない実験を作成した時の trial-by-trial 記録ファイルから、ループを実行しなかった時の記録を判別できる。

11.5.10 第 10 章

- テキストエディタを用いて適切な改行コード、文字コードで psyexp ファイルを開くことが出来る。
- psyexp ファイルをテキストエディタで開いて **名前** の値を検索して、コンポーネントのパラメータの定義を探し出すことが出来る。
- psyexp ファイルをテキストエディタで編集して、コンポーネントのパラメータを変更することが出来る。
- psyexp ファイルをテキストエディタで編集して、Builder で配置したコンポーネントをコピーして個数を増やすことが出来る。
- 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出来る。(low、high は整数)
- 整数の一様乱数を用いて、試行毎に複数の値のリスト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出来る。
- range( ) を用いて、0 から n (n>0) までの整数を並べたリストを作成することができる。
- range( ) を用いて、m から n (n>m) までの整数を並べたリストを作成することができる。
- range( ) を用いて、m から n まで、s の間隔で整数を並べたリス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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。
- 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出来る。