2
BUILDING YOUR FIRST EXPERIMENT

Learning objectives

This chapter will teach you all the basic aspects of the PsychoPy Builder interface while creating a simple experiment: the Stroop task. You will learn how to present stimuli, create conditions that can be randomized, and how to run and analyze the study.

You might be thinking, 'How on Earth do I create an experiment from start to finish?' You might be thinking, 'I'm not very good with computers; this is going to be difficult.' Hopefully we'll be able to show you that you can absolutely create experiments of your own. Even the least 'technical' of users can get up and running.

One of the first things to realize is that you don't have to build the whole study in one go. Work on a piece at a time, save the study and see if it worked. That way you'll feel like each chunk is manageable and you'll narrow down what might have gone wrong if your experiment stops working, because it's probably related to what you just did. So, we'll go through a series of steps thinking about how to define the structure of a single trial in your study (including how to present stimuli and how to collect responses), then how to set that trial to repeat several times and how to make it vary on each repeat, and how to extract the data that you just collected. The study we create here will be basic – there are lots of ways you could extend and improve it – but this will be enough to measure an effect.

Before we get started, though, let's take a quick look at the PsychoPy Builder view.

2.1 THE BUILDER INTERFACE

The PsychoPy Builder interface is designed to allow quick visual construction of a wide range of studies. Let's take a look at what makes a Builder experiment. If you start up PsychoPy and the Builder view isn't immediately visible then you might be in the Coder view instead. Go to the `View` menu and select `Builder` or press `Ctrl-L` (`Cmd-L` on a Mac).

An experiment in Builder has three panels showing you:

- a single *Flow*
- one or more *Routines* that can be combined in the *Flow*
- several *Components* that are combined in each *Routine.*

Info

Why can't Builder do my experiment?

You may find yourself wishing that Builder had more features, but it was designed deliberately to be simple and handle most experimental designs easily. If the interface contained all the necessary controls for all possible experimental designs then, for a beginner seeing it for the first time, it would look much more scary than it does!

Components

Components are the basic items that can be combined to create an experiment. They typically represent either stimuli (e.g. Image or Textbox Components), or methods of responding (e.g. Mouse or Keyboard Components), but can also represent points at which your experiment should perform other actions, like connecting with external hardware. Your experiment can include as many Components as you like and different Components can interact with each other however you choose. The right-hand panel of the Builder interface shows you all the different Components available to you, arranged by category (Figure 2.1).

Pro Tip

Adding favorite Components

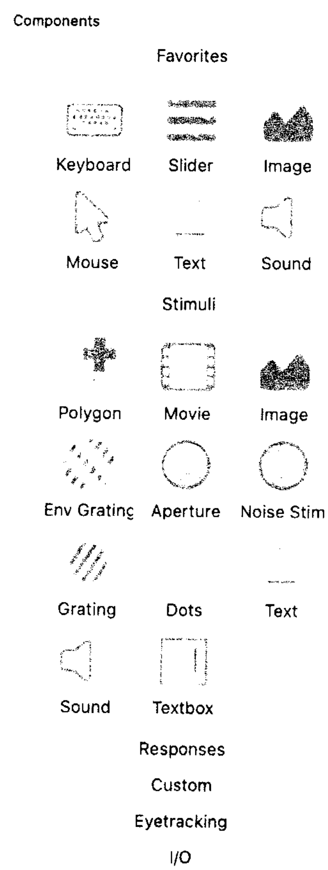
If you expect to use one of the Components a lot you can right-click it and add it to the Favorites category so you can find it more easily in future.

When you click on one of these Component buttons a dialog box will appear for you to set various variables for that Component, and if you press OK the Component will be added to the currently selected Routine.

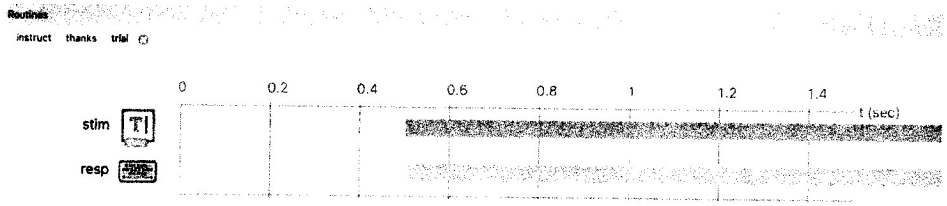
Routines

These define how *Components* interact in time. A typical Routine would be the definition of a trial, presenting one or more stimuli and getting a response, but it might be something else like the presentation of instructions, or feedback, or some other message. Also, a trial could comprise multiple Routines.

Let's take a look at the `trial` Routine for the Stroop task (Figure 2.2). You can see that the Routine has a time view, like a track editor for movie/audio editing software, so Components can be controlled independently in time. A fixation point might stay on for your entire trial, but the stimulus would just appear for a short period.



The Components panel in the Builder interface. Components are [illegible] by groups, which can be opened or collapsed with a mouse click. Note that [illegible] available Components and the appearance of this panel can differ according to [illegible] computer and the version of PsychoPy you have installed. Also note that [illegible] probably more Favorites in the screenshot than in your version – you can [illegible] on a Component and add/remove it from this category.

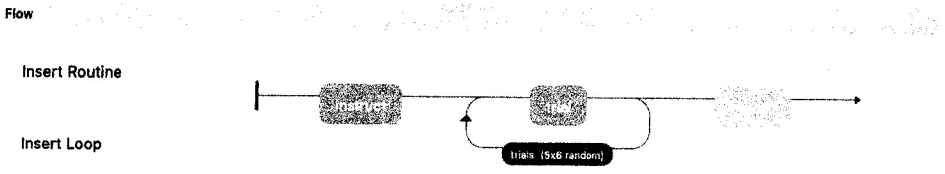


The `trial` Routine in the basic Stroop task. You can see there are just two things comprising the trial, a Textbox Component and a Keyboard Component for the response. In the Stroop demo the stimulus and keyboard both start at 0.5 s after the beginning of the Routine. They both last an infinite duration (the time bar extends beyond the screen).

The Flow

The *Flow* panel is at the bottom of the Builder view. When you first create an experiment it simply has a blue box on the timeline indicating the (default) `trial` Routine. The Flow is just a flow chart (Figure 2.3) and controls the order of things and whether or not they repeat in Loops. It cannot control how long things last in time and does not contain information about what happens within a Routine. Each item on the Flow simply continues until it ends and then we continue to the next item (or around a Loop).

As many Routines can be added to the Flow as you choose (you can also make the Flow icons larger or smaller from the `View` menu if you have too many to be able to see clearly), although if you have an awful lot of Routines in your Flow it *might* suggest that your experiment could be organized more efficiently. Also, a single Routine can be added multiple times at different parts of your Flow. For instance, your study might have some practice trials preceding some main trials, in which case you could have a single Routine called 'trial' that appeared twice, surrounded by two different Loops.



The Flow panel of the Builder view provides a flow chart showing how the different parts of the experiment are combined and in what order. In the case of the Stroop task we have three Routines that run in turn, and the `trial` Routine is repeated a number of times in a Loop. You can alter the size of icons and detail from the `View` menu. When Routines have a known duration they show in a different color than when they have an unknown duration.

Loops

Loops are used to control repetition and also to allow things to vary from one repeat to the next. They are attached to the Flow and can wrap around one or more Routines, but also around any other object on the Flow, including other Loops!

Loops can be used to select random or sequential presentation of conditions (Figure 2.4) or to create 'staircase' procedures whereby the next condition is determined by responses to the previous one (Figure 2.5).

Themes

PsychoPy version 2020.2 added the option to use themes, which customise the way the buttons look, for both the toolbar and the Components. If you want the old button images back, for instance to get more of a 1990s feel, you can go to the `View` menu and select `View>Themes>ClassicLight`. For all the screenshots in this book we will be using the PsychoPy Light theme.

The Toolbar

The first group of icons are probably obvious (New experiment, Open, Save, Undo, Redo). With the others, if you hover you get a tooltip telling you what they do.

The second group of icons concern the settings:

- This controls monitor calibrations and tells PsychoPy about screen sizes.
- This changes settings for this experiment, such as whether it should run in full-screen mode and what the opening dialog should contain and, if you're running studies online, where participants should be redirected to after completing your experiment.

The third group of icons perform a number of useful tasks:

- Builder is ultimately going to create a (slightly complicated) Python script. This button allows you to view the Python code that it will run. If you are running studies online, you can also look at the JavaScript code that will be used using the JS icon.
- This opens the task in the Runner view.
- This launches your current experiment.

The last group of icons allow the project to communicate with Pavlovia to sync your study and/or run it online. We will talk about these in Chapter 5.

2.2 BUILDING THE STROOP TASK

We'll build a task to measure the Stroop effect (Stroop, 1935) in which color names are presented on the screen either in colored letters that match the word being spelled (a congruent condition) or with letters that differ in color from the word.

The task for the participants is simply to identify the color of the letters, ignoring the word itself, but, it turns out, that's extremely hard to do.

First things first. We would recommend you start off by creating a new folder in which to store your experiment files. Don't be tempted to save your experiment onto the desktop 'for now' or you'll quickly find you have a thousand files there and you don't know what half of them do. Also avoid having a folder for all your experiments. We recommend you put each experiment into its own folder. Create that now.

2.3 DEFINE YOUR CONDITIONS

The key to understanding how to set up your study in PsychoPy is to think about what *differs* from one trial to the next and what *stays the same*. Things that stay the same (e.g. the fact that an image is presented or that a response is recorded) will be controlled within the structure of the Routine that determines the trial (which we look at next), whereas things that differ from one trial to the next need to be controlled by a *conditions* file (in either the `.csv` or `.xlsx` format).

For example, in the Stroop task, we always present a text stimulus (a word) and collect a response from the keyboard, so those things stay the same. What changes on each trial is the word that the text stimulus presents, the color of its letters and the value of the correct answer. Those things are *variables* and we need to create a conditions file to determine what values they can have on different trials.

Info

The Stroop effect

John Ridley Stroop (doesn't that name have a nice ring to it? Well done, J. Ridley Stroop's parents!) first demonstrated a very simple but very robust interference effect (known as the 'Stroop effect'). When people are asked to report the color ink a word is written in, they struggle to perform the task when the letters spell the name of an incongruent color. He ran his studies before computers could do this sort of thing and simply asked participants to read out loud from a set of cards and timed them with a stopwatch. His control condition, in which there was no interference between ink and word, was just to present a block of ink, with no word at all. Most modern variants of his task use letters that are different from the word (incongruent) and compare reaction times with letters that are the same as the word (congruent).

The effect was discovered in 1935, as a part of Stroop's PhD dissertation, and is still widely used in psychology and linguistics research. The paper he wrote on it has over 10,000 citations. In particular, it has been used a great deal to study 'executive function' and attention. The fact that most people find it hard to ignore the letters of the written word makes it an interesting tool to study individual differences, asking whether certain groups of people show more or less Stroop-type interference. See MacLeod (1991) for a (slightly outdated) review of those studies.

Others have claimed that this is good training for your brain. A Google search for `brain training Stroop` results in 140,000 pages, most of which suggest you play their game based on the Stroop effect. We do not wish to express any public opinion on whether the Stroop task will increase the longevity of your intellectual abilities!

You can create your conditions file in any spreadsheet application, like Microsoft Excel, OpenOffice Calc or Google Sheets. (Some experiment-generator applications provide dialog boxes for you to create the conditions files yourselves, but since there are many very powerful spreadsheet applications it seems silly not to make use of them.) All you need is something that can save either a 'comma-separated value' (`.csv`) file or a Microsoft Excel (`.xlsx`) file.

For example, in the Stroop task, we always present a text stimulus (a word) and collect a response from the keyboard, so those things stay the same. What changes on each trial is the word that the text stimulus presents, the color of its letters and the value of the correct answer. Those things are *variables* and we need to create a conditions file to determine what values they can have on different trials.

Info

The Stroop effect

John Ridley Stroop (doesn't that name have a nice ring to it? Well done, J. Ridley Stroop's parents!) first demonstrated a very simple but very robust interference effect (known as the 'Stroop effect'). When people are asked to report the color ink a word is written in, they struggle to perform the task when the letters spell the name of an incongruent color. He ran his studies before computers could do this sort of thing and simply asked participants to read out loud from a set of cards and timed them with a stopwatch. His control condition, in which there was no interference between ink and word, was just to present a block of ink, with no word at all. Most modern variants of his task use letters that are different from the word (incongruent) and compare reaction times with letters that are the same as the word (congruent).

The effect was discovered in 1935, as a part of Stroop's PhD dissertation, and is still widely used in psychology and linguistics research. The paper he wrote on it has over 10,000 citations. In particular, it has been used a great deal to study 'executive function' and attention. The fact that most people find it hard to ignore the letters of the written word makes it an interesting tool to study individual differences, asking whether certain groups of people show more or less Stroop-type interference. See MacLeod (1991) for a (slightly outdated) review of those studies.

Others have claimed that this is good training for your brain. A Google search for `brain training Stroop` results in 140,000 pages, most of which suggest you play their game based on the Stroop effect. We do not wish to express any public opinion on whether the Stroop task will increase the longevity of your intellectual abilities!

You can create your conditions file in any spreadsheet application, like Microsoft Excel, OpenOffice Calc or Google Sheets. (Some experiment-generator applications provide dialog boxes for you to create the conditions files yourselves, but since there are many very powerful spreadsheet applications it seems silly not to make use of them.) All you need is something that can save either a 'comma-separated value' (`.csv`) file or a Microsoft Excel (`.xlsx`) file.

Your file should be organized with a column for every variable (each thing that needs to change from one trial to the next) and with a row for each type of trial that you intend to run. For the Stroop task that might look something like this:

word	letterColor	corrAns	congruent
red	red	left	1
red	green	down	0
green	green	down	1
green	blue	right	0
blue	blue	right	1
blue	red	left	0

The `word` variable is obviously the word that will be presented on each trial whereas the `letterColor` represents the color of the letters on the screen. Half the time these variables have the same value and half the time they don't.

Then we have the `corrAns` column. The participants in the task have to respond by pressing the `<left>` key (i.e. the arrow keys near to the number pad here, but you could use any set of keys you choose) when they see red letters, press the `<down>` key when they see green letters and press the `<right>` key when they see blue letters. (Remember: red, green, blue, in that order for the three keys. Remembering the keys is the hardest part of this task!) By specifying in this file what the correct answer is on each trial (the `corrAns` variable) we can get PsychoPy to check whether the participants got the answer right each time.

The final column isn't really necessary, but it makes it easier to perform an analysis in the data file. Remember, our aim is to test whether reaction times are different for congruent trials than for incongruent ones, and by including a column like this we'll be able to simply sort by this variable to make it easier to compare the trials where this was true and false (1 and 0).

There are a few very important rules to follow in creating names of variables (and also of Components):

- *Variables in your conditions file must have unique names*. That means unique from each other and unique from all other aspects of your experiment. If you have a variable called `word` but also a Component called `word` then PsychoPy cannot know which one you're referring to.
- *Variables cannot contain spaces, punctuation or accents*. Apologies to the non-English speakers out there, but these names are going to be converted into variables in a Python script and variables can only have names using ASCII letters. Underscores (_) are allowed and numbers can be used anywhere except as the first character of the name, but that's about all.

- *All columns with values in them need a variable name at the top.* Don't be tempted to have an extra column that contains some information 'just for your benefit' where you didn't need a heading. It is really confusing for PsychoPy. ('You've got values here but I don't know what to call them. Help!')
- *Naming of variables/conditions is case-sensitive.* This means that the names `Stimulus` and `stimulus` would count as unique names (though you might struggle to remember which was the one you intended to refer to if you use both!).

Warning

Watch out for spaces in your conditions file

If you add a space character in one of your headers (e.g. at the end of a variable name) it can be really hard to spot but it will count as one of those forbidden characters in your variable names.

2.4 DEFINING THE TRIAL STRUCTURE

To define how each trial will run we need to think about what stays the same on each trial. In the Stroop task each trial consists of a text stimulus being presented and a response being recorded on the keyboard. We can do all this in a single Routine but in some experiments your trial might need more than one Routine.

Open the PsychoPy Builder view and make sure you have an empty (New) experiment so that you are starting with a clean state. Save it straightaway, next to the conditions file, in the folder you created. You might call it something like `stroop.psyexp`, although the `.psyexp` extension should be added automatically. The `.psyexp` extension always denotes a PsychoPy experiment and is distinct from the `.py` files which indicate (Python syntax) script files. Do keep in mind the location of the folder and experiment file. This may seem obvious, but it seems quite common that we say that the data are in a folder 'next to the experiment file' and it turns out the user doesn't know where their experiment file was saved.

Create your stimulus. The first thing we need to add for each trial is a Textbox Component to present the stimulus. Create a new Textbox Component by clicking the button in the Components panel (on the right of the Builder view). Note that the buttons in the Components panel create new Components, but you can edit an existing Component by clicking on its icon or timeline in the Routines panel. You can remove a Component by *right*-clicking its icon in the Routines panel. (If you have an Apple Macintosh computer you may find that right-click isn't enabled because Apple seems to think that using two buttons on a mouse is too confusing for the average user! You can/should turn this on in the System Settings because your mouse almost certainly does actually contain two buttons.)

ADD TEXTBOX 'STIM'

Info

Textbox versus Text Components

In the first edition of this book we referred to a Text Component, whereas we're now using a Textbox. The Textbox is a new Component that has various advantages over the original Text Component. The first is that you can give it a box, a border color and a fill color. The next is that this object has the option to be editable so you can use the same object either to provide a stimulus, as in the Stroop task, or to allow participants to type a response. The Textbox, unlike the Text, allows you to set individual characters to be bold or italicized by using and or and around the characters. Lastly, despite having more features, this Component actually renders to the screen faster than the original Text Component.

All Components need to be given a name, and the names follow the same rules as for the variables (unique, no punctuation, case-sensitive). For instance, we can't call our Component 'word' because we already used that name as a variable in the conditions file. Let's call it `stim` instead.

Pro Tip

Use good names

It's really worth thinking carefully about the name you give your Component, especially if your experiment has many Components and variables. When you come back to your experiment in a year's time and find three Components called `image_1`, `image_2` and `image_3`, you might not remember what each of the images was for. If you called them `target`, `flank_left`, `flank_right` it would have been more obvious. For response objects (like the Keyboard Component) the name also appears in the header of the data output, so if you name them clearly it will be much easier to tell later which column you need to analyze and which one was caused simply by the participant advancing to the next block of trials (both of which might have been driven by keyboard events).

If the stimulus comes on at the very beginning of the Routine it will appear immediately after the participant responded to the previous trial. For a participant that's surprisingly stressful; they'll complain that your study is 'running too fast'. We can create an *inter-trial interval* (ITI) simply by delaying the start of our stimulus until 0.5 s (500 ms) into the

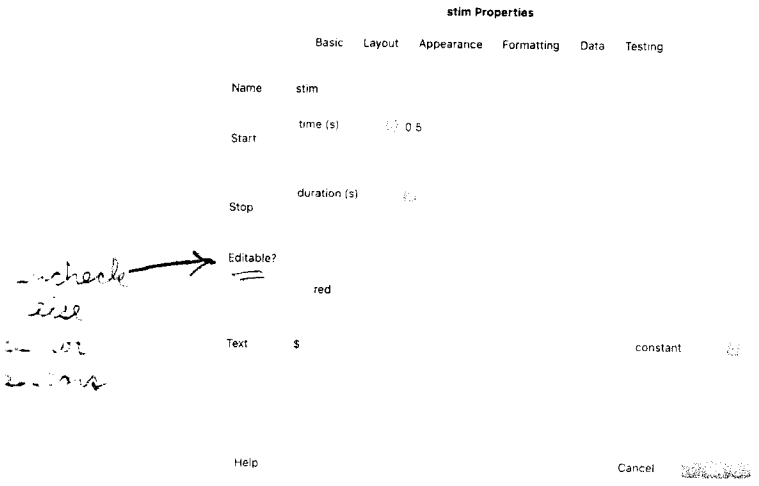
Routine. Set the *start* time in your Textbox Component dialog to 0.5 (PsychoPy refers to all times in units of seconds, but that doesn't mean it's less precise than other packages that refer to time in milliseconds). We could present the stimulus for any *duration* really, but let's delete all the contents of the *duration* box, which will cause the `stim` to be presented indefinitely. Lastly, set the Text field of the Component to have the value `red`. Your dialog box should look something like Figure 2.6.

Note that we changed the following settings from their default entries:

- the `Start` time to `0.5` (seconds after Routine started)
- the `Stop` (duration) was set to be blank, but it could also be shorter if you prefer
- the `Text` became `red`.

Several of the properties of Textbox Components are common to many Components. We encountered `Start` and `Stop` already but note that there are several ways they can be set:

- So far we simply used seconds, which is the most intuitive way to time stimuli, but not always the most precise.
- We could also have used the number of frames instead of seconds, which is generally more precise for brief stimulus presentations.
- We could also have used a condition, which would be a Python expression. Often this is used to yoke one stimulus to another. For instance, you could set a condition that `distractor.status==FINISHED` to make something start/stop when the `distractor` stimulus stopped, even though that time might not be known in advance.

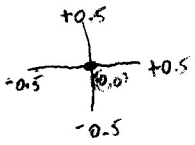


Presenting the word 'red' for a while before starting at 0.5 s and fading. This screenshot has come from PsychoPy and the dialog looks slightly different on Windows.

- For start times and durations where the time in seconds isn't known in advance (because it is based on a variable or condition or computed using frames which can differ from one computer to another), these don't automatically appear on the timeline of the Routine, but you can set the `Expected duration` in seconds if you wish to make them more visible. The `Expected duration` has no effect on the actual experiment.

While we're here, take a look at some of the parameters in the other tabs. We've got tabs for `Layout`, where we can position the stimuli and set the size of their box, as well as `Appearance` (for properties like colors), the Textbox has `Formatting` to control letter sizes and fonts, and nearly all objects have `Data` and `Testing` to control what data are collected about this object in data files and whether we should ignore the object due while testing our experiment.

The `Position` parameter in the `Layout` tab is common to nearly all stimuli, and refers to the center of the stimulus, although some objects also support an `Anchor` property which allows the user to vary which part of the object the position refers to. The coordinates of the position are controlled by the *units* that have been set for that stimulus (or for the experiment as a whole). PsychoPy supports various units for its coordinates, explained more fully starting in Section 13.1. In all cases the center of the screen is (0,0) corresponding, fairly obviously, to the x and y axes of your stimulus. The first coordinate runs negatively to the left and positively to the right, whereas the second runs negatively to the bottom and positively to the top. By default (and for our purposes here it's fine), the units are 'height' in which the bottom is –0.5 (half the height of the screen down from the center) and the top is +0.5. Beware of confusions between units. Also, using units incorrectly is one of the very common reasons why a stimulus 'doesn't appear' because you've accidentally tried to draw it 4 m to the right of your monitor, say, or 1/100th the size of a pixel. So do go and read the section on units (Section 13.1) carefully at some point!



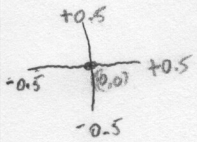
While we're talking about units, there's also a Textbox Component parameter called `Letter height`. It sets, well, the letter height, but there's a caveat as well, which is that the height refers to the maximum height of any character in the font. Obviously, a letter like 'e' doesn't go near the top or bottom, so its height will be a lot smaller than this. 'E' goes quite near the top and 'g' goes near the bottom for many fonts but there are often still gaps: 'É' obviously goes higher than 'E'. If you want a very precise size of letter you need either to measure its height in the specific font you wish to use, or to create a bitmap image of the text and present that image instead.

Colors, in the `Appearance` tab, can be defined for most visual stimuli, and these can also be set using a variety of ways: names (any of the X11 colors), hexadecimal values like #00FFC0, as well as a triple of values in various color spaces (RGB, RGB255, DKL, HSV, etc.). These are covered in more detail on Section 13.2. There is also an `Opacity` parameter: most visual stimuli can be made semi-transparent so that you can see things behind them, or with `Opacity=0` they can be made to disappear altogether.

- For start times and durations where the time in seconds isn't known in advance (because it is based on a variable or condition or computed using frames which can differ from one computer to another), these don't automatically appear on the timeline of the Routine, but you can set the `Expected duration` in seconds if you wish to make them more visible. The `Expected duration` has no effect on the actual experiment.

While we're here, take a look at some of the parameters in the other tabs. We've got tabs for `Layout`, where we can position the stimuli and set the size of their box, as well as `Appearance` (for properties like colors), the Textbox has `Formatting` to control letter sizes and fonts, and nearly all objects have `Data` and `Testing` to control what data are collected about this object in data files and whether we should ignore the object due while testing our experiment.

The `Position` parameter in the `Layout` tab is common to nearly all stimuli, and refers to the center of the stimulus, although some objects also support an `Anchor` property which allows the user to vary which part of the object the position refers to. The coordinates of the position are controlled by the *units* that have been set for that stimulus (or for the experiment as a whole). PsychoPy supports various units for its coordinates, explained more fully starting in Section 13.1. In all cases the center of the screen is (0,0) corresponding, fairly obviously, to the x and y axes of your stimulus. The first coordinate runs negatively to the left and positively to the right, whereas the second runs negatively to the bottom and positively to the top. By default (and for our purposes here it's fine), the units are 'height' in which the bottom is –0.5 (half the height of the screen down from the center) and the top is +0.5. Beware of confusions between units. Also, using units incorrectly is one of the very common reasons why a stimulus 'doesn't appear' because you've accidentally tried to draw it 4 m to the right of your monitor, say, or 1/100th the size of a pixel. So do go and read the section on units (Section 13.1) carefully at some point!



While we're talking about units, there's also a Textbox Component parameter called `Letter height`. It sets, well, the letter height, but there's a caveat as well, which is that the height refers to the maximum height of any character in the font. Obviously, a letter like 'e' doesn't go near the top or bottom, so its height will be a lot smaller than this. 'E' goes quite near the top and 'g' goes near the bottom for many fonts but there are often still gaps: 'É' obviously goes higher than 'E'. If you want a very precise size of letter you need either to measure its height in the specific font you wish to use, or to create a bitmap image of the text and present that image instead.

Colors, in the `Appearance` tab, can be defined for most visual stimuli, and these can also be set using a variety of ways: names (any of the X11 colors), hexadecimal values like #00FFC0, as well as a triple of values in various color spaces (RGB, RGB255, DKL, HSV, etc.). These are covered in more detail on Section 13.2. There is also an `Opacity` parameter: most visual stimuli can be made semi-transparent so that you can see things behind them, or with `Opacity=0` they can be made to disappear altogether.

You can find out about the other parameters of the Textbox Component by hovering your mouse over their entry boxes, or by pressing the `Help` button at the bottom of the dialog box, which will take you to online help for any dialog.

Collecting a Keyboard Response

Given that we just set the text stimulus to last forever, we'd better make sure we can end the trial somehow. Very often a trial will be set to end when the participant responds to the stimulus, and that will work fine for the Stroop task.

Let's add a Keyboard Component (click on the keyboard icon in the Components panel). The structure of this is similar to the above. We need to give our Keyboard Component a name; let's call it `resp`. Now, we don't want participants to respond before the stimulus appears, but we're happy for them to respond as soon as it does. We'll set the keyboard to start at 0.5 s (i.e. we'll begin polling it for keypresses at the same time as the stimulus. The Keyboard Component will measure the reaction time from its start time, so this will mean that the reaction time is relative to the start of the stimulus as well.

Make sure you set the Keyboard to last as well (set the stop value to be blank). If we leave the stimulus on the screen but stop checking the keyboard it can look as though the experiment has 'crashed'. It's actually still running but hasn't been told how to get out of the stimulus presentation cycle.

We can set the keys that are *allowed* by inserting a list of key names. To do this each key name must be in quotes (either the `"` character or the `'`, though they must match) and separated with a comma from the others. On this occasion we use the keys `'left'`, `'down'`, `'right'`, so type the key names in this box. Alternatively, you could set the box to contain nothing (meaning any key is allowed), but if you've given your participant instructions to use the arrow keys and then they press something else, you don't want that to count as a response.

Pro Tip

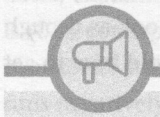
But why do I need to type in quotes and commas in the keys list?

It might seem unnecessary, but there are two reasons. The first is that this allows PsychoPy to know the difference between `"left"` and `"l","e","f","t"`, whereas if we just wrote `left` it would be ambiguous which thing we meant. The second reason is that this is actually valid Python code and that makes it very easy for PsychoPy to interpret.

OK, the last thing we need to do with the Keyboard Component (for now) is to check that `Force end of Routine` is ticked on. This is turned on by default, but just check it hasn't accidentally been switched off. That is going to end this Routine and advance the Flow to whatever comes next.

Save and Run

Once you've saved the file you can run it by pressing the lovely green run icon (or press `Ctrl-R` in Windows/Linux or `Cmd-R` on a Mac). With any luck a dialog box came up asking you to provide a participant ID and a session number. Neither are important right now, so you can just press `OK`. You should see your stimulus appear on the screen and you should be able to end the trial (and the entire experiment so far) by pressing one of the `<left>`, `<down>` or `<right>` arrow keys. If you got stuck with your stimulus word on the screen then pressing the `<esc>` key (top left corner of most keyboards) will quit the experiment. If this was needed then go and check your Keyboard Component; did you remember to clear its duration to be blank (so it lasts forever)? Did you set the `Allowed keys` correctly?



Warning

The red stop button aborts with no data saved

In general, PsychoPy will attempt to save data for you and is generally successful at that, even if your participant aborts with the `<esc>` key or if your experiment has an error right at the end. If you press the red stop button that will not happen. This is a very forceful exiting of the experiment and typically results in no data being saved.

2.5 ADDING A LOOP TO REPEAT TRIALS

Technically you could create a large number of Routines, one for each of your trials, and add them sequentially to your Flow, but that would be very inefficient and also wouldn't allow your trial Routines to be presented in different order. We can do this more efficiently and more flexibly using a *Loop*.

Click *once* on `Insert Loop` in the Flow panel, as shown in Figure 2.7. This will cause a dot to appear on the timeline of the Flow. You can move the dot to any location to signify the start or end of the Loop (it doesn't matter which). Click *once* to insert the end of the Loop *just before* the trial Routine on the Flow as in Figure 2.8.

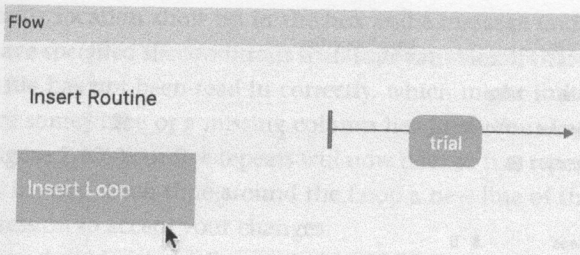


Figure 2.7 Clicking the `Insert Loop` button on the Flow panel (don't double-click it!).

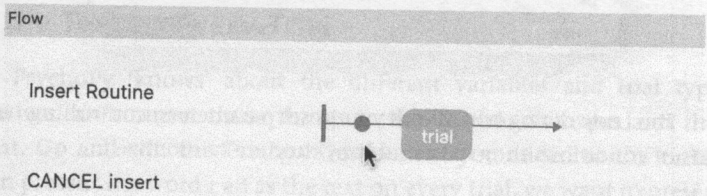


Figure 2.8 Hover over where you want the Loop to start and/or finish and click the mouse once (don't double-click it!).

Then click the start and the end points for your Loop (it doesn't matter which you do first). If there is only one remaining valid place for your Loop to go, the other point will be added automatically.

Usually, you need to insert both the start and end points of your Loop but, on this occasion, there is only one valid place for the end to go once the start has been decided (and vice versa), so this time PsychoPy will insert the other end of the Loop automatically for you.

When the start/end points are added, a dialog will appear to allow you to control the settings (see Figure 2.9).

As with the Components that you added to the Routine, you can *edit* the properties of an existing Loop by clicking on its name/icon later, and you can remove it by *right*-clicking the name or icon.

As with Components, the Loop needs a name, and it also follows the same rules of needing to be unique and not contain punctuation or spaces. In our case the default name of 'trials' seems sensible, but if you intend to have practice trials and main trials then it would be best to distinguish between them in the Loop name.

The next parameter determines what *type* of Loop will be inserted. This is most often `random`, in which case the lines of your conditions file will be chosen at random for each repeat of the Loop (there's also a `full random` option; the difference between this and

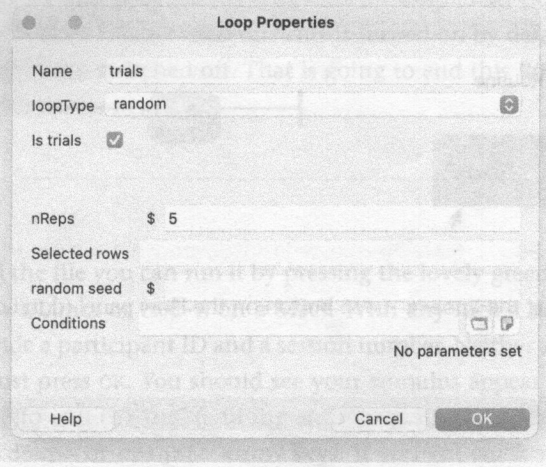


Figure 2.9 The Loop dialog box allows you to set parameters, controlling how many repeats of conditions they have and how they are randomized.

`random` is covered in Section 10.2). You can also have a `sequential` Loop where the lines of your conditions file are used in the same order as you had specified. Lastly, there are options for using the staircase procedures discussed in Section 18.6.

The next tick-box parameter, with the label `Is trials`, might seem strange. It effectively indicates whether each entry should be on a new line in the data file.

The reason for this parameter is that a Loop might not indicate a trial. You could have an inner Loop in which 10 stimuli are presented and these 10 stimuli represent one trial. You wouldn't want a new line for each stimulus, just a new line for the next *trial* of 10. Or you might have an outer Loop to indicate blocks around an inner Loop of trials (see Chapter 8).

For now, let's just make our trial repeat five times without changing any stimuli, just to check it works. To do so all the existing properties can be left as they are. Now when you save and run your experiment you can see five 'trials' (albeit identical each time) running and you can end each one by 'responding' (of sorts) using the arrow keys.

2.6 VARYING YOUR STIMULI ON EACH TRIAL

So far so good, except that this experiment doesn't achieve a whole lot, on account of the stimuli never changing. The remaining piece of the puzzle is simply to take the variables from the conditions file that we created and connect them to the parameters of our Components.

Click on the label of your `trials` Loop that you already created to edit its properties. Press the open file icon next in the dialog to find your conditions file (`conditions.xlsx`). Actually, if you hadn't already made a conditions file, you could select the table editor icon to automatically open up Excel (or your default table editor) and make the conditions file. If all went well when you created that file you should now see the

conditions file location show up in the box and a message underneath reminding you that you have specified six conditions with four variables. If that message doesn't appear then your file has not been read in correctly, which might indicate an error in the file (e.g. a space somewhere or a missing column heading). Your Loop settings should now look like Figure 2.10. Your five repeats will now refer to five repeats of all six conditions, so 30 trials in total. Each time around the Loop a new line of the file will be randomly selected. Press `OK` to accept your changes.

Your Flow should now look something like Figure 2.11. If you have other Loops or Routines, or if your Loop doesn't go around your Routine, then you may need to remove it by *right*-clicking on the erroneous entry.

Update the Text on Every Trial

Now that PsychoPy 'knows' about the different variables and trial types in your experiment we can connect those things to the Textbox stimulus and the Keyboard Component. Go and edit the Textbox Component you added to the `trial` Routine. Rather than present the word `red` as the text on every trial, we want to present the `word` variable from our conditions file. Obviously, we can't just write `word` into this box; that would present the word 'word' as a stimulus! We want it to present the *contents of the variable* named `word`. To do this we add a `$` symbol to the box to indicate that PsychoPy should treat this as code, not as literal text. Note that some boxes (like `Position`) are always treated as code because they would make no sense to be interpreted as text. These boxes have a `$` next to them always so you don't *need* to add one. So go ahead and delete the word `red` from your text and add `$word`, making sure you spell it *exactly* as you did in the conditions file, with the same letter case, etc.

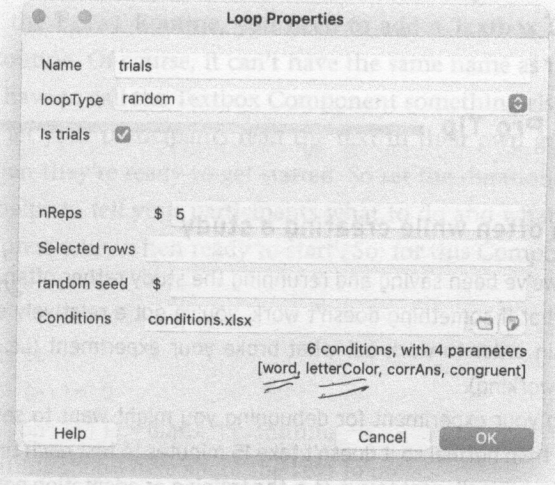


Figure 2.10 Completed dialog for your Loop. Note that the dialog tells you about the conditions file that you imported (indicating that it was imported successfully). The five repeats will result in a total of 30 trials.

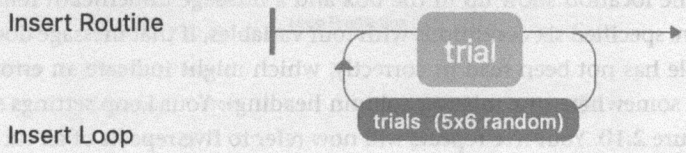


Figure 2.11 The Flow of the basic Stroop experiment after the Loop is added.

For most parameters that are set as a variable you also need to tell PsychoPy when to update those variables with a new value. The choice box to the right of the value allows you to select options like `constant`, `set every repeat` and `set every frame`. For the `Text` field of our Textbox (where we just wrote `$word`) we will need to set this option to `set every repeat`, because we want the written word to change on each trial (i.e. on every iteration of our Loop).

In this study you also need to set the `Text Color` of your text to `$letterColor` so that it also uses the value of your `letterColor` variable in each trial. Set this to be `set every repeat` as well.

At this point you could run your experiment again to see if it still works and if the stimulus is correctly changing its text and its letter color on each trial.

Update the Correct Answer on Every Trial

You also need to update your Keyboard Component because the correct answer was also changing using a variable in your conditions file. Click on your Component (it was called `resp` if you stuck carefully to the instructions) to edit its parameters. You need to tick

Pro Tip

Save and run often while creating a study

You may notice we've been saving and rerunning the study rather often. The advantage of doing this is that if something doesn't work, you've got a relatively small number of things to check in order to work out what broke your experiment (i.e. only the things since it was last working).

While running your experiment for debugging you might want to set the number of repeats to fewer than normal so it doesn't take 15 minutes to test each time. Also, if there are sections that you don't want to run (e.g. the training or adaptation period) you can set a Loop to have zero repeats and that will cause the contents of that Loop not to run at all.

the box that says `Save correct response`, which can be found in the `Data` tab, and this will cause a new box to appear in which you can tell PsychoPy what the correct response is. Based on your conditions file you want this to be `$corrAns`. Again, it's really important that you use the `$` symbol here or PsychoPy will look for the key literally called 'corrAns' and won't find one, so all your responses will appear to be 'incorrect'.

At this point your `stim` settings and your `resp` settings should look like those in Figures 2.12 and 2.13.

2.7 ADD SOME INSTRUCTIONS

By the time your experiment is running, with the stimulus being updated correctly on each repeat and the keyboard signalling the end of the trial, you are almost finished. That is, your experiment is fully capable of running and demonstrating the Stroop effect. It could be made a lot nicer, though. For instance, it really helps to have some text at the beginning to remind you of the keys to press and enable you to get ready before you start the trials (if you press `OK` in the dialog box and it goes straight to the first trial it can be off-putting).

Just click `Insert Routine` in the Flow panel and select the option called `(new)`. You'll be asked to provide a name for the new Routine. Call it `instructions` and then you can set it to appear on the Flow just before the `trials` Loop starts (Figure 2.14). Alternatively, you can do this in two steps: the `Experiment` menu has an entry called `New Routine` and you can use the Flow to insert a Routine that you previously created (including one that is already somewhere else on the Flow).

Now click on that Routine (either in the Flow or in the `Routines` panel at the top) so as to switch to editing it.

As you did in the `trial` Routine, you need to add a Textbox Component to the `instructions` Routine. Of course, it can't have the same name as the `instructions` Routine so you'll have to call the Textbox Component something else, like `instrText`. You also want to let your participants read the text in their own good time and then press a button when they're ready to get started. So set the duration to be infinite. For the text itself, it helps to tell your participants what to do and what keys to press, but also tell them to 'press a key when ready to start'. So, for this Component these are the key things to change:

- `Name = instrText`
- `Start [time (s)]= 0`
- `Stop [duration (s)]` = blank, i.e. infinite
- `Text` = something useful here about the keys to press

Of course, because we made the text last forever we need to provide something to end the Routine or we'll never get to the main experiment!

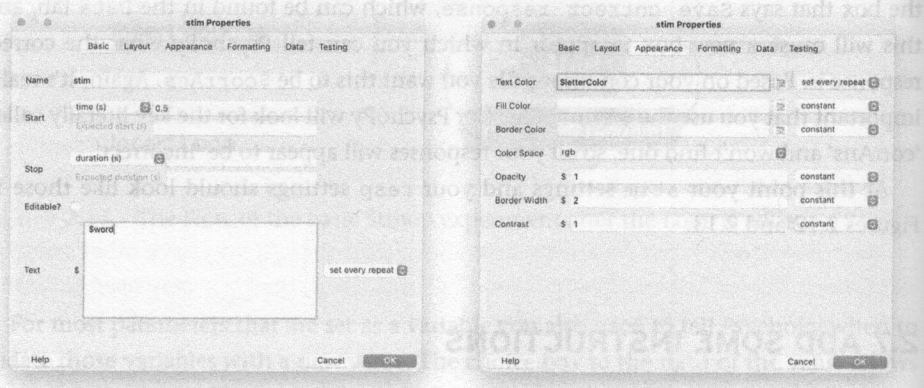


Figure 2.12 The final state of the Textbox Component in the `Basic` (left) and `Appearance` (right) settings. Note the `$` symbols where we've inserted variable names and the `set every repeat` option for the variable parameters.

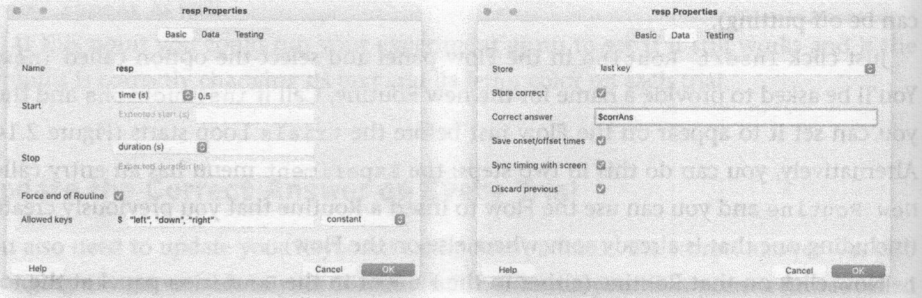


Figure 2.13 The Keyboard Component in its final state. Note in the `Basic` settings the quote marks around the key names (these can be single- or double-quote symbols, but they must match). Also make sure in the `Data` settings that you insert `$corrAns` and not simply `corrAns`, or PsychoPy will check whether the participant pressed the key with the name 'corrAns', and there is no such key!

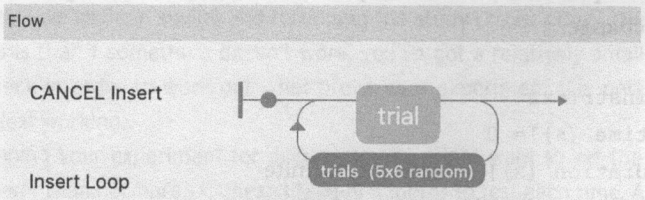


Figure 2.14 Adding another Routine just before the `trials` Loop, to hold our instructions.

Add a Keyboard Component with parameters like these:

- `Name` = `ready`
- `Save` = nothing
- `Stop [duration (s)]` = blank, i.e. infinite
- `Force end of Routine` = `ticked i.e. True`
- `Allowed keys` = blank, i.e. any keys

Again, make sure the duration of your keyboard is blank. If the keyboard stops being checked after 1 second then PsychoPy will never spot that the key has been pressed. The user will have to press the `escape` key and quit the study.

2.8 ADD A THANK-YOU SLIDE

A similar touch that isn't *strictly* necessary is to add something at the end of your study to say thanks. Partly this is just a nice thing to do, but also, if at the end of the study there is no message and the screen suddenly disappears, your participants will be surprised and may think the experiment 'crashed'. You'll get lots of concerned participants thinking they broke your study! A message that says 'Thank you for taking part' reduces this.

This time you probably don't have so much to say and you don't need to leave your participants unlimited time to read the brief text, so you could just put it on screen for 2 or 3 seconds and not bother with a Keyboard Component to end the thanks screen.

So go ahead and try to do these tasks:

- Add a new Routine called `thanks` that comes after your `trials` Loop.
- Go to that Routine and add a Text Component called `thanksText` with a simple thank-you message.
- Set the Textbox Component to start immediately and last for 2 s.

2.9 CHANGING YOUR INFO DIALOG

In the Experiment Settings you can alter the information you collect at the beginning, as well as a variety of other options. For instance, the value called `session` in the dialog box at the beginning of your study could be used to study which session someone was in if you run your study on different days, but if you only run your study once per participant then it seems unnecessary. We don't care (do we?) about the session number for this study so we could remove that row, but we do care about storing demographics, like the participant's age for the sake of our report/publication.

Click on the Experiment Settings icon  and rename the field called `session` so that it is called `age (years)`. If you like, you can provide a default value (e.g. 20) which

shows the participant what format you expect. Do keep the `participant` setting (it is used in creating the data filename for each participant).

Imagine you want to have two 'groups' of participants, group A and group B. You could add a field for participant group (we'll look at that in more depth in Chapter 10). Just click one of the 'plus' symbols to add a row. Anything you put into these fields is stored with your data file. If you want to make a 'dropdown' option you can give this field a list ['A', 'B']. Is there anything else you want to be kept alongside the data?

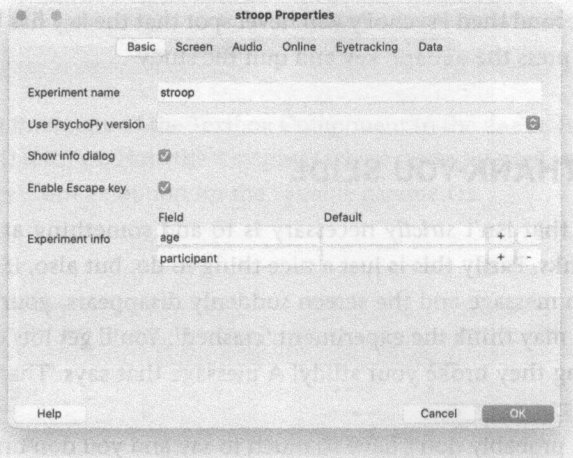
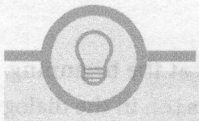


Figure 2.15 Experiment Settings for the Stroop task with no 'session' information but with an `age` field added. It's a good idea always to leave the `participant` field as is.

The Experiment Settings dialog also allows you to control other aspects of the screen, such as:

- whether or not the mouse is visible
- what color the background of the screen should be
- whether it should be full-screen mode or shown in a window.



Pro Tip

Debug your experiment in window mode, not full-screen

When the experiment is in full-screen mode it can be hard to escape if something goes wrong (e.g. your experiment gets stuck in a cycle that it can't get out of or loads a file that is taking a lifetime). You can usually force-quit the application using your computer's 'task manager', but that's annoying. If you stay in window mode until your experiment is definitely working then you can always hit the red stop button to force it to quit.

2.10 ANALYZE YOUR DATA

Data File Types

The Experiment Settings dialog box also allows you to control the data file outputs. The main file output is the trial-by-trial `.csv` file, and this is all you generally need (it opens in Excel or it can be imported into R or most other statistics packages).

- *Don't* bother about summarized formats and Excel (native) files. They don't add information and can slow things down.
- *Do* keep the `log` file and the `psydat` file. You might not need them, but they're a safety net in case things go wrong. They exist in PsychoPy for historical reasons.

The format of the filename (and folder to which it is saved) can also be changed, but that's easy to break if you don't know what you're doing. You can read about that in Section 17.6.

When we run the experiment, PsychoPy will create a folder called `data` next to your `.psyexp` experiment file. It will contain three files for each run of the study:

- `csv` file. This is what you'll most likely use for analysis. The data are stored in columns of values and with rows representing the trials as they occurred (in chronological order). It can be opened in any spreadsheet application, like Excel, and this is what we'll focus on in the analysis below.
- `log` file (a text file). This is a file containing lots of detail. Every time something changes in the experiment it sends a message to the log file with a time-stamp and in chronological order. This is handy for checking that your stimuli had good timing or to find data that you didn't think you would need, but isn't easy for normal analysis because it has far too much information.
- `psydat` file. This is a file for doing analyses in Python scripts. It isn't a file you can easily 'read'. It doesn't double-click to open in an application (currently), but it contains a great deal of information, including all the data to re-create the `.csv` file if you accidentally lose/damage it, or a copy of the experiment you originally ran. Unfortunately, it currently requires that you write Python code in order to load/use the file. Keep these files – they potentially rescue you if other things go wrong.

Pro Tip

Applications for opening the log file

The log file is simple text (tab-delimited). In macOS it opens in the Console by default, which works fine. In Windows it opens in the `Notepad` application by default, which is horrible. `Notepad` doesn't recognize the line-end characters in the log file, which makes it unreadable by most humans! You could install `Notepad++` instead, as a much better free alternative (https://notepad-plus-plus.org/) to `Notepad`, or you could open the file in a spreadsheet package like Excel.

Analyzing the .csv File

Let's open the most recent `.csv` file that you saved. If you gave yourself a participant name when you started the experiment it will now show up in the filename. Otherwise you may need to simply choose the most recent file. You could actually get rid of all the earlier files from when you were simply testing whether the experiment ran properly and didn't really run any trials. (For the future you might want to give the participant name as `test` or just leave it blank when testing the experiment so that it's easy to see which data files really contain data and which are from debugging your experiment.)

You could try double-clicking it and hopefully it will open directly in your spreadsheet application. We analyze this assuming you have Excel, but all the features we will use (sorting by column and calculating an average) should be available in any package.

Once you've opened your data file in Excel the first thing to do is to *save it straight away as an Excel workbook*. The `.csv` format is useful in that it's very simple and can be used by many software applications, but `.xlsx` files in Excel are more powerful in the things they can store, like graphs. Using this method (starting with a `.csv` file but then analyzing with an `.xlsx` file) has the advantages that:

- Excel will not keep asking if you want to change the format
- you can add things like graphs and formulas and they will get saved
- if you mess up your analysis here the original `.csv` file will still be there for you to go back to.

Let's look at the data, with an example file shown in Figure 2.16. Each row represents one trial, organized chronologically, and each column represents a different variable. Some of these variables (the first four in this experiment) came from your conditions file, telling you in each trial what the letter color and word were. Next come some columns containing information about the Loop and trial number:

- `trials.thisRepN`: how many of the five repeats in the `trials` Loop have finished (starts at zero)
- `trials.thisTrialN`: how may trials have occurred within this 'repeat', starting at zero
- `trials.thisN`: how many trials have completed in total, starting at zero
- `trials.index`: which line in the conditions file this condition came from – and, yes, this also starts at zero!

After those Loop information columns we have three columns referring to the participant's responses:

- `resp.keys`: the key the participant pressed in the `resp` Keyboard Component (or keys if we allow multiple keypresses). This will be empty if no response was detected.

Pro Tip

Expanding columns in Excel

In Excel, if you cannot see all of the data in the heading then you may need to expand the width of the column. If you double-click on the line between a pair of columns it will expand that column to the width of the widest cell.

In some data sets you may find cells, or often an entire column, that appear to be ########. Don't panic, your data are probably still there - this isn't a PsychoPy bug or a mistake on your part. This is what Excel does when it has a number that's too long to display in a column. Again, try expanding the column and seeing if the real numbers appear.

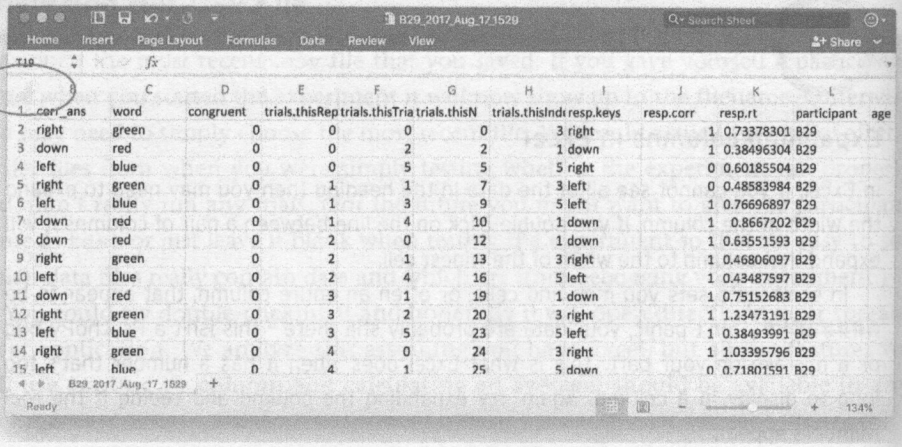
- `resp.corr`: stores whether or not that answer was considered 'correct' according to your conditions file. Hopefully this will have mostly ones (correct) in it with maybe the occasional zero (incorrect). If it shows up as all zero then check you used `$corrAns` rather than simply `corrAns` in your Keyboard Component.
- `resp.rt`: the time in seconds since the beginning of the Keyboard Component which, for most purposes, indicates response time.

Lastly there are some columns that are recorded from the information dialog box at the beginning of the run (age, participant, etc.) and some additional information about the computer (e.g. the frame rate of the monitor).

For this particular analysis all you would need to do is calculate the average reaction time of the congruent and incongruent trials. You could do that with a calculator, but it will be easier using Excel's `average` function.

First, sort your data by whether the condition was a *congruent* one, which we conveniently stored in our file. To do this make sure you select all the data first (if you accidentally sort your data while only a few columns are selected then those columns will no longer match up with the order of the others). You can select all by pressing `Ctrl-A` (`Cmd-A` on a Mac) or by pressing the square in the top left of the spreadsheet table, as in Figure 2.16.

Hopefully all your data will move into position, with the top half being incongruent trials and the bottom being congruent (or vice versa). Now we can simply take the average of the `resp.rt` column for the top half and compare it with the bottom. Click on a cell somewhere in the spreadsheet where you want to store the average of the incongruent trials and create an average function there. There are a few options for how to do this. The first two may seem appealing because they involve clicking on buttons rather than typing, but from one version of Excel to another the items never seem to stay in the same place, so you might do better learning the third (typing) option:



B29_2017_Aug_17_1529

Home Insert Page Layout Formulas Data Review View

	B	C	D	E	F	G	H	I	J	K	L	
1	corr_ans	word	congruent	trials.thisRep	trials.thisTria	trials.thisN	trials.thisIndi	resp.keys	resp.corr	resp.rt	participant	age
2	right	green	0	0	0	0	3	right	1	0.73378301	B29	
3	down	red	0	0	2	2	1	down	1	0.38496304	B29	
4	left	blue	0	0	5	5	5	right	0	0.60185504	B29	
5	right	green	0	1	1	7	3	left	0	0.48583984	B29	
6	left	blue	0	1	3	9	5	left	1	0.76696897	B29	
7	down	red	0	1	4	10	1	down	1	0.8667202	B29	
8	down	red	0	2	0	12	1	down	1	0.63551593	B29	
9	right	green	0	2	1	13	3	right	1	0.46806097	B29	
10	left	blue	0	2	4	16	5	left	1	0.43487597	B29	
11	down	red	0	3	1	19	1	down	1	0.75152683	B29	
12	right	green	0	3	2	20	3	right	1	1.23473191	B29	
13	left	blue	0	3	5	23	5	left	1	0.38493991	B29	
14	right	green	0	4	0	24	3	right	1	1.03395796	B29	
15	left	blue	0	4	1	25	5	down	0	0.71801591	B29	

B29_2017_Aug_17_1529 +

Ready 134%

Figure 2.16 An example data file from the Stroop task. You can select all the data (important if you plan to sort them) by pressing this lesser-known button at the top left (circled in red).

- Go to the 'ribbon' tab named `Formulas`, hit the button to `Insert` (probably on the left-hand edge) and select `Average` in the Statistical category. In the dialog that comes up you should be able to enter the cells whose average you want to find by dragging your mouse around them.
- *Or* go to the `Insert` menu and find the option `Function`. This should bring up roughly the same dialog as above and, again, you should select the cells you want to include in the average.
- *Or* if you can't find those options because Microsoft keeps moving them, you could simply type `=average` (then select the cells you want to include – or type them in if you prefer – and finally type the ending bracket) to complete your function.

Repeat this process for the *congruent* trials and see if the reaction time is faster (or shorter) for these trials than for the *incongruent* trials.

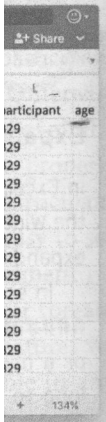
That's it: one study created, run and analyzed!

Exercises and extensions

Exercise 2.1: Change the screen color

Just for practice, try and change the color of the screen (for the entire experiment) to be black instead of gray. Also for practice set the color of the instructions text to red instead of white.

Solution: page 281



Exercise 2.2: Alter the language of the entire study

You could test how much the Stroop effect depends on the stimuli being in your first language. You could create a variant of the task in French, say, so the `word` variable would have values like `rouge`, `vert` and `bleu`. Does it still interfere? (Of course if English *isn't* your first language then you could now test in your first language!)

Solution: page 281

Exercise 2.3: Measure the reverse Stroop effect

You could run a similar study to look at the reverse effect. You've probably seen here that the meaning of the word interfered with your ability to report the letter color, but is there an equivalent effect in which the letter color interferes with your ability to read? Stroop's original paper suggested a lack of effect in this direction but, then again, he was running his study with paper and ink and asking participants to call out the answers while he timed them on a stopwatch. Maybe with the somewhat more precise timing of modern computers we can reveal such an effect.

Solution: page 281

Exercise 2.4: Check out the full experiment

There is also an extended version of this task included in the PsychoPy demos, which adds a feedback Routine and some training trials, with the feedback being used only in the training period. Trying to create this from scratch is probably too much right now, but you might find it useful to take a sneak peak at what you'll be learning in the future!

Solution: page 282