

Collecting Data with Free Software

Henrik Singmann

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Free Software (in the GNU sense)



- Free Software is first of all not only free of charge but gives you the freedom to do what you want with your software:
- *Freedom 0*: The freedom to run the program for any purpose.
- *Freedom 1*: The freedom to study how the program works, and change it to make it do what you wish.
- *Freedom 2*: The freedom to redistribute copies so you can help your neighbor.
- *Freedom 3*: The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits.

- Responses from a limited number of participants to a limited number of items
- Items constructed to answer the research questions
- Experimental research: participants and/or items are randomly assigned
- Items share similarities: e.g., item or response format, topic, layout ...
- Further information from participants: e.g., demographic information, prior knowledge regarding the research question, ...
- Various ways to collect such data:
 - Paper Questionnaires
 - Web Based Experiments
 - Computerized Lab Experiments

- Responses from a limited number of participants to a limited number of items
- Items constructed to answer the research questions
- Experimental research: participants and/or items are randomly assigned
- Items share similarities: e.g., item or response format, topic, layout ...
- Further information from participants: e.g., demographic information, prior knowledge regarding the research question, ...
- Various ways to collect such data:
 - Paper Questionnaires
 - Web Based Experiments
 - **Computerized Lab Experiments**

General Structure of an Experiment



Most experiments follow the following general structure:

1. Greeting, introduction and instruction to the experiment and task
2. Main task: repeatedly working on similar items
3. Demographics, debriefing, ...

Depending on the design, part 2 may consist of multiple blocks or different tasks intermixed with additional instructions.

Different types of Items



Two "types" of stimuli/items:

1. Items where exact presentation and exact timing is important. e.g.:
 - response times, eye tracking, or similar dependent variables
 - priming research (presenting stimuli very brief)
 - psychophysical research (random dots)
2. Items that contain a lot of (semantic) information and the response should reflect deliberate cognitive processes.

Different types of Items



Two "types" of stimuli/items:

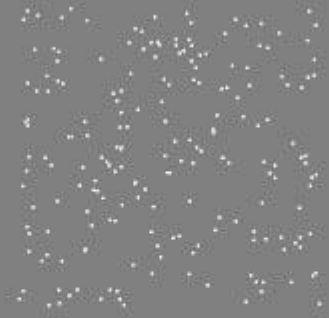
1. Items where exact presentation and exact timing is important. e.g.:
 - response times, eye tracking, or similar dependent variables
 - priming Research (presenting stimuli very brief)
 - psychophysical research (random dots)
2. **Items that contain a lot of (semantic) information and the response should reflect deliberate cognitive processes.**



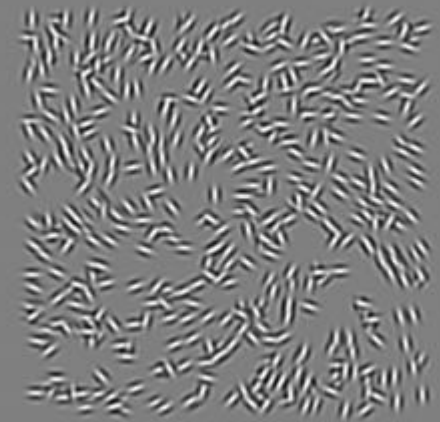
Example Experiment

- Free software for "the presentation of stimuli and collection of data for a wide range of neuroscience, psychology and psychophysics experiments".
- Mainly developed for presenting "type 1" stimuli, but I will show how to easily present "type 2" stimuli.
- PsychoPy uses the Python programming language: a free "general-purpose, interpreted high-level programming language whose design philosophy emphasizes code readability" (wikipedia)

PsychoPy: Example "Type 1" Stimuli



Hit Q to quit

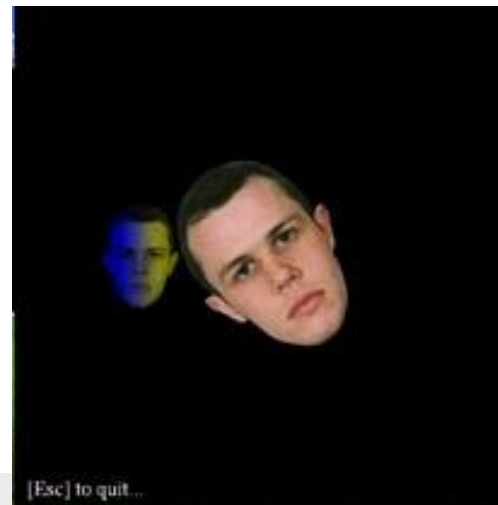


PsychoPy
©Jon Peirce

unicode (eg Ψ Η Σ)

Fonts
rotate!

Very long
sentences
can wrap



[Esc] to quit...

PsychoPy has two different interfaces:

- The builder view: Build an experiment using a graphical user interface (i.e., via click and play)
- The coder view: Build an experiment using Python code.

PsychoPy is very good in controlling the main task of an experiment but not so comfortable when it comes about instruction screens and collecting demographic data.

The Builder View



The screenshot displays the PsychoPy Builder interface for a stroop experiment. The main window is titled "stroop.pyexp - PsychoPy Builder" and features a menu bar (File, Edit, Tools, View, Experiment, Demos, Help) and a toolbar with various icons for file operations and execution.

The central area shows a timeline labeled "t (sec)" with markers at 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, and 1.4. Two blue horizontal bars represent the duration of the "word" and "resp" components, both starting at 0.5 seconds and ending at 1.5 seconds. To the left of the timeline, there are icons for "word" (a red 'T' on a black 'T') and "resp" (a keyboard icon).

On the right side, a "Components" panel lists various elements available for the experiment, including a landscape image, a blurred image, another landscape image, a keyboard icon, a red 'T' on a black 'T' icon, a code block with the text `y=sin(x)`, `x=random()`, and `f='tst.jpg'`, a stack of images, a "Rating" component, and a graph icon.

At the bottom, a "Flow" diagram illustrates the sequence of events. It starts with an "instructPractice" component (red rounded rectangle). This is followed by a loop labeled "practice" containing a "trial" component (red rounded rectangle) and a "feedback" component (green rounded rectangle with "1.00s" below it). After the "practice" loop, there is an "instruct" component (red rounded rectangle), followed by another loop labeled "trials" containing a "trial" component (red rounded rectangle). The flow concludes with a "thanks" component (green rounded rectangle with "2.00s" below it). On the left side of the flow diagram, there are options for "Insert Routine" and "Insert Loop".

The Builder View



The screenshot shows the PsychoPy Builder interface for a Stroop task. The main window displays a flowchart with the following sequence of components: **instructPractice** (red), a loop containing **trial** (red) and **feedback (1.00s)** (green) labeled "practice", **instruct** (red), another loop containing **trial** (red) labeled "trials", and **thanks (2.00s)** (green). The right-hand side features a "Components" panel with various icons for adding elements like images, text, and code. The top menu includes File, Edit, Tools, View, Experiment, Demos, and Help. The title bar reads "stroop.pyexp - PsychoPy Builder".

- Good for items which consist of a small number of single elements (i.e., not too much text).
- The builder view can be used to easily develop classical psychological "type 1" paradigms such as e.g., the Stroop task.

```

K:\GitHub\psytml\examples\psychopy\PsychoPy1.py - PsychoPy Coder
File Edit Tools View Demos Help
PsychoPy1.py x PsyTML.py
1 #!/usr/bin/env python
2 #-*-coding:utf-8-*-
3
4 ## Note, you want to change the number, name, and size of the screen before running this!
5
6 ## import all libraries that are needed
7 from psychopy import visual, event, core, monitors .....# needed for PsychoPy functionality
8 import os .....# needed for constructing well-formed paths to HTML files
9 from PyQt4 import QtGui .....# needed for initializing Qt
10 import sys .....# needed only for initializing the QT App
11 from numpy import random as rd .....# needed for randomizing the trial order
12 import PsyTML .....# needed to present the HTML pages
13 import helper .....# needed for getting the id, condition and writing data.
14
15 ## Parameters for the experiment
16 expName = "example1"
17 bgColor = "#c0c0c0"
18 data_folder = "data"
19
20 data = [] .....# empty list that will collect the data per trial (as Python dictionaries)
21
22 pathToldFile = "." .....# set the (preferably network) path to the file containing the Ids and Conditions.
23
24 ## Get Participant Number and initialize everything
25 (id, condition) = helper.getld(pathToldFile) .....# get participant number and condition
26
27 app = QtGui.QApplication(sys.argv) .....# initialize QT App

```

Shelf

Output Shell

```

##### Running: K:\GitHub\psytml\examples\psychopy\PsychoPy1.py #####
Traceback (most recent call last):
  File "K:\GitHub\psytml\examples\psychopy\PsychoPy1.py", line 80, in <module>
    helper.writeComments(comments, data_folder, id, expName)
  File "K:\GitHub\psytml\examples\psychopy\helper.py", line 95, in writeComments
    if (comment["comment"] != ""):
KeyError: 'comment'

##### Running: K:\GitHub\psytml\examples\psychopy\PsychoPy1.py #####
[{'u'power': u'strong', u'resp': u'-2', u'Next': u'Next', u'content': u'obesity', 'trial': 1, 'id': 2, u'causal':
u'non_causal', 'condition': 'noncausal'}, {u'power': u'strong', u'resp': u'3', u'Next': u'Next', u'content':
u'sexual_lyric', 'trial': 2, 'id': 2, u'causal': u'non_causal', 'condition': 'noncausal'}]
{u'glasses1': u'glasses_no', u'age': u'23', u'education': u'highschool', u'sex': u'female'}
{u'comment': u''}

```

- Good for doing whatever you want to do and is not possible with other methods.
- We will learn how to use the coder view to control the structure of the experiment, collect the data, present the stimuli (using add-on PsyTML), and do all necessary administrative stuff.
- We will design the ("type 2") items in html using an external easy to use WYSIWYG editor.

```
##### Running: K:\GitHub\psytml\examples\psychopy\PsychoPy1.py #####  
[{'u'power': u'strong', u'resp': u'-2', u'Next': u'Next', u'content': u'obesity', 'trial': 1, 'id': 2, u'causal':  
u'non_causal', 'condition': 'noncausal'}, {'u'power': u'strong', u'resp': u'3', u'Next': u'Next', u'content':  
u'sexual_lyric', 'trial': 2, 'id': 2, u'causal': u'non_causal', 'condition': 'noncausal'}]  
{u'glasses1': u'glasses_no', u'age': u'23', u'education': u'highschool', u'sex': u'female'}  
{u'comment': u''}
```



Designing "Type 2" Stimuli

html Form Elements



Example HTML Forms

Text:

Select:

Radio Button (select one):
 Yes
 No
 Indifferent

Radio Button 2:
Yes No Maybe

Checkbox (select multiple):
A B C

- add-on for PsychoPy developed by me
- presents html page and collects the status of form elements after clicking the Submit button
- html forms may contain any type of html form element
- data will be handed back to Python
- See: <https://github.com/singmann/psytml>

- KompoZer is a free WYSIWYG html editor (<http://www.kompozer.net/>)
- Layouting in html is done using tables with border = 0

Steps in creating a html page:

1. Create a 1 cell table with width of e.g., 800 pixel centrally.
2. Create a form inside the table using method "get".
3. Place all elements inside the form (headline, text, form elements, ...)
4. Use table (and tables inside tables) for nice layouting.
5. Add a "Submit" button.

Creating HTML forms



- html form elements have a name and a value
- The name is the variable name and the value the value of this variable in Python
- Of radio buttons with the same name only one can be selected
- Form elements other than radio buttons should have unique names

A brief introduction to HTML



- Instead of using the WYSIWYG editor, it is easy to change things in the html code.
- html is plain text accompanied with *markup tags*.
- markup tags are enclosed between `< >`: `<tag>`
- Usually you have an opening tag and a closing tag:
 - `<tag>` opens
 - `</tag>` closes
- html may look difficult in the beginning, but is easy to learn.

A barebone html page



- A html document is split into two parts: head and body
- The head contains meta information and the title
- The body contains what is visible.

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

Important html Tags



- a html form uses is enclosed by `<form>` and `</form>` (for PsyTML `method="get"` is mandatory)
- most form elements are created with `<input ...>`
- `<tr>` and `</tr>` create a table column, `<td>` and `</td>` a cell.
- `
` adds a line break/newline
- `<p>` `</p>` encloses a paragraph of text.
- `<div ...>` `</div>` is used for text formatting

- General formatting (background and other color, font type, font size, ...) is done using cascading style sheets (CSS). Add e.g. to the header:

```
<link rel="stylesheet" type="text/css" href="design.css">
```
- To make button presses mandatory, Javascript is necessary. (See the examples of PsyTML)
- PsyTML should allow Flash and other plugins (this behavior is untested)

Summary html



- html forms are a simple way to construct instruction screens or "type 2" items.
- html tutorials can be found all over the web, good places to start are e.g.: www.w3schools.com or www.boogiejack.com/html_tutorials.html (in German: selfhtml.org)
- KompoZer is a free comfortable editor (but seems unstable sometimes when opening more than one file)

- Each screen and item is a single html file (one could replace certain content, but ...)
- Instructions and other screens also need to contain a form and a Submit button (otherwise PsyTML does not know when the participant is done).
- Items are all of the same structure and contain EXACTLY the same form elements with the same names.
- `<input ... type = hidden>` can be used to code the item.



Transforming the Example Experiment into an own Experiment

The Example Experiment



- PsyTML contains the example experiment in the folder `examples/psychopy/`
- The main folder contains:
 - `.py` files containing the Python scripts
 - `id.lst` and `make.idLst.R`: for controlling which is the next participant
 - README files
- the `html` folder contains:
 - the `html` files representing all screens (instruction, items, ...)
 - the `css` files containing graphical parameters
 - `jsval.js`: a javascript library controlling required items
 - an `images` folder with some images
- The `data` folder contains the data after running the experiment

The example experiment



- PsyTML contains the example experiment in the folder `examples/psychopy/`
- The main folder contains:
 - `.py` files containing the Python scripts
 - `id.lst` and `make.idLst.R`: for controlling which is the next participant
 - README files
- **the html folder contains:**
 - **the html files representing all screens (instruction, items, ...)**
 - **the css files containing graphical parameters**
 - **`jsval.js`: a javascript library controlling required items**
 - **an images folder with some images**
- The data folder contains the data after running the experiment

Building your own Experiment I



1. Write instruction screens and end screens.
2. Develop a template of the item in the main task.
3. Make as many copies of this file as you need it.
4. Exchange the content and adapt set the appropriate hidden html forms to identify each item.
5. Feed the html files to PsychoPy.

Building your own Experiment I



1. Write instruction screens and end screens.
2. Develop a template of the item in the main task.
3. Make as many copies of this file as you need it.
4. Exchange the content and adapt set the appropriate hidden html forms to identify each item.
5. **Feed the html files to PsychoPy.**

```

K:\GitHub\psytml\examples\psychopy\PsychoPy1.py - PsychoPy Coder
File Edit Tools View Demos Help
[Icons]
PsychoPy1.py x PsyTML.py
1  #!/usr/bin/env python
2  #-*-coding:utf-8-*-
3
4  ## Note, you want to change the number, name, and size of the screen before running this!
5
6  ## import all libraries that are needed
7  from psychopy import visual, event, core, monitors .....# needed for PsychoPy functionality
8  import os .....# needed for constructing well-formed paths to HTML files
9  from PyQt4 import QtGui .....# needed for initializing Qt
10 import sys .....# needed only for initializing the QT App
11 from numpy import random as rd .....# needed for randomizing the trial order
12 import PsyTML .....# needed to present the HTML pages
13 import helper .....# needed for getting the id, condition and writing data.
14
15 ## Parameters for the experiment
16 expName = "example1"
17 bgColor = "#c0c0c0"
18 data_folder = "data"
19
20 data = [] .....# empty list that will collect the data per trial (as Python dictionaries)
21
22 pathToldFile = "." .....# set the (preferably network) path to the file containing the Ids and Conditions.
23
24 ## Get Participant Number and initialize everything
25 (id, condition) = helper.getld(pathToldFile) .....# get participant number and condition
26
27 app = QtGui.QApplication(sys.argv) .....# initialize QT App
[Shelf]
Output Shell
##### Running: K:\GitHub\psytml\examples\psychopy\PsychoPy1.py #####
Traceback (most recent call last):
  File "K:\GitHub\psytml\examples\psychopy\PsychoPy1.py", line 80, in <module>
    helper.writeComments(comments, data_folder, id, expName)
  File "K:\GitHub\psytml\examples\psychopy\helper.py", line 95, in writeComments
    if (comment["comment"] != ""):
KeyError: 'comment'

##### Running: K:\GitHub\psytml\examples\psychopy\PsychoPy1.py #####
[{'u'power': u'strong', u'resp': u'-2', u'Next': u'Next', u'content': u'obesity', 'trial': 1, 'id': 2, u'causal':
u'non_causal', 'condition': 'noncausal'}, {u'power': u'strong', u'resp': u'3', u'Next': u'Next', u'content':
u'sexual_lyric', 'trial': 2, 'id': 2, u'causal': u'non_causal', 'condition': 'noncausal'}]
{u'glasses1': u'glasses_no', u'age': u'23', u'education': u'highschool', u'sex': u'female'}
{u'comment': u''}

```


A very brief Introduction to Python I



- Python has a shell (`>>>`) so one can interact with Python directly.
- Variables are dynamically typed (i.e., type of variables does not need to be declared):

```
>>> x = 3
>>> type(x)
<type 'int'>
```
- Whitespace indentation delimit blocks (instead of `{}`):

```
>>> if (x > 0):
...     print("Yes")
... else:
...     print("no")
...
Yes
```
- four spaces (and not tab) represent the indentation to bind blocks of code together.

A very brief Introduction to Python II



Data types:

- Numbers:

```
>>> x = y = z = 0 #
```

```
Zero x, y and z
```

```
>>> x
```

```
0
```

```
>>> y
```

```
0
```

```
>>> z
```

```
0
```

```
>>> tax = 12.5 / 100
```

```
>>> price = 100.50
```

```
>>> price * tax
```

```
12.5625
```

- Strings (use ' ' or " "):

```
>>> word = 'Help' +  
'A'
```

```
>>> word
```

```
'HelpA'
```

```
>>> word[0]
```

```
'H'
```

```
>>> word[2:4]
```

```
'lp'
```

- Unicode Strings (u' '):

```
>>> y = u'Rüdiger'
```

```
>>> y
```

```
u'R\xfcddiger'
```

A very brief Introduction to Python III



- Lists: A list of comma-separated values between square brackets, that do not have to have the same type:

```
>>> a = ['spam', u'eggs', 100, 1234]
>>> a
['spam', u'eggs', 100, 1234]
```
- Like string indices, list indices start at 0, and lists can be sliced, concatenated and so on:

```
>>> a[1:-1]
[u'eggs', 100]
>>> a[1:]
[u'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', u'eggs', 123, 1234]
```

A very brief Introduction to Python IV



- Dictionaries: An unordered set of key: value pairs enclosed by braces {}.

```
>>> tel = {'jack': 4098, 'sape': 4139}
```

```
>>> tel['guido'] = 4127
```

```
>>> tel
```

```
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

```
>>> tel['jack']
```

```
4098
```

A very brief Introduction to Python V



- Python contains loops (repeated operations):



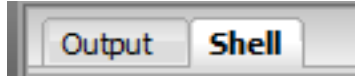
```
>>> y = 0
>>> for x in xrange(5):
...     y = y + x
...
>>> y
10
```

- Important keywords / elements in Python are:

- `import`: imports library
- `if / else / for`: control flow
- `.` : seperates functions from libraries; methods/elements from objects
- `#` : afterwords comes a comment

Summary Python



- Run a script in PsychoPy by pressing 
- Terminate a script by pressing 
- PsychoPy contains a Python shell (>>>): 
- Work through parts 3, 4, and 5 of "The Python Tutorial": <http://docs.python.org/tutorial/index.html>
- The important data types are strings (" "), unicode strings (u" "), lists ([]), and dictionaries ({ }) with their corresponding methods.
- Python runs a script from the first line to the end.

The example experiment



- PsyTML contains the example experiment in the folder `examples/psychopy/`
- The main folder contains:
 - `.py` files containing the Python scripts
 - `id.lst` and `make.idLst.R`: for controlling which is the next participant
 - README files
- the `html` folder contains:
 - the `html` files representing all screens (instruction, items, ...)
 - the `css` files containing graphical parameters
 - `jsval.js`: a javascript library controlling required items
 - an `images` folder with some images
- The `data` folder contains the data after running the experiment

The example experiment



- PsyTML contains the example experiment in the folder `examples/psychopy/`
- **The main folder contains:**
 - `.py` files containing the Python scripts
 - `id.lst` and `make.idLst.R`: for controlling which is the next participant
 - **README files**
- `PsychoPy1.py` contains the code of the example experiment.
- `PsyTML.py` contains the code that displays the html pages.
- `helper.py` contains some helper functions (e.g., for writing the data).

Besides necessary technical stuff, the file contains:

- Parameters for the experiment such as name and path of the data folder.
- List of items for the main task (split for the two conditions)
- Presentation of instructions, items and end screen.
- Functions writing the data.

Data Examples



example1_1.rtd example1_demographics.dat

```
1 id→condition→trial→content→causal→power→resp→
2 1→causal→1→obesity→causal→strong→1→
3 1→causal→2→sexual_lyric→causal→strong→0→
4
```

example1_1.rtd example1_demographics.dat

```
1 1;causal;male;32;Please select;glasses_no;None;
2
```

How does the Data look like?



- Internally saved in a list of dictionaries with the same keys (dictionary keys are the names of the html forms)
- `helper.writeTrials()` writes data list to the data folder in a single file per participant (filename contains experiment name and id) separating values by argument `sep` (default separator is tab).
- Each trial occupies one row in the data file.
- the header argument defines which variables should be written (and the order of the variables.)

Demographic Data



- Demographic data of all participants is written to a single file.
- Each row corresponds to one participant and each new participant is simply appended to the file.
- Comments of the participants are added to a comments folder.

Data Examples



example1_1.rtd example1_demographics.dat

```
1 id→condition→trial→content→causal→power→resp→
2 1→causal→1→obesity→causal→strong→1→
3 1→causal→2→sexual_lyric→causal→strong→0→
4
```

example1_1.rtd example1_demographics.dat

```
1 1;causal;male;32;Please select;glasses_no;None;
2
```

How to determine Participant ID?



- `id.lst` is a text file containing participant number and condition, one per row
- * indicates that the id has been used
- `helper.getIdFile()` tries to open `id.lst` and returns the next id and condition and marks it with *
- When running experiments on multiple machines this file is best stored in a network folder
- `make.idList.R` contains an R script for creating files with randomized conditions

```
1 · causal · *
2 · noncausal · *
3 · causal ·
4 · noncausal ·
5 · causal ·
6 · noncausal ·
7 · causal ·
8 · noncausal ·
9 · causal ·
10 · noncausal ·
11 · causal ·
12 · noncausal ·
13 · noncausal ·
14 · causal ·
15 · noncausal ·
16 · causal ·
```

Building your own Experiment II



After creating the desired html files:

- Create a PsychoPy Monitor and set the correct screen size (<http://psychopy.org/general/monitors.html>)
- Change experiment name and set correct folders.
- Replace the references to the html files with your own html files.
- Add calls to `viewPsyTML` for additional screens, e.g.:
`intro.viewPsyTML("html/intro2.html")`
- Set the `data_header` list to match your variable names (i.e., html form names).
- Create your own `id.lst`

- PsychoPy and PsyTML provide a framework for running experiments with "type 2" items:
 - Create the screens and items using html forms.
 - Python and PsychoPy are used for running the experiment, PsyTML displays the forms and gathers the data.
- Learning basic html and Python is probably a good investment.