Programming for Cognitive and Brain Sciences (notes for the Cogmaster's PCBS course)

Christophe Pallier

Fall 2018

This document – a work in progress – is available at `https://chrplr.github.io/PCBS`. Its source as well as the course materials is on github, at `http://www.github.com/chrplr/PCBS`; to download them, open a terminal (`git bash` under Windows) and type:

```
        git clone https://github.com/chrplr/PCBS
''**

**Companion documents***:

* [how-to-solve-problems.md](how-to-solve-problems.md)
* [tools-for-reproducible-science.md](tools-for-reproducible-science.md)
* [additional exercises](exercices.md). I answer questions on a [slack discussion forum](https://cogmaster
 https://join.slack.com/t/cogmaster-pcbs/shared_invite/enQtNDUzNDk0NTMyNjk0LTM0YzVhMmI1YjU3ZjRhMjNmZDRjMmV
* [Suggestions for projects](ideas-for-projects.md)



<!-- markdown-toc start - Don't edit this section. Run M-x markdown-toc-refresh-toc -->
**Table of Contents**

- [License](#license)
- [Objective](#objective)
- [Prerequisites:](#prerequisites)
- [Resources](#resources)
    - [Manipulations:](#manipulations)
    - [Stimulus/Experiment generation modules](#stimulusexperiment-generation-modules)
    - [Data analyses, Statistics](#data-analyses-statistics)
    - [Simulations](#simulations)
    - [Relevant Books](#relevant-books)
- [Creating static visual stimuli](#creating-static-visual-stimuli)
    - [Kanizsa triangle](#kanizsa-triangle)
    - [Herman grid](#herman-grid)
    - [Ebbginhaus-Titchener](#ebbginhaus-titchener)
    - [Honeycomb and Extinction illusions.](#honeycomb-and-extinction-illusions)
- [Creating dynamic visual stimuli](#creating-dynamic-visual-stimuli)
    - [Wertheimer line-motion illusion.](#wertheimer-line-motion-illusion)
    - [Flash-lag illusion](#flash-lag-illusion)
```

```
- [Creating and playing sounds](#creating-and-playing-sounds)
    - [Sound localisation from binaural dephasing](#sound-localisation-from-binaural-dephasing)
    - [Pulsation (Povel & Essen, 1985)](#pulsation-povel--essen-1985)
- [Experiments](#experiments)
    - [Simple reaction times](#simple-reaction-times)
    - [Posner's attentional cueing task](#posners-attentional-cueing-task)
    - [Stroop Effect](#stroop-effect)
    - [Lexical Decision Task](#lexical-decision-task)
    - [A general audio visual stimulus presentation script](#a-general-audio-visual-stimulus-presentation-
    - [More examples using expyriment.org](#more-examples-using-expyrimentorg)
- [Lexical Statistics](#lexical-statistics)
    - [Zipf law](#zipf-law)
    - [Benford's law.](#benfords-law)
- [Simulations](#simulations-1)
    - [Cellular Automata](#cellular-automata)
    - [Artificial Neural networks](#artificial-neural-networks)
    - [Natural Language Parsing](#natural-language-parsing)

<!-- markdown-toc end -->

# License

All documents in this repository are distributed under the [Creative Commons Attribution-ShareAlike 4](htt


# Objective

The purpose of this lecture is to get students to learn to write clean and simple programs in order to sol


# Prerequisites:

* knowledge of basic programming concepts expressions, instructions, variables, lists, dictionaries, tests

  Complete beginners should look at [Code Academy's _Learn Python_ module](https://www.codecademy.com/lear

  Then, there is an excellent online course [Python 3 : des fondamentaux aux concepts avancés du langage](

  Good books to start with Python include:
  * [_Automate the boring stuff_](http://automatetheboringstuff.com/),
  * [_Think Python_](http://greenteapress.com/thinkpython2/),
  * [_Invent Your Own Computer Games with Python_ (4th Edition)](https://inventwithpython.com/invent4thed/
  * [_Apprendre à programmer avec Python 3_](http://inforef.be/swi/python.htm).
```

```
* know how to edit a text file (with a text editor like [atom](https://atom.io/)), how to open a terminal,

* know the basic usage of [Git](https://www.gitbook.com/), that is the commands‘git clone, git pull, git i

    * see [tools-for-reproducible-science.md]
    * <https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners>
    * <https://git-scm.com/book/en/v2/Getting-Started-Git-Basics>

# Resources

## Manipulations:

_Automate the boring stuff with Python_ by Al Sweigart (<http://automatetheboringstuff.com/>) is a great b

## Stimulus/Experiment generation modules

* <http://www.pygame.org>
     - Tutorial "PyGame Drawing Basics": <https://www.cs.ucsb.edu/~pconrad/cs5nm/topics/pygame/drawing/>
* <http://www.lexique.org>
* <http://www.expyriment.org> (my favorite)
     - Tutorial: <https://docs.expyriment.org/Tutorial.html>
* <http://psychopy.org>
     - Tutorial "Programming with PsychoPy": <https://www.socsci.ru.nl/wilberth/nocms/psychopy/print.php>
* <http://psychtoolbox.org/> (Matlab only)

## Data analyses, Statistics

* Modules: numpy, scipy, pandas, seaborn, statsmodel, sklearn
   - Data manipulation: <http://pandas.pydata.org/pandas-docs/stable/tutorials.html>
   - Plotting:
      - <http://matplotlib.org/users/pyplot_tutorial.html>
      - <https://seaborn.pydata.org/tutorial.html>

* _Scipy Lecture Notes_: <http://www.scipy-lectures.org/>
* _Think Stats_ by Allen B. Downey: <http://greenteapress.com/thinkstats2/>
* _Python Data Science Handbook_ by Jake VanderPlas: <https://jakevdp.github.io/PythonDataScienceHandbook>


## Simulations

* [_Think Complexity_](http://greenteapress.com/wp/think-complexity-2e/) by Allen B. Downey
* The [Brian spiking neural network simulator](http://briansimulator.org/)
* [Deep Learning for Natural Language Processing with Pytorch](https://pytorch.org/tutorials/beginner/deep
```

Relevant Books

- [_Programming Visual Illusions for Everyone_](https://www.programmingvisualillusionsforeveryone.online/)
- _Neural Data Science: A Primer with MATLAB and Python_ by von Erik Lee Nylen and Pascal Wallisch
- _Matlab for Brain and Cognitive Scientists_ and _Analyzing neural time series data_ by Mike X Cohen
- [_Python in Neuroscience_](https://www.frontiersin.org/research-topics/8/python-in-neuroscience)
- _Modeling Psychophysical Data in R_ by Kenneth Knoblauch & Laurence T. Maloney

---

Creating static visual stimuli

We are going to use [pygame](http://www.pygame.org). You can read a [quick introduction on drawing with py

Kanizsa triangle

0. Open the script [square.py](visual-illusions/square.py) that generates and displays a square.

1. Copy the script and rename it 'circle.py', then modify it to display a red circle

2. Now, modify the script to display Kanizsa's figures:

   ![Kanizsa square](images/Kanizsa-square.jpeg)

   ![Kanizsa triangle](images/Kanizsa1.png)

  You can find more examples of 'illusory contours' by googling.

  Check [my solution](visual-illusions/kanizsa-square.py)

Herman grid

Starting from the square.py script, write a program to display the [Herman grid](https://en.wikipedia.org/

   ![Hermann Grid](images/HermannGrid.png)

 Check [my solution](visual-illusions/grid.py)

Ebbginhaus-Titchener

Create the static [Ebbinghaus-Titchener stimulus](http://www.abc-people.com/illusion/illusion-3.htm#axzz5S

   ![Ebbinghaus illusion](images/ebbinghaus-titchener.png)

```
## Honeycomb and Extinction illusions.

![Honeycomb illusion](images/honeycomb.png)

- Watch [this video](https://www.youtube.com/watch?v=fDBYSFDXsuE)
- Check out [Bertamini, Herzog, and Bruno (2016). “The Honeycomb Illusion: Uniform Textures Not Perceived
 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5030753/pdf/10.1177_2041669516660727.pdf)
- Program the stimulus of the extinction illusion (the lines can be horizontal and vertical rather than ob

![Extinction illusion](images/extinction.png)

Check [my solution](visual-illusions/extinction.py)

- Try to program the honeycomb stimulus above (optional). A []solution psychopy](visual-illusions/Honeycom


# Creating dynamic visual stimuli

## Wertheimer line-motion illusion.

Check out [Jancke et al (2004) Imaging cortical correlates of
illusion in early visual cortex](http://www.cnbc.cmu.edu/cns/papers/nature02396.pdf). Program the stimulus

## Flash-lag illusion

Read about the [Flash-lag illusion](https://en.wikipedia.org/wiki/Flash_lag_illusion). Program the stimulu


# Creating and playing sounds

Install the _simpleaudio_ module if it is not already installed on your computer (check with ipython: ‘imp
    ‘‘‘
    pip install simpleaudio
    ‘‘‘


Run the quick check with ipython:

    ‘‘‘
    import simpleaudio.functionchecks as fc
    fc.LeftRightCheck.run()
    ‘‘‘
```

```
Check out [simpleaudio tutorials](https://simpleaudio.readthedocs.io/en/latest/tutorial.html)

Study [sound_synth.py](sound/sound_synth.py)‘.


## Sound localisation from binaural dephasing

Take a mono sound and create a stereo sound by progressively dephasing the two channels.

## Pulsation (Povel & Essen, 1985)

3. Create rhythmic stimuli as described in [Povel and Essen (1985) _Perception of Temporal Patterns_](http


# Experiments

## Simple reaction times

1. Write a script that presents a series of trials in which a dot or a cross is presented at the center of

    - Here is a [solution using pygame](reaction-times/simple-detection-visual-pygame.py). Run it and chec
    - Here is a [solution using expyriment](reaction-times/simple-detection-visual-expyriment.py).

    Run the previous script. Check the results file in the folder ‘data‘. Launch ipython in the ‘data‘ fol

    ‘‘‘
    import pandas as pd
    d = pd.read_csv('simple-detection... .xpd', comment='#')
    d.RT.mean()
    d.RT.std()
    d.RT[1:].mean()

    import matplotlib.pyplot as plt
    plt.hist(d.RT)
    ‘‘‘

3. Read <https://docs.expyriment.org/Tutorial.html> to understand the basic pinciples of expyriment. See ‘

4. Modify ‘simple-detection-visual-expyriment.py‘ to play a short sound (‘click.wav‘) in lieu of displayin

5. Modify the script to have 3 blocks of trials: one in which the target is visual, one in which it is aud

## Posner's attentional cueing task
```

Program [Posner's attentional cueing task](https://en.wikipedia.org/wiki/Posner_cueing_task) See solution

Stroop Effect

The [Stroop Effect](https://en.wikipedia.org/wiki/Stroop_effect) demonstrates the automaticity of reading.
Write a python script to create 4x8 cards for the task, avoiding repetitions of colors.

![Stroop card](images/stroop.png)

  You can read a tutorial on [how to display text with pygame](https://nerdparadise.com/programming/pygame

   * After trying to program it yourself, you can compare with [my solution](Stroop-effect/create_stroop_c

   * Run [stroop_task.py](Stroop-effect/stroop_task.py) and check the naming times in 'data'. Compute the

Lexical Decision Task

In a lexical decision experiment, a string of characters is flashed at the center of the screen and the pa

   * Using [lexical-decision/select-words-from-lexique.py] as an example, select 20 high frequency nouns,
   * generate 50 pseudowords using either [Lexique tools](http://www.lexique.org/toolbox/toolbox.pub/) or
   * Program a lexical decision using expyriment.
   * Run it and compute the average decision times using pandas

A general audio visual stimulus presentation script

See <https://www.github.com/chrplr/audiovis>

More examples using expyriment.org

* See <http://docs.expyriment.org/old/0.9.0/Examples.html>
* Fork <https://github.com/expyriment/expyriment-stash> and contribute by adding new scripts!

Lexical Statistics

Zipf law

* The script (word-count.py])[Zipf/word-count.py] computes the distribution of frequencies of occurences i

   Note: To remove the punctuation, you can use the following function:

```
import string
def remove_punctuation(text):
    punct = string.punctuation + chr(10)
    return text.translate(str.maketrans(punct, " " * len(punct)))
```

"""

- Zipf law states that the product rank X frequency is roughly constant. This 'law' was discovered by Estoup and popularized by Zipf. See `http://en.wikipedia.org/wiki/Zipf%27s_law`. Create the Zipf plot for the text of *Alice in Wonderland* showing, on the y axis, the log of the frequency and on the x axis the word rank (sorting words from the most frequent to the least frequent).
- Display the relationship between word length and word frequencies from the data in `lexical-decision/lexique382-reduced.txt`
- Generate random text (each letter from a-z being equiprobable, and the spacecharacter being 8 times more probable) of 1 million characters. Compute the frequencies of each 'pseudowords' and plot the rank/frequency diagram.
- To know more about lexical frequencies:
  - Read Harald Baayen (2001) *Word Frequency Distributions* Kluwer Academic Publishers.
  - Read Michel, Jean-Baptiste, Yuan Kui Shen, Aviva P. Aiden, Adrian Veres, Matthew K. Gray, The Google Books Team, Joseph P. Pickett, et al. 2010. "Quantitative Analysis of Culture Using Millions of Digitized Books." Science, December. `https://doi.org/10.1126/science.1199644`. (use scholar.google.com to find a pdf copy). Check out **google ngrams** at `https://books.google.com/ngrams`. (Note that at the bottom of the page, there is a message "Raw data is available for download here").

Benford's law.

Learn about Benford's law. Write a Python script that displays the distribution of the most significant digit in a set of numbers. Apply it to the variables in `Benford-law/countries.xlsx`.

A solution: `Benford-law/Benford.py`

Simulations

Cellular Automata

Learn about Conway's Game of Life. Watch this and that videos.

- Implement an Elementary cellular automaton. The aim is to reproduce the graphics shown at the bottom on the previous page. you can take inspiration from the excellent *Think Complexity* by Allen B. Downey. My solution is at `cellular-automata/1d-ca.py`.
- Implement the Game of Life in 2D.
- Going futher: If you enjoy Cellular Automata, you can read Stephen Wolfram's *A New Kind of Science*. A more general book about Complexity is Melanie Mitchell's *Complexity: a guided tour*.

Artificial Neural networks

To understand the basics of artificial neural networks, I recommend thayt you watch the four excellent videos at `https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi` . The last two of them focus on the backpropagation algorithm that allow to train network to learn mapping.

Next, you can read and try to understand this implementation of the backpropagation algorithm.

Then, see a modern and efficient implementation of neural networks: `https://pytorch.org/tutorials/beginner/deep_learning_nlp_tutorial.html`

More readings:

- The Unreasonable Effectiveness of Recurrent Neural Networks on Andrej Karpathy's blog.
- understanding LSTM Networks
- Pattern recognition and machine learning by Christopher M. Bishop

Natural Language Parsing

Parsing refers to building the syntactic structure of a sentence from the linear sequence of words that compose it. Explore the various parsing algorithmsusing the Natural Language Toolkit.