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

Christophe Pallier

Fall 2018

This document 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
```

A companion document is exercises. I answer questions on discussion forum on slack Please join !

Table of Contents

License

All documents in this repository are distributed under the Creative Commons Attribution-ShareAlike 4. The code is distributed under the GPL v3 LICENSE.

Objective

The purpose of this lecture is to get students to learn to write clean and simple programs in order to solve tasks that are typically encountered in cognitive or neurosciences (data manipulation and analysis, creation of stimuli, programming of real time experiments, simulations. . .).

Prerequisites:

- knowledge of basic programming concepts expressions, instructions, variables, lists, dictionaries, tests (if..then..else), string manipulations, loops (while and for), functions (call and definition), file input/output operations) and their implementation in Python 3.

  Complete beginners should look at Code Academy's *Learn Python* module.

  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*,
  – *Think Python*,
  – *Invent Your Own Computer Games with Python* (4th Edition),
  – *Apprendre à programmer avec Python 3*.

- know how to edit a text file (with a text editor like atom), how to open a terminal, navigate the directory structure with 'cd', execute a .py script and launch ipython. A very useful read is *Learning the bash shell*.

- know the basic usage of Git, that is the commands`git clone, git pull, git init, git add, git status, git commit`.

  – 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 book to learn to manipulate files, extracting information from web pages, etc.

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* by Allen B. Downey
- The Brian spiking neural network simulator
- Deep Learning for Natural Language Processing with Pytorch

Relevant Books

- *Programming Visual Illusions for Everyone* by Marco Bertamini:
- *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*
- *Modeling Psychophysical Data in R* by Kenneth Knoblauch & Laurence T. Maloney

---

Creating static visual stimuli

We are going to use pygame. You can read a quick introduction on drawing with pygame.

0. Open the script 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. Make another of the script and modify it to display Kanizsa's figures:

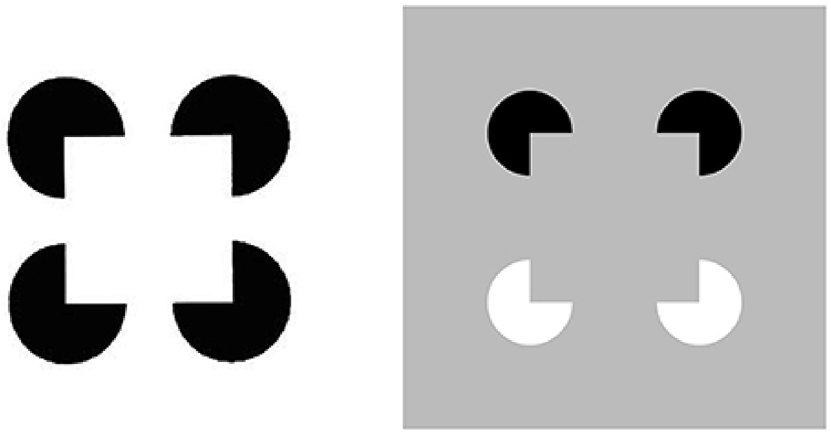


Figure 1: Kanizsa square

   (to know more, google 'illusory contours')
   Check my solution

3. Copy this script and modify it to display the Herman grid

   Check my solution

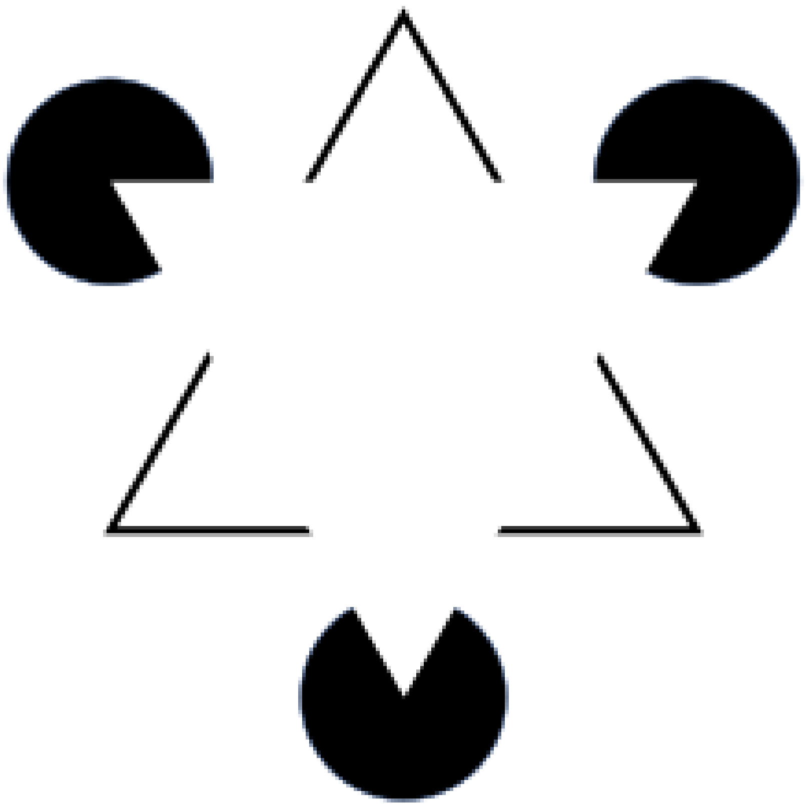


Figure 2: Kanizsa triangle

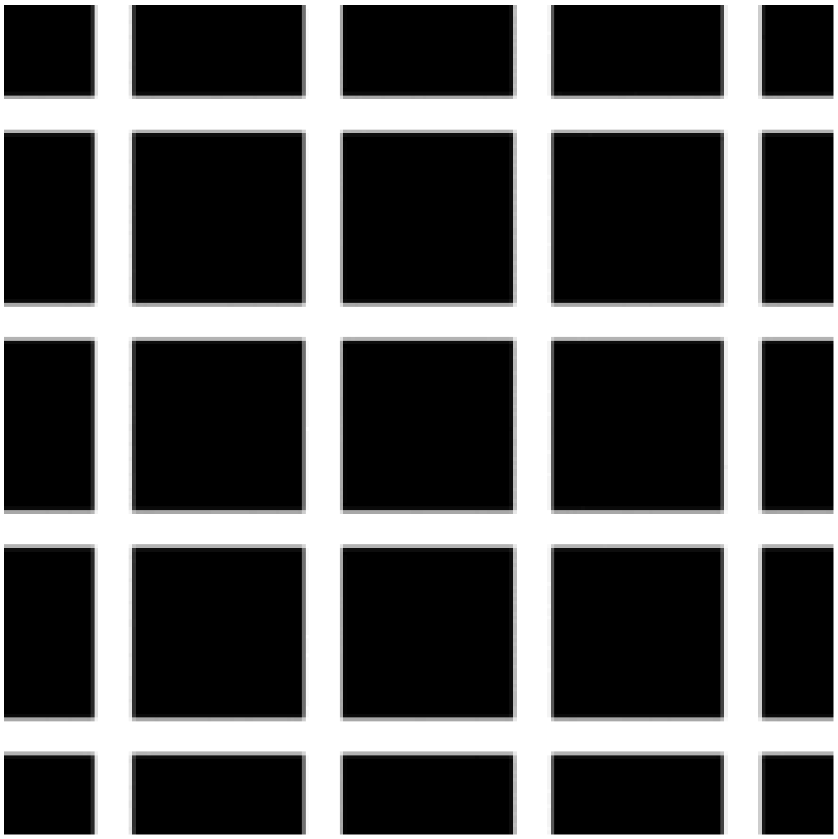


Figure 3: Hermann Grid

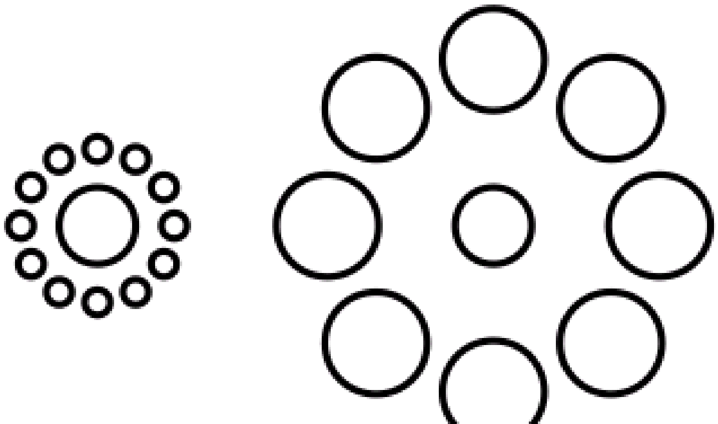


Figure 4: Ebbinghaus illusion

4. Copy this script and modify it to generate the static Ebbinghaus–Titchener stimulus. You can also watch this video.

5. Honeycomb and Extinction illusions.

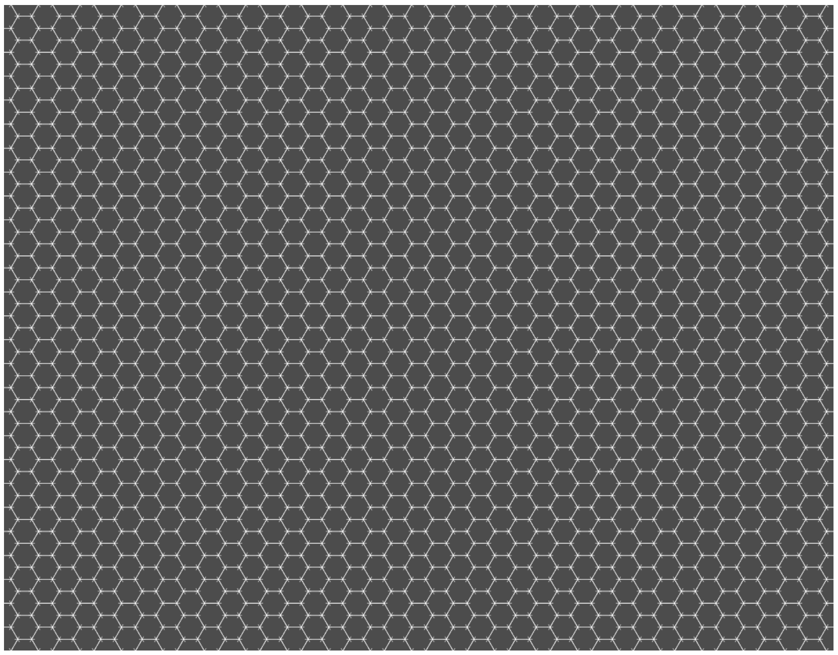


Figure 5: Honeycomb illusion

  - Watch this video
  - Check out Bertamini, Herzog, and Bruno (2016). "The Honeycomb Illusion: Uniform Textures Not Perceived as Such."
  - Program the stimulus of the extinction illusion (the lines can be horizontal and vertical rather than oblique)

  Check my solution

- Try to program the honeycomb stimulus above (optional). A implementation with psychopy is available from (Bertamini's web site)[`https://www.programmingvisualillusionsforeveryone.online/scripts.html`]

Creating dynamic visual stimuli

- Wertheimer line-motion illusion. Check out Jancke et al (2004) Imaging cortical correlates of illusion in early visual cortex. Program the stimulus. Compare with `visual-illusions/line-motion.py`

- Read about the Flash-lag illusion. Program the stimulus. Compare with [v]isual-illusions/flash-lag.py](isual-illusions/flash-lag.py)

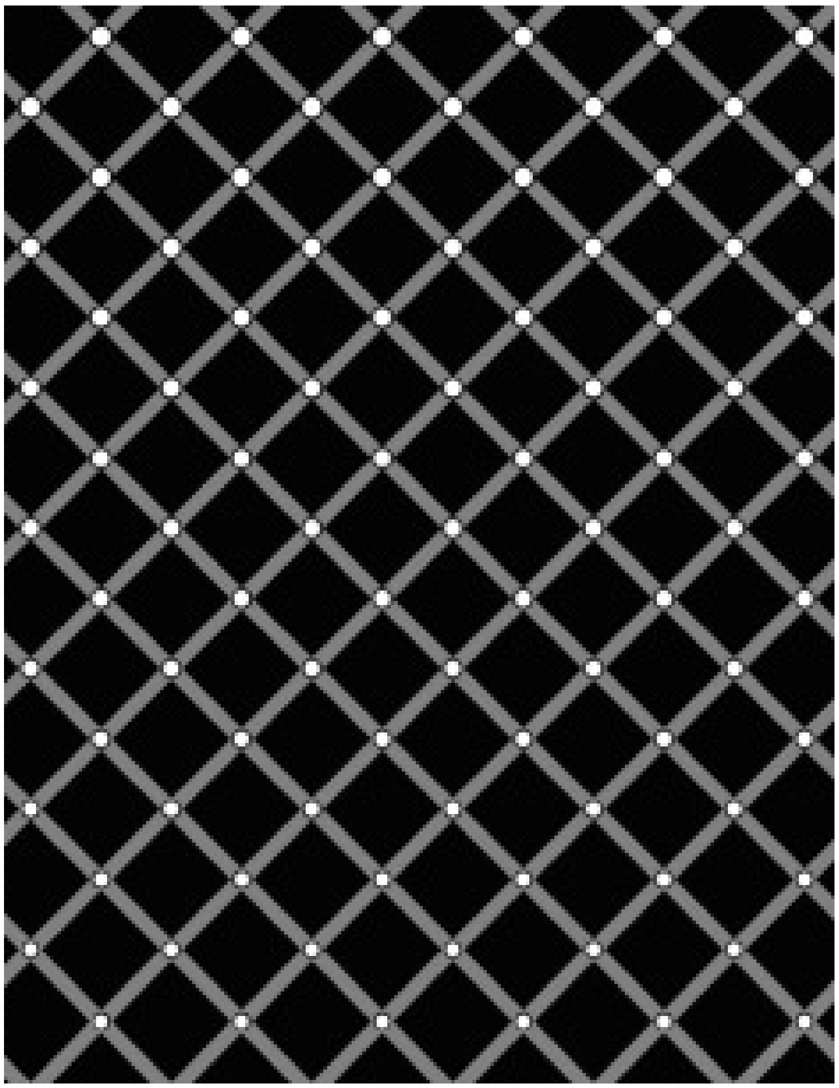


Figure 6: Extinction illusion

Creating and playing sounds

0. Install the *simpleaudio* module if it is not already installed on your computer (check with ipython: `import simpleaudio`), : `pip install simpleaudio`

   Run the quick check with ipython:

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

   Check out simpleaudio tutorials

1. run `sound_synth.py`, look at the code.
2. Take a mono sound and create a stereo sound by progressively dephasing the two channels.
3. Create rhythmic stimuli as described in Povel and Essen (1985) *Perception of Temporal Patterns*

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 the screen and the participant must click on the mouse as quickly as possible. The reaction times must be recorded in a file for further analyses.

   - Here is a solution using pygame. Run it and check `reaction_times.csv`.
   - Here is a solution using expyriment.

   Run the previous script. Check the results file in the folder `data`. Launch ipython in the `data` folder and type:

   ```
   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)
   ```

2. Read `https://docs.expyriment.org/Tutorial.html` to understand the basic pinciples of expyriment. See 'PCBS/expyriment_template.py"

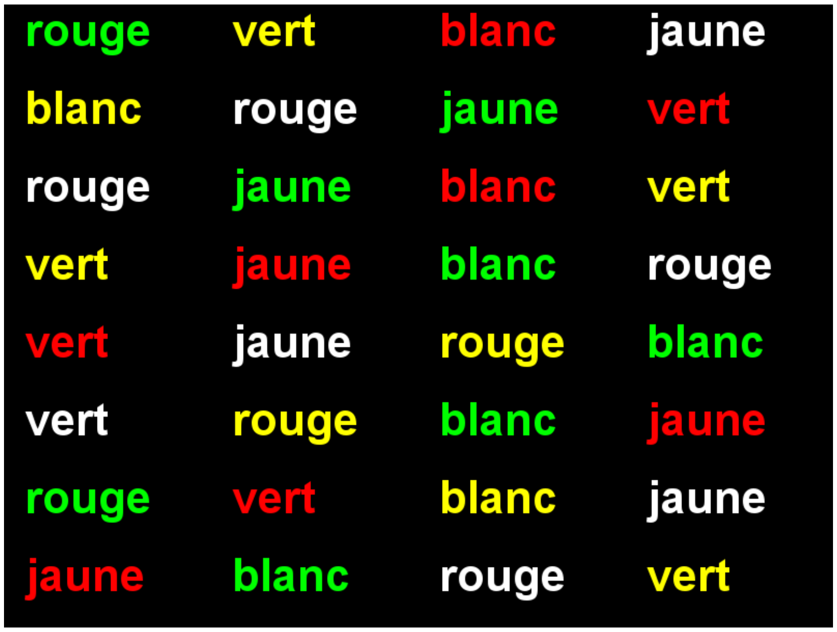
3. Modify `simple-detection-visual-expyriment.py` to play a short sound (`click.wav`) in lieu of displaying a cross. Thus you have a simple detection audio experiment.

4. Modify the script to have 3 blocks of trials: one in which the target is visual, one in which it is audio, and one in which it is randomly visual or auditory. Are we slowed down in the latter condition?

5. Program Posner's attentional cueing task See solution in `Posner-attention/posner_task.py`.

Stroop Effect

The Stroop Effect demonstrates the automaticity of reading. Write a python script to create 4x8 cards for the task, avoiding repetitions of colors.



rouge	vert	blanc	jaune
blanc	rouge	jaune	vert
rouge	jaune	blanc	vert
vert	jaune	blanc	rouge
vert	jaune	rouge	blanc
vert	rouge	blanc	jaune
rouge	vert	blanc	jaune
jaune	blanc	rouge	vert

Figure 7: Stroop card

You can read a tutorial on how to display text with pygame

- After trying to program it yourself, you can compare with my solution

- Run stroop_task.py and check the naming times in `data`. Compute the average reading times as a function of the language (you can use R or Python).

Lexical Decision Task

In a lexical decision experiment, a string of characters is flashed at the center of the screen and the participant has to decide if it is real word or not, indicating his/her decision by pressing a left or right button. Reaction time is measured from the word onset, providing an estimate of the speed of word recognition.

- Using [lexical-decision/select-words-from-lexique.py] as an example, select 20 high frequency nouns, 20 low frequency nouns, 20 high frequency verbs and 20 low frequency verbs, from Lexique382.txt (which can be download from `http://www.lexique.org`). They must all have a length of 5 to 8 characters.
- generate 50 pseudowords using either Lexique tools or Wuggy
- Program a lexical decision using expyriment.
- Run it and compute the average decision times using pandas

More examples of experiments using expyriment.org

- See `http://docs.expyriment.org/old/0.9.0/Examples.html`
- See `https://www.github.com/chrplr/audiovis` : a general audio visual stimulus presentation script using expyriment
- Fork `https://github.com/expyriment/expyriment-stash` and contribute by adding new scripts!

Lexical Frequencies

Zipf law

- The script (word-count.py])[Zipf/word-count.py] computes the distribution of frequencies of occurences in a list of words. Use it to compute the distribution of word frequencies in *Alice in Wonderland*.

  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

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

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*.

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.