Mind and Brain: Introduction to Python
Winter Semester 2019/2020
Final programming assignment

To get the highest level of credit for the course, you can complete a final programming assignment in addition to the exam. This document explains the nature of the final programming assignment and outlines the grading criteria.

Because the course covers a few slightly different applications of Python, you have a choice of five different assignments. Each involves writing a Python module or script, but for different purposes.

These tasks go somewhat beyond what we will learn about in class. The things that we learn will give you the basic structure, but you will need to look up some new techniques and do some independent learning.

I have left some of the minor details unspecified. You should choose what you think works best, and add any additional features that you think are useful.

Format

Create a repository on GitHub and put your program files there. The use of git is not the main focus of the course, so you do not have to develop your program using git, and I will not look at the version history of your program when grading your project. If you prefer you can develop your program completely without the help of git, and then just upload the files when you have finished. You may choose whether your GitHub repository is public or private. If you have kept it private, you should add me (username: luketudge) as a collaborator when you have finished, so that I can see your repository. I of course won't abuse my collaborator status to change anything in your project or to make your work public.

The contents of your repository will vary depending on which project you have chosen, but should include the following:

- A *README* file that explains how I should use your program.
- One or more Python files containing the program itself.

When you are ready, email me a link to your repository. I will clone it and follow the instructions in your *README* file.

I will use Python 3 to test your program. If your program requires any additional Python packages that are not part of the standard library, you can use a *requirements.txt* file to state which third-party packages I need, and I will install them myself.

There are a few extras that you shouldn't bother with unless you really want to learn about them. These will take extra time and are not really necessary for small programs:

- You don't need to make sure your program is fast or memory-efficient. This isn't the focus of the course.
- You don't need to turn your program into a fully installable Python package. Apart from being a bit tedious for you, this will also mean it takes longer for me to test it.

Programming is often a cooperative endeavor, so you may of course get help from online forums and adapt solutions that other people have developed. But if you use somebody else's work, even in modified form, you should acknowledge it. You can include a file listing your sources, or you can note them as comments in your Python files.

Grading criteria

The utterly baffling German grading system includes 10 passing grades, from a bare pass at 4.0 to the top grade at 1.0. To make things as clear and objective as possible, there are 10 points available for the project, and each point you earn will put you one grade further up.

The 10 points are distributed over 5 grading criteria, for each of which you can earn either 0, 1, or 2 points.

Correctness. Does your program work as required?

- 0 points: Only works for some cases, or some required features are missing.
- 1 point: Works for all the standard cases.
- 2 points: Gives informative error messages for incorrect use or other problems. Includes some systematic testing.

Clarity. Is your code clear and readable? Are the important parts easy to find? If another programmer came to your work, would they be able to find out what is going on?

- 0 points: Code is messy and contains redundancy.
- 1 point: Variables have clear names. Little redundancy.
- 2 points: No redundancy. Code is intuitively organized, either across more than one file or in sections within one file. Comments clarify the more complex parts of the code.

Documentation. Have you included documentation for your program? Can people find out easily how to use it? (Note that I will not penalize any purely linguistic errors in your documentation so long as it is comprehensible).

- 0 points: Little or no døcumentation, or it is unclear.
- 1 point: More or less everything important is documented. Docstrings cover input parameters and return values.
- 2 points: Documentation additionally gives examples, covers different possible uses where applicable, and takes a more readable form than just comments and docstrings.

Flexibility. If you or another programmer wanted to change some of the parameters in your program, would they be able to do so easily?

- 0 points: Nothing important can be changed without breaking the program.
- 1 point: Some important parameters can be changed by changing the value of a variable.
- 2 points: Important parameters can be changed easily. Documentation explains how to make changes and/or suggests ways to adapt or improve the program.

Initiative. Have you done some independent learning? Have you used some techniques that go beyond what we covered in class? This is the criterion that I will use to sort out the very best projects that get the top grade.

- 0 points: Only used the techniques from class.
- 1 point: Some small innovations, such as using a new third-party library.
- 2 points: Appropriate use of at least one genuinely new technique that was not covered in class, or only in passing. For example (but not limited to): object-oriented programming, generators, decorators, shell scripts, code coverage, automated style checking, jupyter notebooks, Sphinx.

1) Basic Python: Scrambled text

In 2003, an internet meme circulated in which an unusual piece of text was presented. Each word in the text had been ‘scrambled’ such that every letter except the first and last letters was in a random position. Nonetheless, people found this text surprisingly easy to read. A vague neurosciency explanation was offered, along the lines of the brain ‘filling in’ the middle parts of the words. You can read more about this phenomenon at:

http://www.mrc-cbu.cam.ac.uk/people/matt-davis/cmabridge/

Write a Python module that provides four functions, each of which scrambles a piece of text in a different way:

- Randomly reorder all but the first and last letters of each word (as in the original internet meme).
- Randomly reorder all the letters in each word.
- Randomly reorder all the letters in words that are shorter than a given number of letters (make this given number of letters an extra argument to the function).
- Randomly make some letters lowercase and some uppercase (for example: “fOR eXaMPLe”)

You can add any additional helper functions you think might be useful for psychologists or linguists who want to to generate scrambled texts.

Hints:

- Make sure you handle punctuation correctly. Surrounding punctuation should not be scrambled or counted as the first or last letter of a word.
- Testing will be particularly important for getting this task right. Make sure you test your functions to check they really work as required.

2) Images: Checkerboard mask

In the British television game show *Catchphrase*, contestants have to identify an image. The image is divided into 9 rectangular pieces, and each piece is revealed in turn until one of the contestants is able to identify the image. You can see an example of the game here:

https://www.youtube.com/watch?v=qqb-j1cNPhQ

Write a Python module providing a function that can be used to create partially masked versions of images, like in the show. The function should take an image object as its input and return a partially masked version of that same image, in which some of the 9 rectangular regions of the image have been made black. An integer input argument is used to determine how many of the 9 pieces of the image are blacked out and how many are revealed.

Hints:

- You might find it easier to move some of the image processing steps into additional 'helper functions', so that you can more easily test these steps separately.
- Not every image will have dimensions that are exactly divisible by 3 so as to produce equal-sized sections. You will need to decide on a means of dividing the image up into sections that are as equal in size as possible.

3) Data analysis: Bugs

A somewhat silly psychology experiment conducted in 2013 measured subjects' reactions to pictures of bugs. Independent raters had deemed the bugs to be either disgusting or non-disgusting, and either frightening or non-frightening. The subjects had to rate how much they wanted to kill each bug, on a scale from 1 to 10. You can read more about the experiment here:

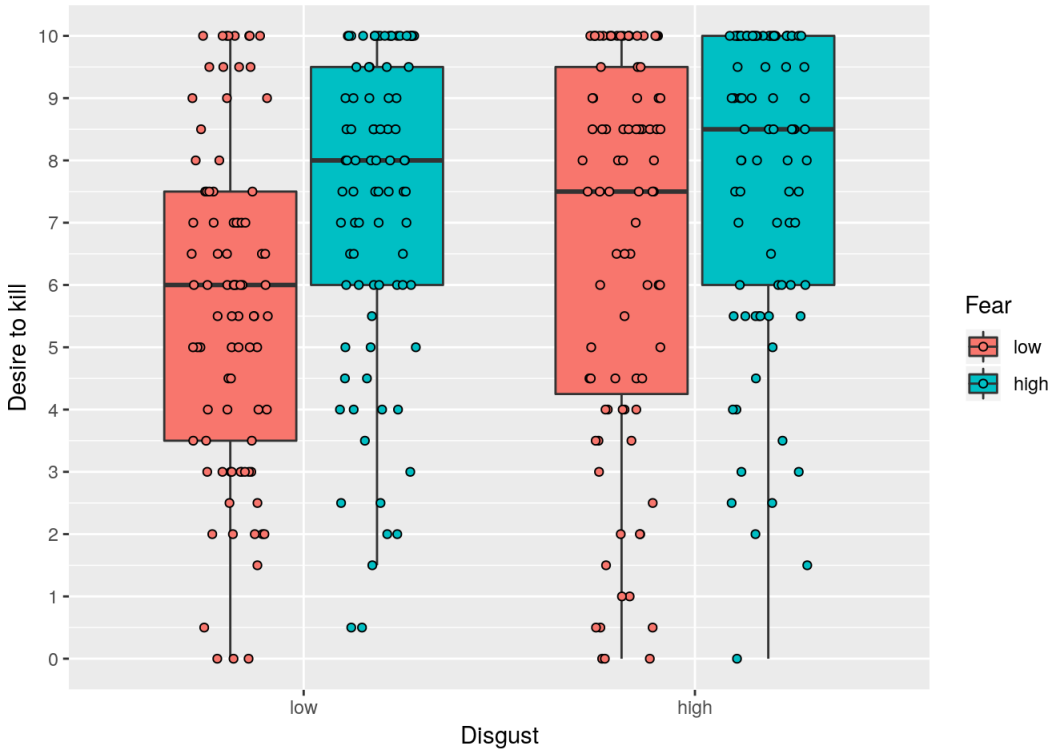
https://doi.org/10.1016/j.chb.2013.01.024

And you can obtain a csv file of the data from the experiment here:

https://raw.githubusercontent.com/luketudge/stats-tutorials/master/tutorials/data/bugs.csv

Write a Python program that produces an analysis of these data. Your program should print some numerical summaries and produce a plot:

- Summary statistics of the kill ratings for each type of bug:
  - minimum
  - maximum
  - median
  - mean
  - Standard Deviation
- The results of a linear model with kill rating as the outcome variable and the categories of bug as the predictor variables.
- Boxplots with overlaid points showing the distribution of kill ratings for each category of bug. Try to recreate the main features of the plot shown below.



4) The internet: mind-and-brain.de

The ‘Faculty’ page at the website for the Berlin School of Mind and Brain lists the school’s current faculty members:

http://www.mind-and-brain.de/people/faculty/

Write a Python program that retrieves the details of faculty members from the current version of the web page and saves their information into a spreadsheet file. For each faculty member, get all of the information that is displayed in their detail view (Function, Current status, Research area, etc.), except for their phone number and email address (although this information is in the public domain, it is a bit rude to retrieve it like this).

It is up to you whether you organize your program as a module with functions for retrieving and saving information, or as a script that carries out all steps at once.

Hints:

- Look at the html source code of the page first, to find out what elements you should search for.
- Watch out for non-ASCII characters and make sure you specify an encoding for saving these to file.

5) Psychopy: Matching pennies

Matching pennies is a very simple game for two players. One player is called ‘even’ and the other player is called ‘odd’. Each player has a penny. On each round of the game, both players present their pennies to each other, turned either heads-up or tails-up. If the two pennies match (i.e. either both heads or both tails), the ‘even’ player wins a point. If the two pennies do not match (i.e. one heads and one tails), the ‘odd’ player wins a point. You can read some more about the game on Wikipedia:

https://en.wikipedia.org/wiki/Matching_pennies

Write a Python script that allows a subject to play a game of matching pennies against the computer. On each round, the subject chooses with a key press whether to present their penny heads-up or tails-up. The computer makes its choice randomly. When the subject has made their choice, both pennies are shown on the screen and the subject is informed of whether they won that round or not, and what the current scores are. Allow the subject to quit at any round by pressing a quit key instead of making a choice.

To make the script a bit more like a genuine experiment, include at least one of the following additional features:

- Allow the experimenter to allocate the computer a different strategy, for example biasing its choice towards heads or tails, or towards switching from its choice on the previous round.
- Produce a printout of the results at the end of the program, showing:
  - How often the subject switched their choice so that it was different from their choice in the previous round.
  - How often the subject switched their choice so that it was different from the *computer’s* choice in the previous round.

Hints:

- In experiments, user-friendliness counts. Make sure your program offers a smooth and intuitive user experience for the subject. Get a friend to test it out.