PHOT 110: Introduction to programming

Final exam questions, retake

Michaël Barbier, Spring semester (2023-2024)

Before you start

The final exam counts for 50% of your total grade of the course. You will get 3 hours to complete your exam. The exam is performed on the computer, and you do not need to provide any answers on paper. There is one question (the third question) which asks to correct a script which contains various errors, other questions require you to write Python scripts.

During our final exam, there will be a very short (about 5 minutes) oral part. This part is only about the projects, and impacts only the grade of your projects. You will be asked a couple of short questions about the project: the code you wrote and/or the accompanying report. If required you can prepare the answer on paper (there will be paper available), you also will have a print-out version of your report. In principle, you do not need to prepare for this oral part, the questions' only purpose is to see whether you understood what you did in the project.

Take the following points into account before you start the exam.

- For those who use their own computer/laptop: you do not need internet, we will ask you to switch of your WiFi before the exam starts. The exam files are distributed via USB drive. Copy all files from the USB drive to your computer before you start:
  - The questions: `phot110_exam_questions_x.pdf`
  - The script with errors for question 3: `script_with_errors.py`
  - Three cheat-sheets: for Python, Numpy, and Matplotlib.
- For those who work on the desktop computers in the lab: all the required files should be found on the Desktop.
- Make sure that the `Numpy` and `Matplotlib` libraries are installed already (we will test this together before the exam starts).
- At the end of the exam we will go around with USB drives to collect the exams: put your solutions (and output files) in a folder which has both your name and your student number in the folder name, for example "Michael_Barbier_30029034".

- Let us know if during the exam there is any issue with the computer, PyCharm, or libraries. We will try to verify this up front, but please inform us in case of any issues.

You will be asked to save plots to your folder, this can be done using the `savefig` method, see the following example (the output of this example is a simple line plot):

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
x = [1, 4, 3, 0]
y = [2, 2, 5, 4]
ax.plot(x, y)
fig.savefig("output_plot_example.png")
```

Questions

Please solve all of the following 7 questions. Each question is worth an equal amount of points (out of 100).

Question 1:

Write a script that prints the characters of a word (string) each on a separate line and adds a growing number of space symbols in front. Use a loop (e.g. `for` or `while`) structure to perform this automatically. For the example string: `Good morning`, a correct script should have output similar to:

```
G
 o
  o
   d

     m
      o
       r
        n
         i
          n
           g
```

Save your solution as a script with file name: `solution_1.py`.

Question 2: Walking distance counter

Make a script that prompts a user repeatedly to give his/her daily walked distance in kilometers (as a positive integer). Stop the program when the user provides "stop" instead of a number. Every time print the total kilometers walked so far. This is example output of a correct script:

```
Walking tool: Provide positive integers to add a walking distance, or stop to exit.

Please provide the distance you walked today: 4
Total walking distance so far: 4 km

Please provide the distance you walked today: five km
This is not a valid amount please try again!

Please provide the distance you walked today: 6
Total walking distance so far: 10 km

Please provide the distance you walked today: stop
Total walking distance so far: 10 km
```

Save your solution as a script with file name: `solution_2.py`.

Question 3: Correct a Python script

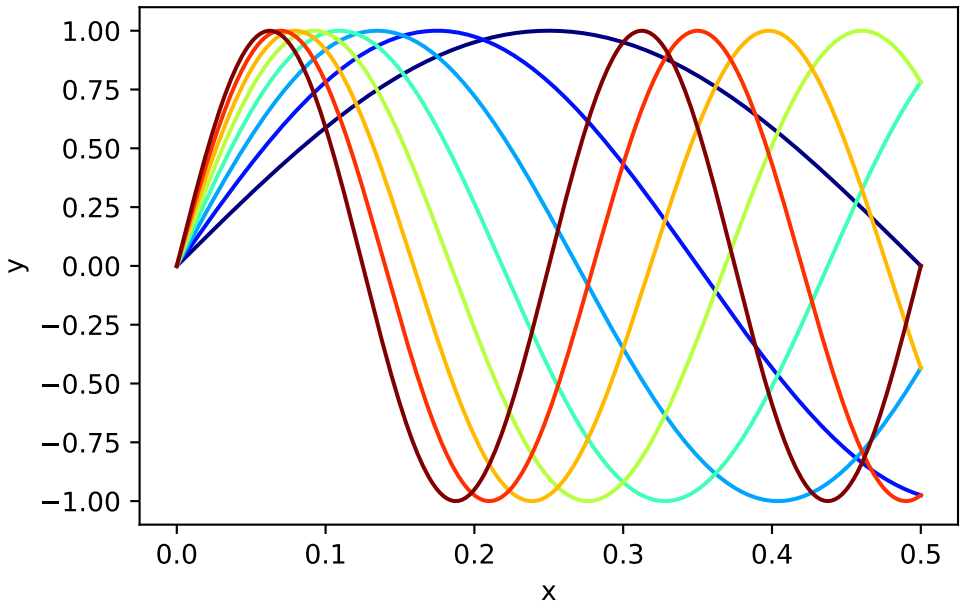
Open the script with name: `script_with_errors.py` and correct the errors. Save the corrected script with file name `solution_3.py`.

The corrected script should plot sinusoidal curves with different frequencies $f_i = 1, 1.5, 2, 2.5, \ldots, 4$ Hz. The y-coordinates of the curves are given by:

$$y = \sin(2\pi\, f_i\, x)$$

The different curves are colored according to their frequency (using the "jet" colormap). The graph is then saved as a png-file with file name `output_script_with_errors.png` to the hard disk.

The output graph should look as the plot below:



Question 4: 2D density plot

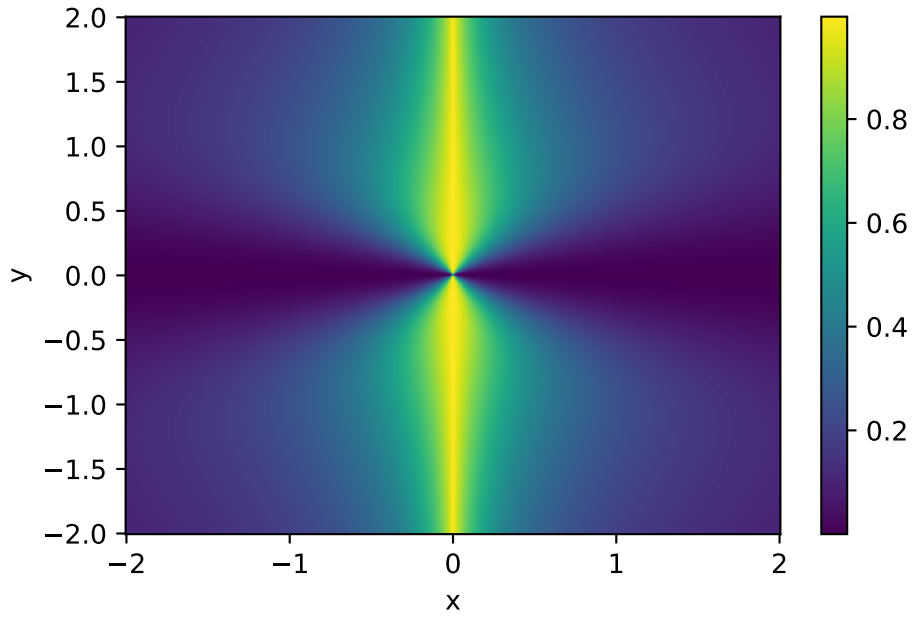
Plot $z(x, y)$ given by the following formula:

$$z(x, y) = \frac{1}{1 + 2|x||y| + \frac{x^2}{y^2}}$$

whereby you can use the function `abs()` of Numpy. Plot the resulting function $z(x, y)$ as a density plot. Remember that you can make a density plot (with a colorbar) using the commands:

```
fig, ax = plt.subplots()
pc = ax.pcolormesh(xx, yy, zz, rasterized=True)
fig.colorbar(pc, ax=ax)
```

Use (x, y)-values within an 2D interval/domain with $x \in [-2, 2]$ and $y \in [-2, 2]$ (take a sufficiently fine grid of (x, y) coordinates to obtain a smooth density plot). Avoid choosing coordinates with $y = 0$, to prevent divisions by zero. **Save the plot** under the file name: `output_plot_cross.png`. Save your solution as a script with file name: `solution_4.py`. The output of the script should look as in the plot below:



Question 5: Creating and using modules

Create a **module** (a separate script file) with file name `module_overlap.py` which helps to verify whether there is any overlap between two circles with radii R_1 and R_2 that are separated by distance D (between their center coordinates). The module should contain two functions:

- `circle_overlap(R1, R2, D)`: verifies whether there is overlap between two circles with radii R_1 and R_2 which centers are distance D apart:

$$D \leq R_1 + R_2$$

- `distance(x1, y1, x2, y2)`: calculates the distance D between two points with coordinates (x_1, y_1) and (x_2, y_2):

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Afterwards, import and use this module in a script (with file name: `solution_5.py`) that defines three circles by following lists of x coordinates, y coordinates, and radii:

```
xs = [1, 3, 2]
ys = [2, 5, -2]
Rs =  [1, 3, 1]
```

and verifies whether any circles are overlapping (pairwise) using the `circle_overlap(R1, R2, D)` function together with the `distance(x1, y1, x2, y2)` function from the module. Afterwards, print out the results: the output should look as follows:

```
Circles:
 - circle 1: x = 1, y = 2, radius = 1
 - circle 2: x = 3, y = 5, radius = 3
 - circle 3: x = 2, y = -2, radius = 1
----------------------------------------
circles 1 and 2 do overlap!
circles 1 and 3 do not overlap.
circles 2 and 3 do not overlap.
----------------------------------------
```

Question 6: Dictionaries

Define the following dictionary with timings (in seconds) of a 100 meter sprint contest with different categories according to age in your script:

```
sprint_times = {
  "seniors": [10.6, 11.3, 12.0, 10.4, 11.8, 10.9],
  "juniors": [12.0, 12.6, 13.4, 12.7],
  "kids": [15.2, 16.1, 13.8, 14.9, 15.4],
}
```

Then let the script automatically print out the average time per age category (loop over the categories). You can use the `mean()` function of Numpy to find the average (after casting the list to a Numpy array). The output of a correct script should be the following:

```
The average times per age category are:
seniors: 11.17 seconds.
juniors: 12.68 seconds.
kids: 15.08 seconds.
```

Save your script under the name `solution_6.py`.

Question 7: Classes

Create a class named `ChessPlayer` which has three attributes

- `name`: the name of the player,
- `rating`: the current rating of the player

The ChessPlayer object represents a chess player in a tournament that plays matches against other chess players. Give the class a constructor that accepts two arguments (in addition to the mandatory `self` argument), the player's name and his/her rating: `__init__(self, name, rating)`.

Then add the following method:

- `update_rating(self, score, rating_opponent)`: which calculates the player's new rating R_{new} after a match against an opponent. The method updates the `rating` attribute of the player. The new rating of a player R_{new} after a match depends on the outcome score S of the match (won: $S = 1$, draw: $S = 1/2$, lost: $S = 0$), and the rating R of the player and the rating R_{opp} of its opponent according to the following formula:

$$R_{new} = R + 40 \times (S - E), \qquad \text{with} \quad E = \frac{10^{R/400}}{10^{R/400} + 10^{R_{opp}/400}}$$

Then use the class in your script to make two ChessPlayer objects for chess players "Ivan" and "Mehmet". Give Ivan a rating of 1184 and Mehmet a rating of 1310. Then update their rating after they play a game which is won by Mehmet. Use the following code to update both their ratings:

```
player_mehmet.update_rating(1, player_ivan.rating)
player_ivan.update_rating(0, player_mehmet.rating)
```

Print their ratings before and after they played the game. This is example output of a correctly working script:

```
Before the game:
 The rating of Mehmet = 1310
 The rating of Ivan = 1184
After the game:
 The rating of Mehmet = 1323
 The rating of Ivan = 1171
```

Save your script under the name `solution_7.py`.