PHOT 110: Introduction to programming

Final exam questions and solutions, retake

Michaël Barbier, Spring semester (2023-2024)

Questions/problems and solutions

We first show the question, then the solution and then how the points are counted (raw score). The points of each question are normalized to 100/7. We then sum these for the 7 questions to get a total score on 100. If question i has M_i points and one obtained m_i/M_i points, then the total score S is:

$$S = \sum_{i=1}^{7} [\frac{100}{7} \times m_i/M_i]$$

In the score calculation of the solutions to the questions, we appoint different amount of points. However, every question has the same weight (100/7), due to the above mentioned normalization of the points.

Remark that there is a minimal amount of comments used in the problem solutions. This is to keep the code listings within this document more compact. In real scripts I would advice to put more comments to document your code.

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`.

Solution 1

```python
word = "Good morning"
for i, c in enumerate(word):
    print(" " * i + c)
```

Score calculation

6 points to be obtained:

1. Being able to add spaces at the front of a string (1 point)
2. Having multiple lines (1 point)
3. Having the number of spaces growing with the number of lines (1 point)
4. automatically grow the number of spaces (1 point)
5. Using a loop structure to print one string per line (1 point)
6. Script runs without or with only trivial errors (1 point)

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`.

Solution 2

```python
print("Walking tool: Provide positive integers to add a walking distance, or stop to exit.")

total_distance = 0

while True:
    user_input = input("Please provide the distance you walked today: ")
    try:
        if user_input == "stop":
            break
        else:
            walked_km = int(user_input)
            if walked_km > 0:
                total_distance += walked_km
            else:
                print("This is not a valid amount please try again!")
    except ValueError:
        print("This is not a valid amount please try again!")
    finally:
        print(f"Total walking distance so far: {total_distance} km\n")
```

Score calculation

6 points to be obtained:

1. Prompt the user for input (1 point)
2. Convert the input to an integer (1 point)

3. Verify whether the input equals “stop” to stop the program (1 point)
4. Print the total distance walked (1 point)
5. Prompt the user again if input is not appropriate (1 point)
6. Script runs without or with only trivial errors (1 point)

Question 3: Correct a Python script

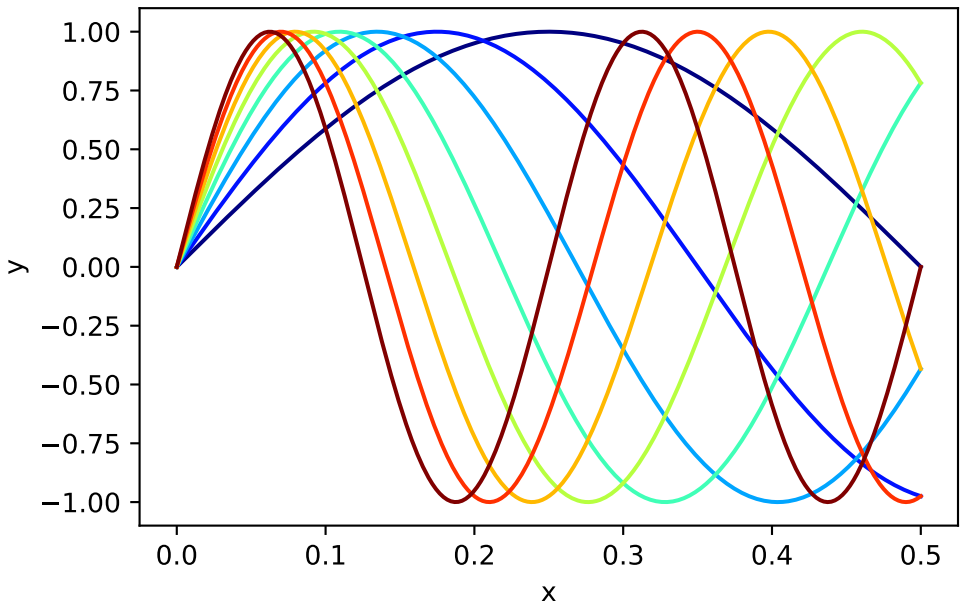
Open the script with name: `script_with_errors.py` and correct the errors. Save the corrected script with file name `solution_3.py`.

The corrected script should plot sinusoidal curves with different frequencies f_i = 1, 1.5, 2, 2.5, ... , 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:



Solution 3

Procedure to reach to the solution:

- Lines 15-18: Indentation should be removed.
- On line 21: Comment `Colors of the lines` needs to start with a `#` symbol.
- On line 24: For loop needs a colon (`:`).
- On line 26: `sin(...)` is a Numpy function: replace by `np.sin(...)`
- On line 27: Missing closing parenthesis `)` at the end of the line
- On line 30: `ax.set_label("x")` should be replaced by `ax.set_xlabel("x")`

```
1  # This is the corrected script.
2
3  # Load the necessary libraries
4  import numpy as np
5  import matplotlib as mpl
6  import matplotlib.pyplot as plt
7
8  # Initialize the figure
9  fig, ax = plt.subplots()
10
11 # Define the interval of x-coordinates to be used
12 xs = np.linspace(0, 1/2, 400)
13 # Define the frequencies to plot
14 n = 8
15 fs = np.linspace(1, 4, n)
16 # Colors of the lines
17 colors = mpl.cm.jet(np.linspace(0, 1, n))
18
19 for i, f in enumerate(fs):
20     # plot the sine curve and indicate its maximum
21     ys = np.sin(2 * np.pi * f * xs)
22     ax.plot(xs, ys, color=colors[i])
23
24 # Set the axes labels and the limits
25 ax.set_xlabel("x")
26 ax.set_ylabel("y")
27
28 # Save the resulting plot
29 fig.savefig("output_script_with_errors.png")
```

Score calculation

8 points to be obtained:

1. Define a figure/axis to plot in (1 point)

2. Having the frequencies (1 point)
3. Having the x coordinates (1 point)
4. Having the colors (1 point)
5. Plotting of all sine curves (1 point)
6. Having the same plot style (1 point)
7. Saving the output to a file (1 point)
8. Script runs without or with only trivial errors (1 point)

Question 4: 2D density plot

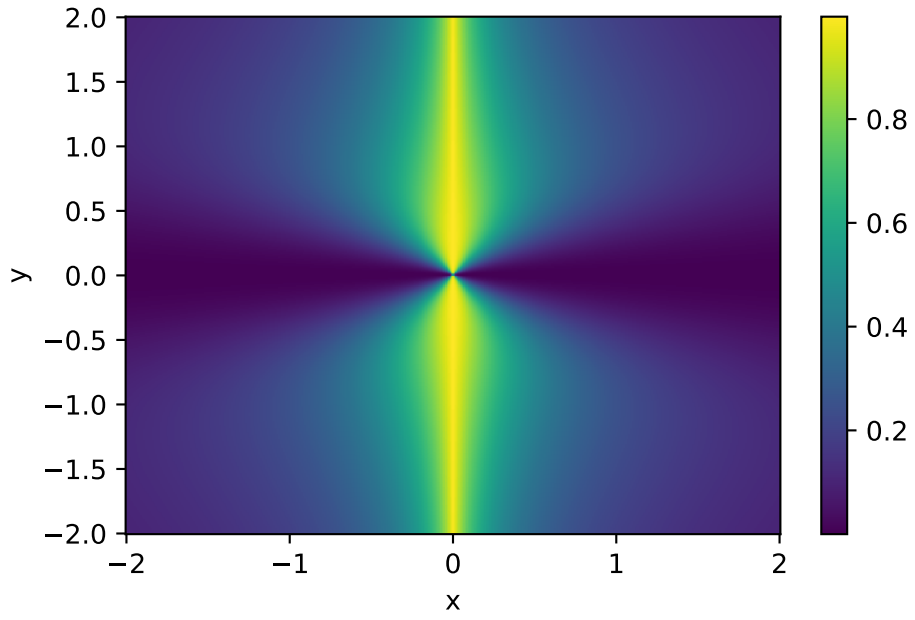
Plot $z(x, y)$ given by the following formula:

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

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:



Solution

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, 400)
y = x
xx, yy = np.meshgrid(x, y)
zz = 1 / (1 + 2*np.abs(xx*yy) + xx**2 / yy**2)

fig, ax = plt.subplots()
pc = ax.pcolormesh(xx, yy, zz, rasterized=True)
ax.set_xlabel("x")
ax.set_ylabel("y")
fig.colorbar(pc, ax=ax)

fig.savefig("output_plot_cross.png")
```

Score calculation

7 points to be obtained:

1. Creating multiple x and y values within the intervals forming the domain (1 point)
2. Creating 2D arrays of coordinates within the domain
3. Obtaining correct z-values from the (x, y)-coordinates (1 point)
4. Having a smooth plot (1 point)
5. Style of the plot looks like the example (1 point)
6. Enabling saving the plot (1 point)
7. Script runs without or with only trivial errors (1 point)

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.
----------------------------------------
```

Solution

The solution exists out of two files:

- `module_overlap.py`: the module providing functionality to calculate the overlap of two circles/disks.
- `solution_5.py`: the main script which plots the current I (in Ampere) as function of the resistance R.

Remark: In the code listings I removed the underscores of the file names as I had a problem with rendering them in this pdf-file.

Listing 1 `moduleoverlap.py`

```
def distance(x1, y1, x2, y2):
  return np.sqrt((x2-x1)**2 + (y2-y1)**2)

def overlap(R1, R2, D):
  return R1 + R2 >= D
```

Score calculation

10 points to be obtained:

1. Creation of a separate module file (1 point)
2. Implementing the correct expressions for the distance and overlap (1 point)
3. Function definitions implementing both functions (1 point)
4. Importing the module (1 point)
5. Adding the circle information (1 point)
6. Calling the `distance()` function to calculate the inter-circle distances (1 point)
7. Calling the `overlap()` function to calculate the inter-circle area overlap (1 point)
8. Verifying the overlap for the correct combinations of circles (1 point)
9. Enabling a similar output as in the question (1 point)
10. Script runs without or with only trivial errors (1 point)

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:

Listing 2 `solution5.py`

```
# Import the module we created
import moduleoverlap as mo

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

print("Circles:")
for i in range(len(xs)):
  print(f" - circle {i+1}: x = {xs[i]}, y = {ys[i]}, radius = {Rs[i]}")

print("-" * 40)
# For this exam it was sufficient to manually select combinations
for i in range(len(xs)):
  for j in range(i+1, len(xs)):
    D = mo.distance(xs[i], ys[i], xs[j], ys[j])
    if mo.overlap(Rs[i], Rs[j], D):
      print(f"circles {i+1} and {j+1} do overlap!")
    else:
      print(f"circles {i+1} and {j+1} do not overlap.")
print("-" * 40)
```

```
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`.

Solution

```
import numpy as np

# Dictionary with sprint times copied from question description
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],
}

# List the average sprint times per age category
print(f'The average times per age category are:')
for k, v in sprint_times.items():
  avg_v = np.round(np.mean(np.array(v)), 2)
  print(f'{k}: {avg_v} seconds.')
```

Score calculation

8 points to be obtained:

1. Understanding how a dictionary is written (1 point)
2. Having a dictionary similar to the one of the task description (1 point)
3. Using a loop to iterate over all the categories (1 point)
4. Obtaining the average of every one of the lists of a category (1 point)
5. Printing the category names on separate lines (1 point)
6. Printing each category name together with its average (1 point)
7. Enabling similar output to the example (1 point)
8. Script runs without or with only trivial errors (1 point)

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`.

Solution

```
class ChessPlayer():
  def __init__(self, name, rating):
    self.name = name
    self.rating = rating

  def update_rating(self, score, R_opp):
    R = self.rating
```

```
    E = 10 ** (R / 400)  /  (10 ** (R / 400) + 10 ** (R_opp / 400))
    R_new = int(R + 40 * (score - E))
    self.rating = R_new
    return R_new

P1 = ChessPlayer("Mehmet", 1310)
P2 = ChessPlayer("Ivan", 1184)
print("Before the game:")
print(f" The rating of {P1.name} = {P1.rating}")
print(f" The rating of {P2.name} = {P2.rating}")
P1.update_rating(1, P2.rating)
P2.update_rating(0, P1.rating)
print("After the game:")
print(f" The rating of {P1.name} = {P1.rating}")
print(f" The rating of {P2.name} = {P2.rating}")
```

Score calculation

12 points to be obtained:

1. Creating a class (1 point)
2. Having all required class attributes (1 point)
3. Having a constructor which allows giving the chess player a name (1 point)
4. Having the method `update_rating(self, score, rating_opponent)` which calculates the rating update (1 point)
5. The `update_rating(self, score, rating_opponent)` method updates the `rating` attribute (1 point)
6. The `update_rating(self, score, rating_opponent)` method is correctly implemented (1 point)
7. Knowing how to create a ChessPlayer object with a name and rating (1 point)
8. Printing the rating before and after the game (1 point)
9. Applying the `update_rating` method to both players (1 point)
10. Printing the ratings by accessing the objects rating attribute (1 point)
11. Enabling similar output to the example (1 point)
12. Script runs without or with only trivial errors (1 point)