PHOT 110: Introduction to programming

Midterm exam (2nd retake) questions and solutions

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. The points of each question are first normalized to $20 = 100/5$ (where we round up to the next integer). We then add these for the 5 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:

$$S = \sum_{i=1}^{5} [20 \times m_i/M_i]$$

In the score calculation of the solutions to the questions, we appoint different amount of points. However, every question has the same weight (20/100), 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: Multiplication table

Write a script that prints the multiplication table of N. Use a loop (e.g. `for` or `while`) structure to automate the process. Your solution script should print each product on a separate line. If you choose `N = 2` the output should look similar to:

```
0 x 2 = 0
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
```

```
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
10 x 2 = 20
```

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

Solution 1

```python
N = 2
for i in range(11):
  print(f"{i} x {N} = {N*i}")
```

Score calculation

6 points to be obtained:

1. Being able to print the outcome by using a product (1 point)
2. Having multiple equations/multiplications (1 point)
3. Having the correct range: 0, 1, .., 10 (1 point)
4. Printing the resulting equations in correct format (1 point)
5. Using a loop structure to print one multiplication per line (1 point)
6. Script runs without or with only trivial errors (1 point)

Question 2: Ask for a multiple of 3

Make a script that prompts a user repeatedly for input of a number until the number is a multiple of 3. To test whether a number is a multiple of 3 you can use the fact that the rest after integer division by 3 should be 0:

```python
# Example: the modulo operator % gives the rest after division
rest = 5 % 3
print("The rest after integer division of 5 by 3 = " + str(rest))
```

```
The rest after integer division of 5 by 3 = 2
```

If the number provided by the user is not a multiple of 3, prompt the user again. You can assume that the user provides a valid number as input (and not some random word/characters). This is example output of a correct script:

```
Please enter a multiple of 3: 5
The number that you provided is not a multiple of 3, please try again.

Please enter a multiple of 3: 22
The number that you provided is not a multiple of 3, please try again.

Please enter a multiple of 3: 12
Thank you, the number you provided is a multiple of 3.
```

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

Solution 2

```
n = 3
while True:
    number = input("Please enter a multiple of 3: ")
    if number % n == 0:
        print(f"Thank you, the number you provided is a multiple of {n}.")
        break
    else:
        print("The number that you provided is not a multiple of 3, please try again.")
```

Score calculation

6 points to be obtained:

1. Prompt the user for input (1 point)
2. Understanding the usage of the `%` modulo operator (1 point)
3. Check whether the input number is a multiple of 3 (1 point)
4. Print a message when the number is indeed a multiple of 3 (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

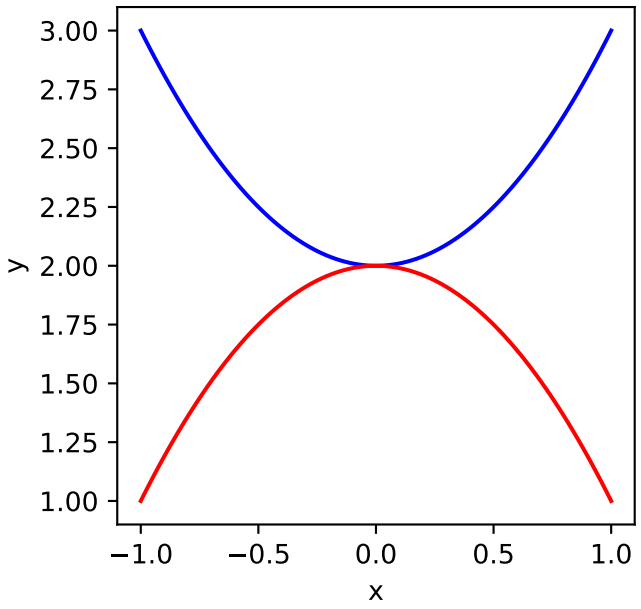
Open the script with name: `script_with_errors.py` and correct the errors. Save the corrected script with file name `solution_3.py`.

The corrected script should plot two parabolic curves with equations:

$$\begin{cases} y_1(x) = x^2 + 2 \\ y_2(x) = -x^2 + 2 \end{cases}$$

This 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:



File of question 3

```
# This script contains errors and doesn't run.
#
# The correct script plots two parabola's. Afterwards, it saves that
# figure to the current folder as a png-file.
#
# Correct all the errors so that it gives the intended output.
```

```
7
8  # Load the necessary libraries
9  import numpy as np
10 import matplotlib.pyplot as plt
11
12 # Initialize the figure
13 fig, = plt.subplots()
14
15 # Define the parabola's
16 x = np.linspace(-1, 1, 100)
17  y1 = x ** 2 + 2
18 y2 = - x ** 2 + 2
19
20 # plot the curves
21 ax.plot(x, y1, color="blue")
22 ax.plot(x, y2, "red")
23
24  # Set the aspect ratio of the axes to equal
25  ax.set_aspect("equal")
26  ax.set_xlabel(x)
27  ax.set_ylabel("y")
28
29 # Save the resulting plot
30 fig.savefig("output_script_with_errors.png")
```

Solution 3

Procedure to reach to the solution:

- On line 13: Should define also the axis `ax`: replacing the line by `fig, ax = subplots()` will fix the issue.
- On line 17: Indentation should be removed.
- On line 22: replace the argument `"red"` by `color="red"`.
- On line 24-27: Indentation should be removed.
- On line 26: The label parameter should be a string: `ax.set_xlabel(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.pyplot as plt
```

```python

# Initialize the figure
fig, ax = plt.subplots()

# Define the parabola's
x = np.linspace(-1, 1, 100)
y1 = x ** 2 + 2
y2 = - x ** 2 + 2

# plot the curves
ax.plot(x, y1, color="blue")
ax.plot(x, y2, color="red")

# Set the aspect ratio of the axes to equal
ax.set_aspect("equal")
ax.set_xlabel("x")
ax.set_ylabel("y")
# Save the resulting plot
fig.savefig("output_script_with_errors.png")
```

Score calculation

7 points to be obtained:

1. Define a figure/axis to plot in (1 point)
2. Having the y coordinates of both parabolas (1 point)
3. Plotting both of the curves in correct range (1 point)
4. Setting the aspect ratio of the axis to equal (1 point)
5. Enabling the plotting of the two lines in correct colors (1 point)
6. Saving the output to a file (1 point)
7. Script runs without or with only trivial errors (1 point)

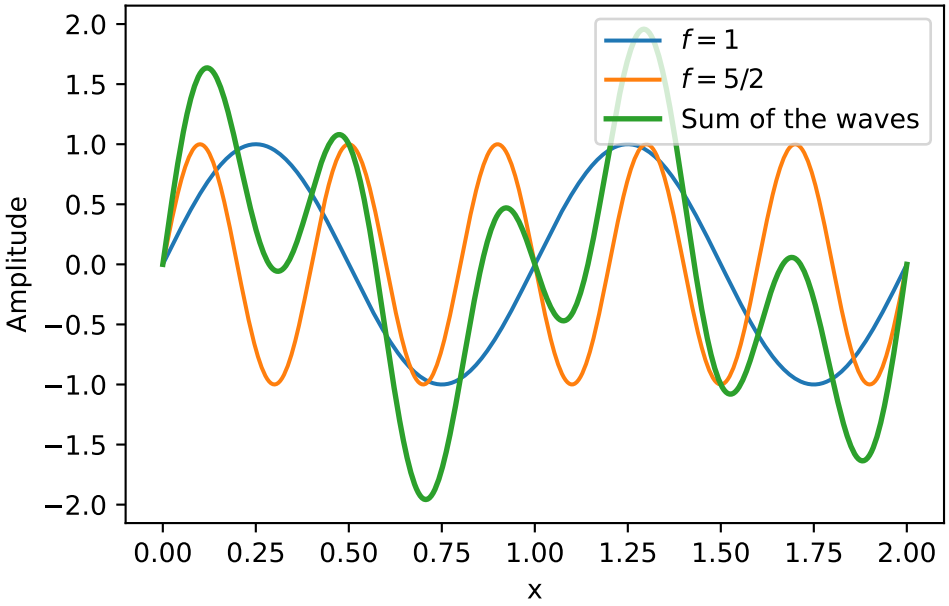
Question 4: Plot superposition of waves

Plot two sine waves with frequencies $f_1 = 1$ and $f_2 = 5/2$:

$$\begin{cases} y_1(x) = \sin(2\pi f_1 x) \\ y_2(x) = \sin(2\pi f_2 x) \end{cases}$$

Plot both these sinusoidal functions as smooth curves. Then add a curve representing the sum of the two function $y_3(x) = y_1(x) + y_2(x)$. Plot both the separate functions and the summed function using a line plot.

For each of the curves you can use the same x-values within an interval $[0, 2]$ (take a sufficiently high number of x values to obtain smooth curves). **Save the plot** under the file name: `output_plot_wave.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 4

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

f_1 = 1
f_2 = 5/2

x = np.linspace(0, 2, 400)
y_1 = np.sin(2 * np.pi * f_1 * x)
y_2 = np.sin(2 * np.pi * f_2 * x)
y_sum = y_1 + y_2
```

```
fig, ax = plt.subplots()
ax.plot(x, y_1)
ax.plot(x, y_2)
ax.plot(x, y_sum, linewidth=2)
ax.set_xlabel("x")
ax.set_ylabel("Amplitude")
ax.legend([r"$f = 1$",
           r"$f = 5/2$",
           r"Sum of the waves"])

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

Score calculation

9 points to be obtained:

1. Knowing how to apply the correct sinusoidal function with a specific frequency (1 point)
2. Creating multiple x-values within the interval (1 point)
3. Having the correct interval (1 point)
4. Obtaining the correct y-values from chosen x-values for a single sinusoidal function (1 point)
5. Obtaining the sum from the separate wave functions (1 point)
6. Having a smooth plot (1 point)
7. Style of the plot looks like the example (1 point)
8. Enabling saving the plot (1 point)
9. 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_validate.py` which helps converting "invalid" text or characters into valid ones. Here we assume that text is invalid if it contains spaces or commas. These symbols will be replaced by an underscore character "_". The module should contain two functions:

- `convert_text(input_text)` which accepts a string `input_text` as argument and returns the same text but with spaces and commas replaced by underscores (`_` symbols)
- `check_validity(input_text)` which accepts a string `input_text` as argument and returns `True` if the text does not contain any spaces or commas, otherwise returns `False`.

Afterwards, import and use this module in a script (with file name: `solution_5.py`) that prompts a user to enter a sentence and converts it to a "valid" sentence using the `convert_text(input_text)` of above. See the following example output of a correct script:

```
Please enter a sentence: Bread, butter, and eggs
The validated text is: Bread__butter__and_eggs
```

See the hints below:

- See the following example to understand how to replace characters in a string:

```
# In this example we replace all "e" characters by "Q"
my_text = "A green house"
adapted_text = my_text.replace("e", "Q")
print(adapted_text)
```

```
A grQQn housQ
```

- To verify whether a string contains a certain character look at following example:

```
# In this example we verify whether the text contains character "s"
my_text = "A green house"
contains_s = "s" in my_text
print(contains_s)
```

```
True
```

Solution 5

The solution exists out of two files:

- `module_validate.py`: the module implementing the text validation functions.
- `solution_5.py`: the main script in which a user is prompted to write a sentence, and then the module function `convert_text(input_text)` is called with that "sentence" as `input_text` argument. Afterwards, the sentence with spaces and commas replaced by underscores is printed.

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 `modulevalidation.py`

```
def convert_text(input_text):
  return input_text.replace(",", "_").replace(" ", "_")

def check_validity(input_text):
  if "," in input_text or " " in input_text:
    return False
  else:
    return True
```

Listing 2 `solution5.py`

```
import modulevalidation as mv

input_text = input("Please enter a sentence: ")
validated_text = mv.convert_text(input_text)
print(f"The validated text is: {validated_text}")
```

Score calculation

10 points to be obtained:

1. Creation of a separate module file (1 point)
2. Knowing how to replace characters in a string (1 point)
3. Using the correct statement to verify the "validity" (1 points)
4. Function definitions implementing both validity check and character replacement (1 points)
5. Importing the module (1 point)
6. Prompting the user for input (1 point)
7. Converting the input text to "valid" text (1 point)
8. Calling the `convert_text()` function to convert to "valid" text (1 point)
9. Enabling similar output to the example (1 point)
10. Script runs without or with only trivial errors (1 point)