PHOT 110: Introduction to programming

Midterm exam (retake) questions and solutions, version A

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: Print out parts of a word

Write a script that prints parts of a word by growing it, starting from the first letter and growing a letter each time. You can make use of the slicing operator, for example, to get the first two characters of a string you could use:

```
# Example of how to use the slicing operator
word = "Apple"
print(word[0:2])
```

```
Ap
```

Your solution script should print each sub-string on a separate line. For example: if you use `word = "Hello"` as word, then the output should look similar to:

```
H
He
Hel
Hell
Hello
```

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

Solution 1

```
word = "Hello"
for i in range(1, len(word)+1):
  print(word[:i])
```

Score calculation

6 points to be obtained:

1. Being able to obtain a part of a word (1 point)
2. Apply the slicing multiple times (1 point)
3. Having the correct parts and range (1 point)
4. Printing the resulting word parts to the console (1 point)
5. Using a loop structure to print one word part per line (1 point)
6. Script runs without or with only trivial errors (1 point)

Question 2: Verify email address

Make the script to verify an email address. Prompt the user to input an email address. Automatically verify whether the format of the student email address ends with `@std.iyte.edu.tr`. Allow the user to try again if his/her input was invalid until the user fills in a valid email address. You can use the `endswith()` method to verify whether the email address is valid:

```
# Example showing how to use the "endswith()" method
my_string = "hello.txt"

# The endswith() method returns True if the string ends in a
# specific substring (here "txt"), otherwise it returns False
print(my_string.endswith("txt"))
print(my_string.endswith("png"))
```

```
True
False
```

This is example output of a correct script:

```
Please enter a valid IYTE student email address: harry@hotmail.com
The provided email address is invalid, please try again.

Please enter a valid IYTE student email address: harry at std.iyte.edu.tr
The provided email address is invalid, please try again.

Please enter a valid IYTE student email address: harry@std.iyte.edu.tr
Thank you, your contact information will be updated.
```

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

Solution 2

```
while True:
    email = input("Please enter a valid IYTE student email address: ")
    if email.endswith("@std.iyte.edu.tr"):
        print("Thank you, your contact information will be updated.")
        break
    else:
        print("The provided email address is invalid, please try again.")
```

Score calculation

6 points to be obtained:

1. Prompt the user for input (1 point)

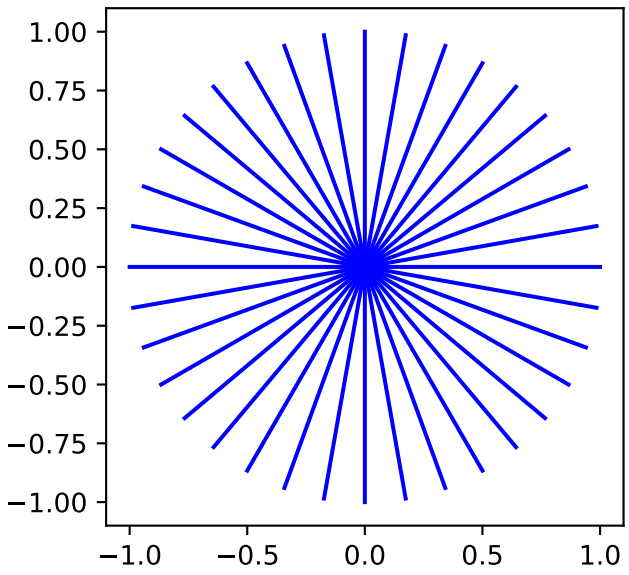
2. Understanding the usage of the `endswidth()` method (1 point)
3. Check the validity of the email address (1 point)
4. Print a message when the email is correct (1 point)
5. Prompt the user again if input is not appropriate (1 point)
6. Script runs without or with only trivial errors (1 point)

Question 3: Correct a Python script

Open the script with name: `script_with_errors.py` and correct the errors.

The corrected script plots a graph where lines are plotted outwards from the origin. There is a line every 10 degrees. 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 lines of length one starting
# from the origin outwards. The lines are plotted every 10
# degrees and are plotted in blue. Afterwards it saves that
# figure to the current folder as a png-file.
```

```
#
# Correct all the errors so that it gives the
# intended output.

# Load the necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Initialize the figure
fig, ax = subplots()

# Define the parameter r of a line and the angles
r = numpy.linspace(0, 1, 100)
angles = np.linspace(0, 2*np.pi, 36, endpoint=False)
# Loop over all angles and plot a line
for angle in angles
    x = r * np.cos(angle)
   y = r * np.sin(angle)
    ax.plot(x, y, color=blue)

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

Solution 3

Procedure to reach to the solution:

- On line 15: function `subplots()` should be called from `Matplotlib.pyplot`. Replacing it by `plt.subplots()` will fix the issue.
- On line 18: `numpy` is imported as `np`, therefore `np.linspace(0, 1, 100)` should be used.
- On line 21: A colon symbol ":" should be added at the end.
- On line 23: The indentation/space before `y = r * np.sin(angle)` should be made equal to the one of line 22 and 24.
- On line 24: The color parameter should be a string: `ax.plot(x, y, color=blue)` should be replaced by ax.plot(x, y, color="blue").

```
# This is the corrected script.

```

```python
# Load the necessary libraries
import numpy as np
import matplotlib.pyplot as plt

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

# Define the parameter r of a line and the angles
r = np.linspace(0, 1, 100)
angles = np.linspace(0, 2*np.pi, 36, endpoint=False)
# Loop over all angles and plot a line
for angle in angles:
    x = r * np.cos(angle)
    y = r * np.sin(angle)
    ax.plot(x, y, color="blue")

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

Score calculation

7 points to be obtained:

1. Fixing the Matplotlib error: having a figure/axis to plot in (1 point)
2. Having radial points in an interval to plot the lines (1 point)
3. Enabling the plotting of multiple lines (1 point)
4. Plotting the lines of correct length (1 point)
5. Plotting the lines with correct angle differences/interval (1 point)
6. Saving the output to a file (1 point)
7. Script runs without or with only trivial errors (1 point)

Question 4: Creating and using modules

Create a **module** (a separate script file) with file name `module_conversion.py` which provides the conversion between units of Watt (W) and Horsepower (hp) contains two functions:

- `value_hp = watt_to_horsepower(value_watt)`: which converts values in kiloWatt to horsepower.

- `value_watt = horsepower_to_watt(value_hp)`: which converts values in Horsepower to values in unit of kiloWatt.

For the definition of the conversion assume that kiloWatt (kW) and Horsepower (hp) are related as follows:

$$1 \text{ kiloWatt} = 1.34 \text{ hp}$$

Afterwards, import and use this module in a script (with file name: `solution_4.py`) that prompts a user to enter his/her car's power in horsepower and then converts it to kiloWatt, and prints the result. See the following example output of a correct script:

```
Please enter the power of your car (in hp): 105
The power of your car in kiloWatt: 140.7 kW
```

Solution 4

The solution exists out of two files:

- `module_conversion.py`: the module implementing the conversions between Horsepower and Watt.
- `solution_4.py`: the main script in which a user is prompted for his/her car's power in Horsepower, and the car's power in kiloWatt 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 `moduleconversion.py`

```python
def hp_to_kw(hp):
  kw = 1.34 * hp
  return kw

def kw_to_hp(kw):
  hp = kw / 1.34
  return hp
```

Score calculation

9 points to be obtained:

1. Creation of a separate module file (1 point)

Listing 2 `solution4.py`

```
import moduleconversion as mc

hp = input("Please enter the power of your car (in hp): 105")
kw = hp_to_kw(hp)
hp = print(f"The power of your car in kiloWatt: {round(kw, 1)} kW")
```

2. Using the correct expression for the conversion forth and back (1 points)
3. Function definitions implementing the conversions (1 points)
4. Importing the module (1 point)
5. Prompting the user for input (1 point)
6. Converting input to a number (1 point)
7. Calling the conversion function to obtain the power in kW (1 point)
8. Enabling similar output to the example (1 point)
9. Script runs without or with only trivial errors (1 point)

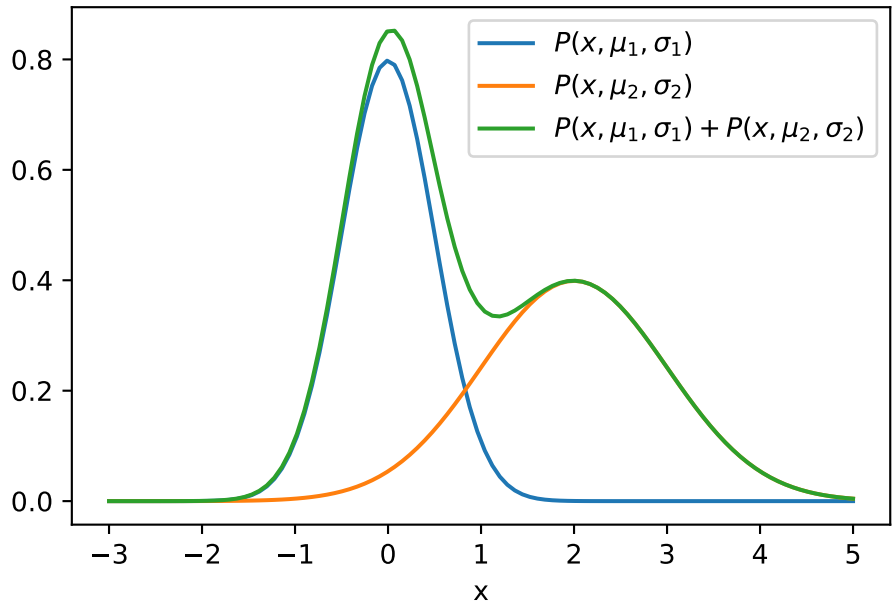
Question 5: Plot Gaussian probability density functions

Consider the Gaussian probability density function $g(x)$, with parameters mean μ and standard deviation σ, defined as:

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Plot this Gaussian function for two different parameter sets of mean and standard deviation, where: ($\mu_1 = 0$, $\sigma_1 = 0.5$) and ($\mu_2 = 2$, $\sigma_2 = 1$). Plot both these probability density functions as a smooth curve. Then add a curve representing the sum of the two Gaussian functions. 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 $[-3, 5]$ (take a sufficiently high number of x values to obtain smooth curves). **Save the plot** under the file name: `output_plot_gauss.png`. Save your solution as a script with file name: `solution_5.py`.



Solution 5

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

mu_1 = 0
sigma_1 = 0.5
mu_2 = 2
sigma_2 = 1

x = np.linspace(-3, 5, 100)
y_1 = 1 / (sigma_1 * np.sqrt(2 * np.pi)) * np.exp(-(x - mu_1)**2 / (2 * sigma_1**2))
y_2 = 1 / (sigma_2 * np.sqrt(2 * np.pi)) * np.exp(-(x - mu_2)**2 / (2 * sigma_2**2))
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)
ax.set_xlabel("x")
```

```
ax.legend([r"$P(x, \mu_1, \sigma_1)$",
           r"$P(x, \mu_2, \sigma_2)$",
           r"$P(x, \mu_1, \sigma_1) + P(x, \mu_2, \sigma_2)$"])
plt.show()
```

Score calculation

10 points to be obtained:

1. Knowing how to apply the correct function (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 Gaussian (1 point)
5. Obtaining the sum from the separate Gaussians (1 point)
6. Having all three curves (1 point)
7. Having a smooth plot (1 point)
8. Style of the plot looks like the example (1 point)
9. Enabling saving the plot (1 point)
10. Script runs without or with only trivial errors (1 point)