PHOT 110: Introduction to programming

Final exam questions & solutions, version A

Michaël Barbier, Spring semester (2024-2025)

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 100/7. We then add 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:

$$S = \sum_{i=1}^{7} [20 \times m_i/M_i]$$

In the score calculation of the solutions to the questions, we appoint different amount of points. However, every question has the same weight (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: Loops

Write a script that prompts the user to input a word and then print every character of the reversed word on a separate line to the console. Use a loop (e.g. `for` or `while`) structure to perform this task. The output should look similar to:

```
Please provide a word or sentence: Computer
r
e
t
u
```

```
p
m
o
C
```

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

Q1: Solution

```
word = input(f"Please provide a word or sentence: ")

for a in reversed(word):
  print(a)
```

Q1: Score calculation

6 points to be obtained:

1. Allowing for user input (1 point)
2. Having multiple lines (1 point)
3. Printing all the characters (1 point)
4. Automatically reversing the order of the characters (1 point)
5. Using a loop structure to print one character per line (1 point)
6. Script runs without or with only trivial errors (1 point)

Question 2: Tables of multiplication

Create a script that repeatedly provides the user with a simple calculus question such as `5 x 7 = ?` to train the tables of multiplication, and verifies his/her answer. Generate the numbers (between 1 and 10), at random as exemplified in the following code:

```
import random

a = random.randint(1, 10)
print(a)
```

```
5
```

- Whenever the user provides a correct answer, provide a next question,
- When the user provides a wrong answer or gives invalid input, let him/her try again the same question.
- If the user writes the word `stop`, stop the program.

```
Tables of multiplication (write "stop" to stop exercising).
3 x 5 = ? : 35
Correct!
6 x 3 = ? : blah
The provided answer is not valid! Please try again.
6 x 3 = ? : 18
Correct!
1 x 8 = ? : 1
This answer is wrong!
1 x 8 = ? : 8
Correct!
4 x 2 = ? : stop
Goodbye!
```

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

Q2: Solution

```python
import random

print('Tables of multiplication (write "stop" to stop exercising).')

correct = True
while True:
    try:
        if correct:
            a = random.randint(1, 10)
            b = random.randint(1, 10)
        answer = input(f"{a} x {b} = ? : ")
        if answer == "stop":
            break
        correct = int(answer) == a * b
        if correct:
            print("Correct!")
        else:
            print("This answer is wrong!")
    except:
        correct = False
        print("The provided answer is not valid! Please try again.")

print("Goodbye!")
```

Q2: Score calculation

7 points to be obtained:

1. Prompt the user for input (1 point)
2. Convert to an integer (1 point)
3. Verify whether the answer is correct (1 point)
4. Allow the user to try again if input is not valid (1 point)
5. Repeatedly prompt the user for exercises (1 point)
6. Stop when the user types `stop` as input (1 point)
7. Script runs without or with only trivial errors (1 point)

Question 3: Correct a Python script

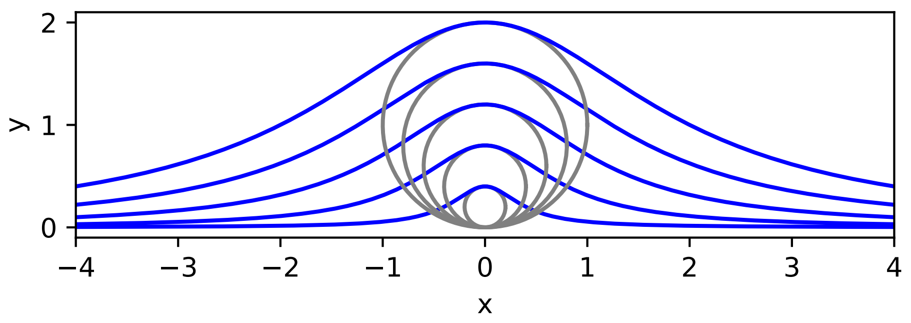
Open the script with name: `script_with_errors.py` and correct the errors. Save the corrected script with file name `solution_3.py`.

The corrected script should plot the "Witch of Agnesi" curve together with the corresponding circle:

$$y = \frac{8a^3}{x^2 + 4a^2}$$

The curve is plotted for various radii $a = 0.2, 0.4, 0.6, 0.8, 1$. 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.
# Correct the errors so that it gives the intended output.

# Load libraries
import numpy as np
from numpy import pi, cos, sin
import matplotlib.pyplot as plt

t = np.linspace(0, 2 * pi, 200)
x = linspace(-4, 4, 200)
radii = [0.2, 0.4, 0.6, 0.8, 1]]

fig, ax = plt.subplots()

for a in radii:
    # plot the circle
    xc = a * \cos(t)
    yc = a * sin(t) + a
    ax.plot(xc, ycc, color="gray")

    # plot the curve
     y = 8 * a**3 / (x**2 + 4 * a**2)
    ax.plot(x, y, color="blue")

ax.set_aspect("equal")
ax.set_xlabel("x")
ax.set_ylabel(y)
 ax.set_xlim([-4, 4])
```

```python

# Save the resulting plot
fig.savefig("output_script_with_errors.png")
```

Q3: Solution

Procedure to reach to the solution:

- On line 10: Change `linspace()` into `np.linspace()`.
- On line 11: Remove extra "]" bracket at the end of the line.
- On line 17: remove \ before cosine function (`cos`).
- On line 19: Change `ycc` into `yc`.
- On line 22: Remove the extra space at the beginning of the line (single "tab" is required).
- On line 27: Change `y` into string `"y"`.
- On line 28: Remove the extra space/indentation at the beginning of the line.

```python
# This is the corrected script.


# Load libraries
import numpy as np
from numpy import pi, cos, sin, sqrt
import matplotlib.pyplot as plt

t = np.linspace(0, 2 * pi, 200)
x = np.linspace(-4, 4, 200)
radii = [0.2, 0.4, 0.6, 0.8, 1]

fig, ax = plt.subplots()

for a in radii:
  # plot the circle
  xc = a*cos(t)
  yc = a*sin(t) + a
  ax.plot(xc, yc, color="gray")
  # plot the curve
  y = 8 * a**3 / (x**2 + 4 * a**2)
  ax.plot(x, y, color="blue")

ax.set_aspect("equal")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_xlim([-4, 4])

# Save the resulting plot
fig.savefig("output_script_with_errors.png")
```

Q3: Score calculation

7 points to be obtained:

1. Allowing for multiple radii (1 point)
2. Having the correct axis labels (1 point)
3. Having the correct axis limits, and equal aspect ratio (1 point)
4. Plotting of the circles of correct size and color (1 point)
5. Plotting the correct curves and 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: 2D density plot

Plot the intensity $I(x, y)$ of a diffraction pattern from a rectangular aperture with using the formula:

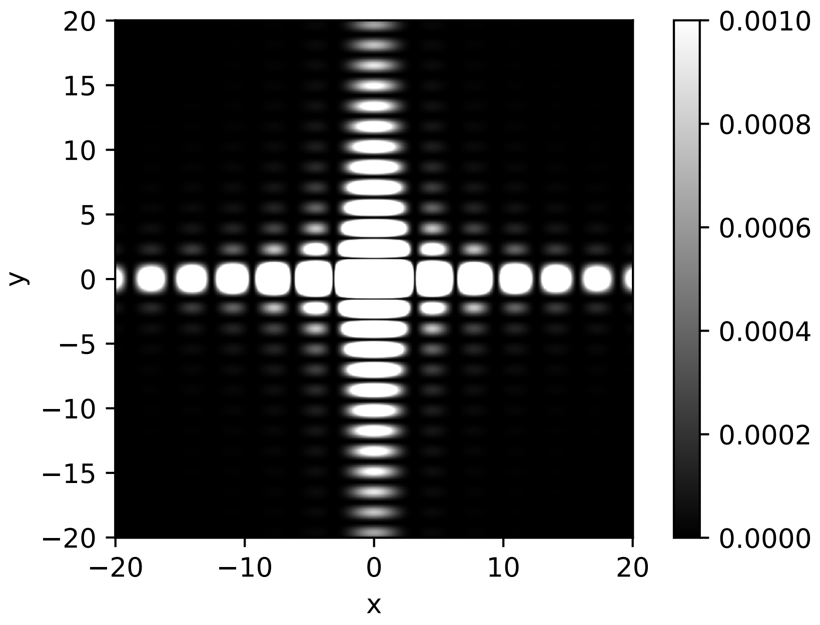
$$I(x,y) = \text{sinc}^2\left(\frac{x\,\delta y}{\pi}\right)\,\text{sinc}^2\left(\frac{y\,\delta y}{\pi}\right)$$

whereby you can use the Sinc function of Numpy called `sinc(x)` available in Numpy. Take parameters $\delta y = 1$ and $\delta y = 2$. Plot the resulting function $I(x, y)$ 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, vmax=0.001, cmap="gray")
fig.colorbar(pc, ax=ax)
```

Seeting the arguments `vmax=0.001` and `cmap="gray"` gives the same z-limit and colormap as in the output plot shown below.

For each of the functions you can use the same (x, y)-values within an 2D interval/domain with $x \in [-20, 20]$ and $y \in [-20, 20]$ (take a sufficiently fine grid of (x, y) coordinates to obtain a smooth density plot). **Save the plot** under the file name: `output_plot_sinc.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:



Q4: Solution

```
import numpy as np
from numpy import pi, sinc, log
import matplotlib.pyplot as plt
import matplotlib as mpl

I0 = 1
dx = 1
dy = 2
x = np.linspace(-20, 20, 1000)
y = x
xx, yy = np.meshgrid(x, y)
rr = xx ** 2 + yy ** 2
zz = I0 ** 2 * sinc(xx*dx/pi) ** 2 * sinc(yy*dy/pi) ** 2

fig, ax = plt.subplots()
pc = ax.pcolormesh(xx, yy, zz, rasterized=True, vmax=0.001, cmap="gray")
ax.set_aspect("equal")
ax.set_xlabel("x")
ax.set_ylabel("y")
fig.colorbar(pc, ax=ax)
```

```python
# Save the resulting plot
fig.savefig("output_plot_sinc.png")
```

Q4: 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 (1 point)
3. Implementing the correct formula and obtaining correct z-values from chosen (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)

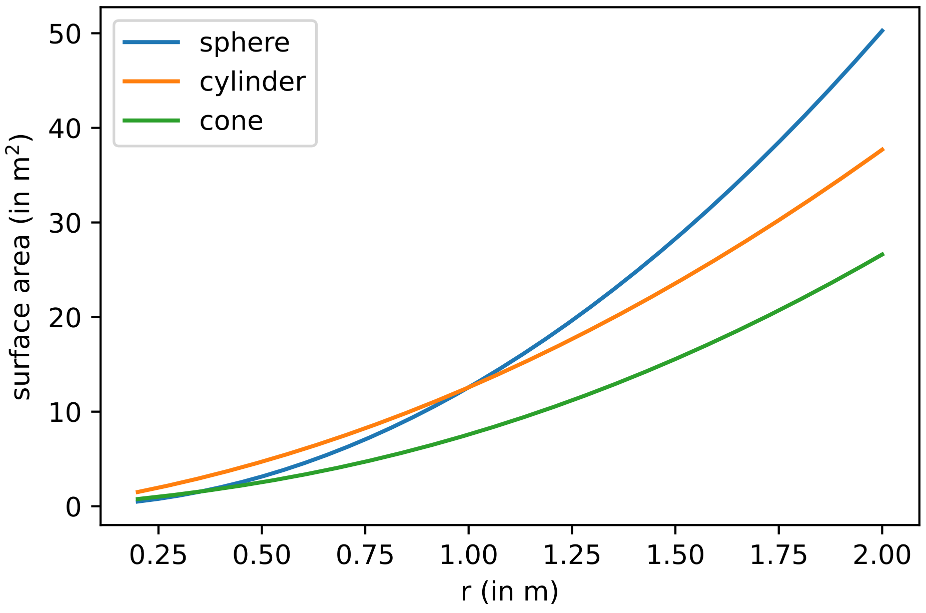
Question 5: Creating and using modules

Create a **module** (a separate script file) with file name `module_shapes.py` which helps to calculate areas of various geometrical shapes. The module should contain three functions:

- `surf_sphere(r)`: Calculates the surface area A of a sphere with radius r: $A = 4\pi r^2$
- `surf_cone(r, h)`: Calculates the surface area of a cone with radius r and height h: $A = \pi r^2 + \pi r \sqrt{h^2 + r^2}$
- `surf_cylinder(r, h)`: Calculates the surface area of a cylinder with radius r and height h: $2\pi r^2 + 2\pi r h$

Afterwards, import and use this module in a script (with file name: `solution_5.py`) that plots the surface area of a sphere, cone, and cylinder as function of the radius r in meter (putting the height $h = 1$ meter), by using the `surf_sphere(r)`, `surf_cone(r, h)`, and `surf_cylinder(r, h)` function from the module. Save the plot as a png-file with file name `output_plot_shapes.png` to the hard disk.

The output graph should look as the plot below:



Q5: Solution

The solution exists out of two files:

- module_shapes.py: the module providing functionality to calculate the surface areas.
- solution_5.py: the main script which plots the surface area as function of radius.

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 `moduleshapes.py`

```python
from numpy import pi, sqrt

def surf_sphere(r):
  return 4 * pi * r**2

def surf_cylinder(r, h):
  return 2 * pi * r**2 + 2*pi*r*h

def surf_cone(r, h):
  return pi*r**2 + pi*r*sqrt(h**2 + r**2)
```

Listing 2 `scriptsolution5.py`

```python
#| output: false
import matplotlib.pyplot as plt
import numpy as np

from module_shapes import surf_sphere, surf_cylinder, surf_cone

h = 1
rr = np.linspace(0.2, 2, 1000)

# Calculate surface areas
y_sphere = surf_sphere(rr)
y_cylinder = surf_cylinder(rr, h)
y_cone = surf_cone(rr, h)

# Plot
fig, ax = plt.subplots()
ax.plot(rr, y_sphere)
ax.plot(rr, y_cylinder)
ax.plot(rr, y_cone)
ax.set_xlabel("r (in m)")
ax.set_ylabel(r"surface area (in m$^2$)")
ax.legend(["sphere", "cylinder", "cone"])
fig.savefig("output_plot_shapes.png")
```

Q5: Score calculation

10 points to be obtained:

1. Creation of a separate module file (1 point)
2. Implementing the correct expressions for the surface areas (1 point)
3. Function definitions implementing the different surface area calculations (1 point)
4. Importing the module (1 point)
5. Creating x coordinates within the correct interval (1 point)
6. Calling the different functions from the module (1 point)
7. Having a smooth plot (1 point)
8. Enabling a similar output plot as in the question (1 point)
9. Saving the output plot to a file (1 point)
10. Script runs without or with only trivial errors (1 point)

Question 6: Dictionaries

Define the following dictionary with costs of four friends: Jack, An, Boris, and Nancy, who went on a trip ("Nancy" didn't pay anything yet). Use the following code in your script:

```python
costs = [
    {"cost": "dinner", "who": "Jack", "amount": 360},
    {"cost": "train", "who": "An", "amount": 1240},
    {"cost": "supermarket", "who": "Jack", "amount": 400},
    {"cost": "hotel", "who": "Boris", "amount": 4800},
]
```

Then let the script automatically calculate the total cost of the trip and the amount of money (in turkish lira) that each person still has to pay to (or should get from) the others. That is, the amount they already paid minus the total cost divided by four.

The output of a correct script should be the following:

```
Balance sheet:
  - Jack: -940.0 TL.
  - An: -460.0 TL.
  - Boris: 3100.0 TL.
  - Nancy: -1700.0 TL.
```

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

Q6: Solution

```python
# cost dictionary copied from question description
costs = [
    {"cost": "dinner", "who": "Jack", "amount": 360},
    {"cost": "train", "who": "An", "amount": 1240},
    {"cost": "supermarket", "who": "Jack", "amount": 400},
    {"cost": "hotel", "who": "Boris", "amount": 4800},
]

# Calculate total cost and individual costs
total_cost = 0
individual_paid = {"Jack": 0, "An": 0, "Boris": 0, "Nancy": 0}
n_people = len(individual_paid)
```

```python
for cost in costs:
    total_cost += cost["amount"]
    individual_paid[cost["who"]] += cost["amount"]

# Print balance sheet
print(f"Balance sheet:")
for k, v in individual_paid.items():
    print(f"  - {k}: {v - total_cost/n_people} TL.")
```

Q6: Score calculation

9 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 costs in the list (1 point)
4. Automatically calculate the total cost (1 point)
5. Automatically calculate the individual paid costs (1 point)
6. Printing of the balance sheet with separate names on separate lines (1 point)
7. Printing each person's name together with his/her balance (1 point)
8. Enabling similar output to the example (1 point)
9. Script runs without or with only trivial errors (1 point)

Question 7: Classes

Create a class named `Spectroscope` which has two attributes:

- `name`: the name or identifier of the device, e.g. "spectro1000xb" or "heatsensor-ST23",
- `data`: contains the measurement data (once loaded in). Initially it is `None`.

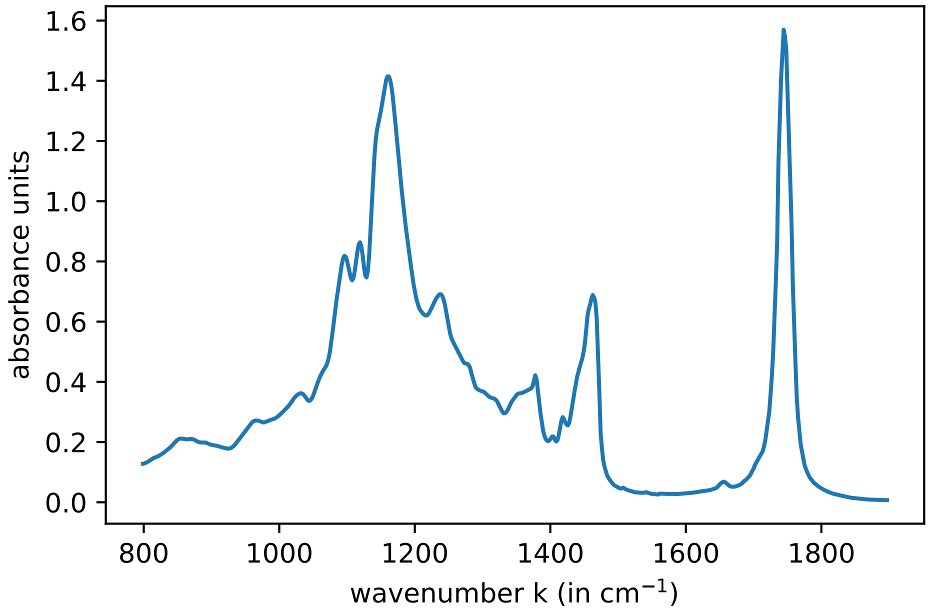
Give the class a constructor that accepts one argument (in addition to the mandatory `self` argument), the name (`name`) of the device: `__init__(self, name)`. Further create the following two methods:

- `load_data(self, data_file)`: which loads the spectroscopy data. The data is saved as a ".txt" file containing tab-separated values, organized in two columns (wavenumber and absorbance). This data can be loaded into a Numpy array using the `loadtxt(data_file)` function of Numpy.
- `plot(self)`: which visualizes the data by plotting the spectrograph (see the example output below).

Then use the class in your script to make a Spectroscope object with identifier “sx100”, load, and plot the provided data (FTIR spectrum of virgin olive oil: “FTIR_olive_oil.txt”) using the following code:

```
spectro = Spectroscope("sx100")
spectro.load_data("FTIR_olive_oil.txt")
spectro.plot()
```

The (correctly working) script should give the following output figure:



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

Q7: Solution

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

class Spectroscope():

  def __init__(self, name):
    self.name = name
    self.data = None
  def load_data(self, data_file):
    self.data = np.loadtxt(data_file)

  def plot(self):
    x = self.data[:, 0]
    y = self.data[:, 1]
    plt.plot(x, y)
    plt.gca().set_xlabel(r"wavenumber k (in cm$^{-1}$)")
    plt.gca().set_ylabel("absorbance units")
    plt.show()

# Script part copied from question description
spectro = Spectroscope("sx100")
spectro.load_data("FTIR_olive_oil.txt")
spectro.plot()
```

Q7: Score calculation

10 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 device name (1 point)
4. Having the method `load_data(self, data_file)` which loads the data (1 point)
5. The `load_data(self, data_file)` method correctly loads the data into `self.data` attribute (1 point)
6. Having the `plot(self)` method (1 point)
7. The `plot(self)` method gives a correct plot (1 point)
8. Succesfully creating an object and applying the methods in the script part (1 point)
9. Enabling similar output to the example (1 point)
10. Script runs without or with only trivial errors (1 point)