Lecture “Programming”
SS 2021

Take-home exam 1 – Python

Simon Scheidegger

You have 24 hours to solve this take-home exam. The solutions to this exam sheet are due **Tuesday, March 30th, 2021, 8.59am.**

You should answer the questions inside the *.py templates that are provided for every question (in the folder **Exam**). All exam questions need to be solved by using the **Python** (3.x) programming language.

Please, once you have finished the exam, check that your code is running correctly by:
a) opening a terminal in Nuvolos,
b) going to the root of the exam folder and,
c) typing the command: ./run_exam.sh.

This command (i.e, ./run_exam.sh) will run all codes and save the console output (what you printed with the function print()) and the error message into output files.

To **submit** your exam. Please do the following:

- Keep all your code in the exam folder on Nuvolos and check that the code runs correctly, as described above.
- Take a snapshot of your instance named "***HT1_Programming_SurnameName***" on Nuvolos. **The last version of your snapshot taken before the deadline will be corrected**.
- Create an archive by running the following command on terminal: **tar -cvf HT1_SurnameName.tar Question_1 Question_2 Question_3 Question_4 run_exam.sh**
- Download your archive and submit it to Dropbox: **https://www.dropbox.com/request/UaqCeR1ltl4VVxyN3pKD**

Note that the Dropbox submission is a backup in case a problem occurs with Nuvolos. Unless there is a legitimate problem, we will ONLY correct your code on Nuvolos.

No late hand-in is accepted. A late hand-in will result in a grade of 1 (zero points) for the take-home exam.

You have to solve this take-home exam alone and therefore you have to submit your individual solutions. **No work in teams is permitted. Plagiarism will result in a grade of 1** (zero points) for this take-home exam.

Don’t forget to include comments into your source code. This will help us understand your thoughts when we correct it.

Moodle forum thread for questions during the exam:
https://moodle.unil.ch/mod/forum/view.php?id=1035124
Answering schedule: Monday, March 29th, 2021 at 10am, 1pm, 7pm and upon availability.

Good luck! Go for it!

Help for you *(Help/PrintStdOut.py)*:
Here is a code example that writes plots to files. You may have to adjust this script accordingly below in some questions.

```
########################################
"""
============
Print Stdout
============

print png to standard out

usage: python print_stdout.py > somefile.png

"""

import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

fig, ax = plt.subplots( nrows=1, ncols=1 )  # create figure & 1 axis
ax.plot([0,1,2], [10,20,3])
fig.savefig('figure.png')   # save the figure to file
plt.close(fig)    # close the figure window
plt.savefig(sys.stdout.buffer)

########################################
```

Question 1 – some finger exercises (10 Points)

a) (2 Points) Write a program **question_1a.py** that produces an approximate Meter-Yard conversion table.
Use the following approximate formula for quickly converting Meters (m) to Yards (y):

$$1 \text{ m} \approx 1.09361 \text{ y}$$

Your code should print out a table with meters ranging from 0 , 10 , 20 , . . . , 100 in the first column and the corresponding Yards in the second column.

Write this table also to a text file **meters_to_yards.txt** in a folder called **results,** such that the relative path of this file with respect to the question_1a.py code is **results/meters_to_yards.txt.**

Make an interactive program of the one implemented above in question a). Make a program that (i) asks the user for Meters and reads the number; (ii) computes the corresponding distance converted to Yards; and (iii) prints out the result to the terminal. Name of program file: **interactive_1a.py**.

b) (2 Points) Write a code called **question_1b.py** to generate a plot of this table using the library **matplotlib**. The x-axis should be labelled as “Meters”, and the y-axis should be labelled as “Yards”. The header of the plot should be in **bold**, and say “Meter–Yard conversion”. Print the said figure to a file called M_Y_conversion.png.

c) (2 Points) Given a linear equation ax + b = c, write a function ***linear_solution(a, b, c)*** that returns the solution of the equation. The returned solution should be a **float** object.

Add a docstring to this function such that when uses the command **help(linear_solution),** it would explain how to call and use this function.

Store this function in a file called **linear_solution.py**, inside a folder called **function** (so that the relative location is **function/linear_solution.py**

Import the **linear_solution** function to a file called **question_1c.py.**

Construct **two test cases** with known solutions.

d) (2 Points) Implement a Python function **poisson(k,λ)** for computing the Poisson distribution, with parameter λ>0:

$$f(k;\lambda) = \Pr(X{=}k) = \frac{\lambda^k e^{-\lambda}}{k!},$$

and store it in a file called **poisson.py**. You may use the library **math** that contains the exponential function.
k is the number of occurrences (k= 0, 1, 2,…).

Call **poisson(k,λ)** from another file called **question_1d.py** and print out the result for k ∈ [0, 5], and **λ = 1** into a file called **results/poisson.txt.**

e) (2 Points) Write a function called **second_diff** for numerical differentiation, and store it to a file called **second_diff.py**. The formula

$$f''(x) \approx \frac{\delta_h^2[f](x)}{h^2} = \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

can be used to find an approximate second derivative of an arbitrary mathematical function **f(x)** if h is small. Write a function **second_diff(f, x, h=0.001)** that returns the second-order approximation above of the second derivative of a mathematical function represented by a Python function f(x). Use the the top right formula – that is, a second-order central scheme.

In a second file **question_1e.py**, you will test this function. To do so, apply the diff function to differentiate

i) f (x) = exp(x) at x = 0,

ii) f (x) = cos(x) at x = 2π , and

iii) f (x) = ln(x) at x = 1.

In each case, write out the error, i.e., the difference between the exact second derivative and the result of the function.

Write a docstring that comments how this function should be used.

<u>Question 2 – More advanced finger exercises (10 Points)</u>

a) (2 Points) Apply a function to a vector. Given a vector v = (2, 3, − 1) and a function

f (x) = x^3 + x*exp(x) + 1,

apply f to each element in v. Then calculate f (v) as $v^3 + v * e^v + 1$ using vector computing rules. Implement a code snippet called **question_2a.py** that shows that the two results are equal.

b) (5 Points) Implement Lagrange's interpolation formula, and store this in code in **question_2b.py**. To do so, imagine we have n +1 measurements of some quantity y that depends on x : (x0 , y0) , (x1 , y1) , . . . , (x_n , y_n).
We may think of y as a function of x and ask what y is at some arbitrary point x not coinciding with any of the points x0 , . . . , x_n . This problem is known as interpolation. One way to solve this problem is to fit a continuous function that goes through all the n + 1 points and then evaluate this function for any desired x.

A candidate for such a function is the polynomial of degree n that goes through all the points. This polynomial can be written as

$$p_L(x) = \sum_{k=0}^{n} y_k L_k(x)$$ (Eq. 0)

where

$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}$$ (Eq. 1).

The notation means

$$\prod_{i=0, i\neq k}^{n} x_i = x_0 x_1 \cdots x_{k-1} x_{k+1} \cdots x_n$$ (Eq. 2).

The polynomial pL (x) is known as Lagrange's interpolation formula, and the points (x0 , y0), . . . , (x_n , y_n) are called interpolation points.

Implement functions **p_L(x, xp, yp)** and **L_k(x, k, xp, yp)** that evaluate pL (x) and Lk (x) by (Eq. 0) and (Eq. 1), respectively, at the point x. The arrays xp and yp contain the x and y coordinates of the n + 1 interpolation points, respectively. That is, xp holds x0 , . . . , xn , and yp holds y0, . . . , yn.

To verify the program, please note that Lk(xk) = 1 and that Lk(xi) = 0 holds for i = k, implying that pL(xk) = yk . That is, the polynomial pL goes through all the points (x0 , y0), . . . , (xn , yn). Implement a test function **verify(xp, yp)** that computes **|pL (xk) − yk |** (i.e., the absolute value) at all the interpolation points (xk , yk) and checks that the value is approximately zero. Call **verify** with xp and yp corresponding to 5 equally spaced points along the three curves

y = sin(x) for x ∈ [0, π],

y = cos(x) for x ∈ [0, 2π],

and

y = exp(x) for x ∈ [0, 1].

Thereafter, evaluate pL (x) for an x in the middle of two interpolation points and compare the value of pL (x) with the exact ones – that is, sin(x), cos(x), and exp(x).

Add a docstring to this function that explains how it is supposed to be used.

c) (3 Points) Plot the polynomials in exam question 2b) with **matplotlib**. As before, all generated figures should be printed into a file in a folder, e.g., **Question2/results/PLOT_1.png**.
To do so, write a **function graph(f, n, xmin, xmax, resolution=1001)** for plotting pL (x) in question 2b (cf. Eq. 0), based on interpolation points taken from some mathematical function f (x) represented by the argument f . Store this code in **L_graph.py**.

The argument n denotes the number of interpolation points sampled from the f (x) function, and resolution is the number of points between xmin and xmax used to plot pL (x).
The x coordinates of the n interpolation points can be uniformly distributed between xmin and xmax . In the graph, the interpolation points (x0 , y0) , . . . , (xn , yn) should be marked by small circles.

Test three the graphs function by choosing 5 points in [0 , π] and f as sin(x), 10 points in [0 , 2π] and f as cos(x), and 20 points in [0 , 20] and f as exp(x).

Make a module **question_2c.py** containing the p_L , L_k , verify , and plot functions. The call to verify described in question 2b) above and the call to graph described above should appear in the module's test block.

Add a docstring to this function that explains how it is supposed to be used.

Question 3 – Object-oriented programming (10 Points)

a) (2 Points) Write a class for circle (store the file in **circle.py**)

Geometric figures, such as a circle, are other candidates for classes in a program. A circle is uniquely defined by its center point (x_0 , y_0) and its radius R . We can collect these three numbers as attributes in a class. The values of x_0 , y_0 , and R are naturally initialized in the constructor.

Two methods that this class should contain are area and circumference for calculating the area πR^2 and the circumference $2\pi R$.

Write a test program called **question_3a.py** for this class that would return

$ c = Circle(2, -1, 5) #init circle with $x_0 = 2$, $y_0 = 2$, R = 5

$ print(c)
A circle with radius 5 at (2, -1) has area 78.5398

b) (4 Points) The purpose of this question is to wrap the code from 2b) and 2c) a class. We want to construct a **class Lagrange** which works like this:

```
################################################
import numpy as np
# Compute some interpolation points along y=cos(x)
xp = np.linspace(0, 2pi, 10)
yp = np.cos(xp)
# Lagrange's interpolation polynomial
p_L = LagrangeInterpolation(xp, yp)
x = 1.2
print('p_L(%g)=g' (x, p_L(x)))
print('cos(%g)=g' (x, cos(x)))
p_L.plot()
################################################
```

The plot method (leveraging matplotlib) visualizes p_L(x) for x between the first and last interpolation point (xp[0] and xp[-1]). The class you are supposed to code here is easy to make if you import the codes from Exercise 2b) and make appropriate calls to the p_L and graph functions in this module.
Also include a verify function in a file **question_3b.py** outside the class and call it as described in 2b)/2c), but now using the class instead of the p_L function directly.

c) (4 Points) In this exercise, we want to study inheritance. To do so, we want to implement a super- and subclass for a point.

A point (x, y) in the plane can be represented by a class:

```
class Point:
        def __init__(self, x, y):
                self.x, self.y = x, y
        def __str__(self):
                return '(%g, %g)' % (self.x, self.y)
```

Store this class in a file **Point_class.py**.

You are now supposed to extend the Point class to also contain the representation of the point in polar coordinates. To this end, create a subclass PolarPoint (stored in **Polar_Class.py**) whose constructor takes the polar representation of a point, (r, θ), as arguments.
For those of you not familiar with the polar coordinate system, please consult:
https://en.wikipedia.org/wiki/Polar_coordinate_system

Store r and θ as attributes and call the superclass constructor with the corresponding x and y values (recall the relations $x = r \cos \theta$ and $y = r \sin \theta$ between Cartesian and polar coordinates).

Add a __str__ method in class PolarPoint which prints out r , θ , x , and y.

Verify the implementation in a file **question_3c.py** by initializing three points and printing these points.

Question 4 – libraries (10 Points)

a) (2 Points) Write a function in a file **question_4a.py** that defines the following matrix as a NumPy array.

$$A = \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ -5 & 3 & 1 \end{bmatrix}$$

Return the matrix $A^3 + 9A^2 + 15A$, and print the result to the screen.

b) (2 Points) Write a function in a file **question_4b.py** that defines the following matrices as NumPy arrays. Use the functions presented in this section instead of np.array() to construct the matrices, then calculate the matrix product ABA.

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} -1 & 5 & 5 & 5 & 5 & 5 & 5 \\ -1 & -1 & 5 & 5 & 5 & 5 & 5 \\ -1 & -1 & -1 & 5 & 5 & 5 & 5 \\ -1 & -1 & -1 & -1 & 5 & 5 & 5 \\ -1 & -1 & -1 & -1 & -1 & 5 & 5 \\ -1 & -1 & -1 & -1 & -1 & -1 & 5 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Change the data type of the resulting matrix to np.int64 , then return it.

c) (2 Points) Write a function **question_4c.py** that defines the following matrices as NumPy arrays.

$$A = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 0 & 0 \\ 3 & 3 & 0 \\ 3 & 3 & 3 \end{bmatrix} \quad C = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

Use NumPy's stacking functions to create and return the block matrix:

$$\begin{bmatrix} \mathbf{0} & A^\top & I \\ A & \mathbf{0} & \mathbf{0} \\ B & \mathbf{0} & C \end{bmatrix}$$

where **I** is the identity matrix of the appropriate size and each **0** is a matrix of all zeros, also of appropriate sizes.

d) (2 Points) In the 20 x 20 grid below, four numbers along a diagonal line have been marked in red.

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

The product of these numbers is **prod = 26 x 63 x78 x 14 = 1788696**. Write a function **question_4d.py** that returns the greatest product of four adjacent numbers in the same direction (up, down, left, right, or diagonally) in the grid.

For convenience, this array has been saved in the file **aux_data.txt**, but in the wrong format.

Use numpy methods to load in the data, and then bring it into the correct format (20 x 20).

One way to approach the problem of computing the product (**prod**) is to iterate through the rows and columns of the array, checking small slices of the array at each iteration and updating the current largest product. Array slicing, however, provides a much more efficient solution.

e) (2 Points) A common test problem for numerical optimization is the Rosenbrock function, which is defined as

$f(x, y)=(1-x)^2 + 100\left(y - x^2\right)^2$

Write a code snipped called **question_4e.py**, where you use optimization routines from scipy, that is, import those:

from scipy import optimize as opt

Use the opt.minimize() function to find the minimum of the Rosenbrock function. Test Nelder-Mead, CG, and BFGS, starting each with the initial guess x_0 = np.array([4., -2.5]) .
Consult the online documentation to obtain how those functions have to be called.

For each method, print to the terminal whether it converged, and if so, print how many iterations it took.

Note that the hessian argument is not needed for these particular methods.