Week 1 - Assignment

Programming for Data Science 2024

Exercises for the topics covered in the first lecture.

The exercise will be marked as passed if you get **at least 10/15** points.

Exercises must be handed in via **ILIAS** (Homework assignments). Deliver your submission as a compressed file (zip) containing one .py or .ipynb file with all exercises. The name of both the .zip and the .py/.ipynb file must be *SurnameName* of the two members of the group. Example: Riccardo Cusinato + Athina Tzovara = *CusinatoRiccardo_TzovaraAthina.zip* .

It's important to use comments to explain your code and show that you're able to take ownership of the exercises and discuss them.

You are not expected to collaborate outside of the group on exercises and submitting other groups' code as your own will result in 0 points.

For questions contact: *riccardo.cusinato@unibe.ch* with the subject: *Programming for Data Science 2024*.

Deadline: 14:00, February 29, 2024.

Exercise 1 - Indentation 1 point

Fix the indentation in the following code. Output should be "[0, 1, 2]".

In [ ]:
```
N_max = 5
n = 0
n_list = []

for n in range(N_max):

if n < 3:

n_list.append(n)

print(n_list)
```

Exercise 2 - Python variables 1 point

In class, you've seen how Python variables are internally stored. Given the following piece of code, what do you expect to happen? Why? How can you check where the two lists are stored in memory? Comment and write additional code.

In [ ]:
```
# Define two lists
list1 = [1, 2, 3]
list2 = list1

# Append
list2.append(4)

print(list1)
```

What is the difference between the following list *lst* and tuple *tpl*? Comment.

In [ ]:
```
lst = [1, 2, 3]
tpl = (1, 2, 3)
```

Exercise 2 - List intersection 2 points

Write a program that takes two lists, and returns another list with the elements common to both of them. Example:

```
list1 = [1, 2, 3]
list2 = [3, 4, 5]
output = [3]
```
Additionally,

- Write the code as a function called *lists_intersection* that takes two lists as input as gives the intersection list as output.

In [ ]:
```
###
# YOUR CODE GOES HERE
###
```

Exercise 3 - Strings filter 3 points

Write a function named *strings_filter* that takes a list of strings and a number and returns another list with only the strings *shorter* than that number. Example:

```
strings_filter(["Python", "hello", "C++"], 4)   # returns ["C++"]
strings_filter(["Python", "hello", "C++"], 6)   # returns ["hello", "C++"]
```
Hint: you can accomplish this using a for loop.

In [ ]:
```
###
# YOUR CODE GOES HERE
###
```

Exercise 4 - Leap years on Mars 4 points

2024 is a leap year on Earth, meaning a year has one more day (29th of February). On Earth, leap years are those that are divisible by 4 (like 2024, 2028, etc.).

On Mars, instead, leap years are those divisible by 6 (like 2028, 2034, etc.), with some exceptions:

- if a year is divisible by 120 (e.g. 1680) or 36 (e.g. 2016), it's *not* a leap year.
- however, if the year is divisible by 600 (e.g. 1800), it *is* a leap year, independently of the previous condition.

Your task is to write a function named *is_leap_year* that asks the user to input a year, and returns True if it is a leap year on Mars, False otherwise. Example:

```
is_leap_year(2028)  # returns True
is_leap_year(2024)  # returns False (not divisible by 6)
is_leap_year(1680)  # returns False (divisible by 120)
is_leap_year(1800)  # returns True (divisible by 600)
is_leap_year(2016)  # returns False (is divisible by 36)
```
N.B. The return type must be *boolean*.

Additionally:

- add a constrain that prints "Can't calculate leap year" if the given year is before 0 or after 5000.

In [ ]:
```
###
# YOUR CODE GOES HERE
###
```

Exercise 5 - Number divisors 4 points

A divisor of a number, is another number that divides the first number without reminder, e.g. 4 is a divisor of 20. 1 and the number itself are always divisors.

Write a function named *find_divisors* that takes a number in input and returns *all* its divisors.
Example:

```
find_divisors(20)  # returns [1, 2, 4, 5, 10, 20]
find_divisors(7)  # returns [1, 7]
```

Additionally:

- write a second version of the function called *find_common_divisors* that takes two numbers and returns all the *common* divisors. Example:

```
find_common_divisors(20, 30)  # returns [1, 2, 5, 10]
find_common_divisors(5, 7)  # returns [1]
```

Hint: exploit the previous function *find_divisors* to build this one.

In [ ]:
```
###
# YOUR CODE GOES HERE
###
```