Modular programming in Python

Objectives

Development of Python modules and functions

- Implement functions
- Learn how to separate code on modules which can communicate by calling the functions
- Work with standard and compound data types in Python
- Learn how to specify and test Python code
- Use Eclipse (or other IDE) to develop Python applications

Deadline

During **lab 3**: present one function from each feature from the 1st iteration (total 2 functions)

During **lab 4**: present one function from each feature from the 2nd iteration (total 2 functions)

Beginning of **lab 5**: upload the whole solution

Requirements

- Implement the solution using feature driven development (if you **not** use modular programming you can get maximum half of the total points)
- The solution should offer a console type interface that allows the user to input the data and visualize the output
- Use only the standard and compound data types available in Python

The application should be developed along 3 consecutive iterations as follows:

1st. Iteration
   a. Implementation
      i. feature 1
      ii. feature 2
   b. Use procedural programming
   c. Give at least 10 data examples in the application (to facilitate testing)
   d. Each function should be documented and tested (at least 3 assertions/function)

2nd. Iteration
   a. Implementation
      i. feature 3
      ii. feature 4
   b. Use procedural programming

c. Give at least 10 data examples in the application (to facilitate testing)
d. Each function should be documented and tested (at least 3 assertions/function)

3rd. Iteration

a. Implementation
   i. feature 5
   ii. feature 6
b. Use modular programming (at least 2 modules: one for UI and one for the functions needed)
c. Give at least 10 data examples in the application (to facilitate testing)
d. Each function should be documented and tested (at least 3 assertions/function)

The application should allow the validation of data – when the user inputs invalid data or commands, the application should give a warning.

Use your registration number ($\boldsymbol{n_{reg}}$) to define the number of the exercise you have to solve: $\boldsymbol{n_{reg}}$ $\mathbf{mod\ 2+1}$
e.g. my registration number is 1491
$1491\ mod\ 2+1 = 1+1 = 2$
⇒ I have to solve exercises: P**2**

Problem specification

P1. Numeric arrays

A math teacher needs a program to help students test different number properties. The program manages an array of numbers and allows students to use the following features offered by the program:

1. **Add numbers in the array**
   - $add(my\_list, value)$ – $value$ as last element of my_list
   - $insert(my\_list, index, value)$ – insert number $value$ at $index$ (the index of the first element is 0)
2. **Modify elements in the array**
   - $remove(my\_list, index)$ – removes the element at $index$
   - $remove(my\_list, from\_index, to\_index)$ – removes elements between the two given index
     e.g. $remove(my\_list, 1, 3)$ – removes the elements at indices 1, 2 and 3
   - $replace(my\_list, old\_value, new\_value)$ – replaces all old_values occurances with new_value
     e.g. $replace(my\_list,\ [1,3,5],[5,3])$ – replaces all sub-arrays 1 3 5 with 5 3
3. **Get the numbers that have certain properties**
   - $prime(my\_list, from\_index, to\_index)$ – get prime number between the two given index
     e.g. $prime(my\_list, 1, 5)$ – get the prime numbers from the array found at indices 1..5
   - $odd(my\_list, from\_index, to\_index)$ – get odd number between the two given index

e.g. $odd(my\_list, 1, 5)$ – get the odd numbers from the array found at indices 1..5

4. **Obtain different characteristics from sub-arrays**
    - $sum(my\_list, from\_index, to\_index)$ – get sum of elements between the two given index
      e.g. $sum(my\_list, 1, 5)$ – get the sum of elements 1..5
    - $gcd(my\_list, from\_index, to\_index)$ – get greatest common divisor of elements between the two given index
      e.g. $gcd(my\_list, 1, 5)$– get the greatest common divisor of elements 1..5
    - $max(my\_list, from\_index, to\_index)$ – get maximum of elements between the two given index
      e.g. $max(my\_list, 1, 5)$– get the maximum of elements 1..5
5. **Filter values**
    - $filter\_prime(my\_list)$ – keep only prime numbers, remove the other elements
    - $filter\_negative(my\_list)$ – keep only negative numbers, remove the other elements
6. **Undo**
    - $undo()$ – undo the last operation that modified the array

P2. Programming competition

In a programming competition, after the evaluation of solutions, the evaluation committee records in an array the scores obtained by participants after solving the problems (at index $\boldsymbol{i}$ in the array, the score of the $\boldsymbol{i^{th}}$ participant is stored). Given that the participants to the competition had to solve 10 problems, each evaluated to a maximum of 10 points, help the committee to access the following features offered by the program:

1. **Add the result of a new participant to the array**
    - $add(score\_list, value)$ – $value$ as last element of $score\_list$
    - $insert(score\_list, index, value)$ – insert number $value$ at $index$ (the index of the first element is 0)
2. **Modify the scores in the array (as a result of appeals)**
    - $remove(score\_list, index)$ – removes the element at $index$
    - $remove(score\_list, from\_index, to\_index)$ – removes elements between the two given index
      e.g. $remove(score\_list, 1, 3)$ – removes the elements at indices 1, 2 and 3
    - $replace(score\_list, index, new\_value)$ – replaces the score on $index$ with new_value
3. **Get the participants with scores having some properties**
    - $less(score\_list, value)$ – get participants with score less than $value$
    - $sorted(score\_list)$ – get all participants sorted by their score
    - $sorted(score\_list, value)$ – get the participants with scores higher than $value$ sorted
4. **Obtain different characteristics of participants**

- $avg(score\_list, from\_index, to\_index)$ – get the average score for participants between the two given index
  e.g. $avg(score\_list, 1, 5)$ – get the average score for participants 1..5
- $min(my\_list, from\_index, to\_index)$ – get minimum score for participants between the two given index
  e.g. $min(score\_list, 1, 5)$– get the minimum score for participants 1..5
- $mul(score\_list, value, from\_index, to\_index)$ – get the score of participants between the two given index, which are multiples of $value$
  e.g. $mul(score\_list, 10, 1, 5)$– get the score of participants 1..5, which are multiples of 10

5. Filter values

- $filter\_mul(score\_list, value)$ – keep only participants with scores multiple of $value$, removing the other participants (scores)
- $filter\_greater(score\_list, value)$ – keep only participants with scores higher than $value$, removing the other participants (scores)

6. Undo

- $undo()$ – undo the last operation that modified the array

Submission

Total points: **10**

You need to submit an **archive** (e.g. .zip, .rar, etc) with the source code (**only** your own *.py* files created, without venv or other generated files) to the assignment on **Teams** before the deadline. Please use the following convention to name the archive file:

$sfmie1234\_A2.zip$, where s – first letter of your surname
f – first letter of your first name
mie – stand for mathematics informatics in English
1234 – is your matriculation number
$A2$ – number of the assignment

If something is not clear, please ask me.

Key

1p	Default
1p	Work during lab 3
1p	Work during lab 4
4p	Modules and Iteration 3 correctly implemented
1p	At least 10 data examples for each iteration
1p	At least 3 assertions for each function
1p	Documentation