Solving complex problems with Python

Objectives

Development of Python modules to solve complex problems

- Develop Python modules and classes
- Use test-driven development
- Learn how to work with exceptions
- Familiarize with special libraries e.g. numpy, matplotlib

Deadlines

- ***Lab 8****:* specified features from Iteration 1 and Iteration 2 *(work during the same lab)*
  Upload your solution online before the end of the Lab8.

- ***Lab 9****:* all features from Iteration 1 and Iteration 2 with tests in PyUnit *(homework from Lab 8)* and add new features in Iteration 2 *(work during the same lab)*
  Upload your solution online before the end of the Lab9.

- ***Lab 10****:* add controller layer and implement Iteration 3 *(homework)*
  Upload your solution online before the start of the Lab10.

Requirements

1. Implement a solution for the following problem using classes and feature driven development
2. The solution should offer a console type interface that allows the user to input the data and visualize the output
3. Use only the standard and compound data types available in Python

The application should be developed along several iterations and the solution should ensure:

- Providing at least 10 data examples in the application
- Documentation and testing of each function (at least 5 assertions)
- Validation of data – when the user introduces invalid commands or data, a warning should be generated

Problem specification

A **math teacher** needs a program that helps **students** perform different vector operations.

Iteration 1:

A vector (class ***MyVector***) is identified by the following properties:

- *name_id* given as a string/int
- *colour* given as one letter (possible values ‘r’, ‘g’, ‘b’, ‘y’ and ‘m’)
- *type* given as a positive integer greater or equal to 1
- *values* given as a list of numbers

The following features are offered by the program (to be implemented in class **MyVector**):

1. **Scalar operations**
   a. Add a scalar to a vector
      `e.g. [1,2,3] + 2 = [3,4,5]`
2. **Vector operations**
   a. Add two vectors
      `e.g. [1,2,3] + [4,5,6] = [5,7,9]`
   b. Subtract two vectors
      `e.g. [1,2,3] - [4,5,5] = [-3,-3,-2]`
   c. Multiplication
      `e.g. [1,2,3] * [4,5,5] = 29`
3. **Reduction operations**
   a. Sum of elements in a vector
      `e.g. for [1,2,3] sum is 6`
   b. Product of elements in a vector
      `e.g. for [1,2,3] product is 6`
   c. Average of elements in a vector
      `e.g. for [1,2,3] average is 2`
   d. Minimum of a vector
      `e.g. for [1,-2,3] minimum is -2`
   e. Maximum of a vector
      `e.g. for [1,2,-3] maximum is 2`

Iteration 2:

The program manages several vectors (class ***VectorRepository***) and allows operations such as:

1. Add a vector to the repository
2. Get all vectors
3. Get a vector at a given index
4. Update a vector at a given index
5. Update a vector identified by name_id
6. Delete a vector by index
7. Delete a vector by name_id
8. Plot all vectors in a chart based on the type and colour of each vector (using library *matplotlib*). Type should be interpreted as follows: 1 – circle, 2 – square, 3 – triangle, any other value – diamond.

Iteration 3:

Implement all features from iteration 1 using special libraries e.g. numpy.