Prof. Dr. Thomas Schultz
Olivier Morelle (morelle@uni-bonn.de)

B-IT
Bonn-Aachen International
Center for Information Technology
November 11, 2023

Winter term 2024/25

Computer Science for Life Scientists

Assignment Sheet 6

Solution has to be uploaded by November 20, 2023, 10:00 a.m., via eCampus

- This exercise can be submitted in **small groups** of 2-3 students. Submit each solution only once, but clearly indicate who contributed to it by forming a team in eCampus. Remember that all team members have to be able to explain all answers.
- Remember to include proper **documentation** in all your code, in particular, docstrings for functions.

If you have any questions concerning this exercise, please ask them on Monday or Wednesday, or use the forum on eCampus.

Exercise 1 (Object-oriented Programming: Rendering Text, *13 Points*)

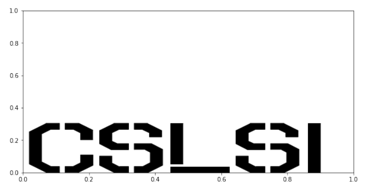

Figure 1: If you use your solution to render the string "CSLSI", the result should look similar to this example.

In this exercise, you should design and implement three classes that can work together to render a string to an image, in a given font size. This task is meant to be solved in a team, such that each team member is responsible for one of the classes.

- The first class **Curve** should be able to represent a polygonal curve as a sequence of points in the plane. It should at least provide methods to add and remove points, to translate the curve by a given 2D vector, to scale it by a given factor, and to compute a tight axis-aligned bounding box around the curve. (4P)
- Instances of the second class **Character** should represent a letter from the alphabet, which is specified during instantiation, as a suitable set of polygons. To save you the work of having to design your own font, we provide a JSON file that includes a polygonal representation of each letter. The class should permit translating and scaling the letter, and computing its bounding box. (4P)
- A third class **Text** should represent a given string by creating corresponding **Character** objects and arranging them next to each other. It should also permit changing the font size by scaling the overall text, and have a method to render it into an image. For rendering, feel free to use the **matplotlib** Python package. (4P)

The final point is given for demonstrating with two different strings in different font sizes that your classes correctly work together to create the desired output.

Note: To encourage following the information hiding principle, this task makes an exception from our general rule that each group member has to be able to explain the full solution. Assuming that each team member was responsible for at least one class, it is sufficient if you can explain your own class(es).

Exercise 2 (Operator Overloading, *7 Points*)

In this exercise, you will write a `Vector` class that supports basic '+', '-', and '*' operators using operator overloading.

- Write a class called Vector that takes a list of numbers as input to initialize the vector. (1P)
- Write a suitable function in your Vector class that allows the contents of a given Vector instance to be printed directly using the print function. (1P) For example:
  v = Vector([1,2,3])
  print(v)
  must print out
  1
  2
  3
  Verify your implementation using a test example.
- Add a suitable function in your class to overload the '+' operator that performs element-wise addition on two vectors. For example, Vector([1,2,3])+Vector([2,3,4]) should return another Vector with values 3,5,7. Similarly, write a function that allows you to use '-' to subtract two vectors element-wise. Verify your implementation using a test example. (2P)
- Finally, write a function to overload the multiplication operator '*' so that it supports both scaling the vector with an integer or floating-point number and multiplication of two vectors. In the case of vector-to-vector multiplication, calculate the dot product. Check your implementation with a test example. (3P)

For more detailed information on Python Data Model and Operator Overloading, you can refer to `https://docs.python.org/3/reference/datamodel.html`.

Exercise 3 (Debugging the Computation of Matrix Determinant, *5 Points*)

In linear algebra, the determinant of a matrix is a value that can be computed from the elements of a square matrix. One way to compute the determinant of square matrix $A_{d\times d}$ with coefficients $a_{i,j}$ is based on its so-called minors $M_{i,j}$, i.e., the determinants of submatrices in which the ith row and jth column have been removed:

$$|A| := det(A) = \sum_{j=1}^{d} (-1)^{i+j} a_{i,j} M_{i,j}$$

In this equation, you can pick i to be an arbitrary row of A; it will always yield the same result. Together with the fact that the determinant of a 1×1 matrix with a single coefficient $a_{1,1}$ is just $a_{1,1}$, this implies a recursive strategy for implementing the determinant.

The file **Determinant.py** on eCampus tries to implement this based on representing matrices as lists of lists. It contains two functions:

- **matrix_get_submatrix(m, i, j)** which should extract from a given $d \times d$ matrix m the $(d-1) \times (d-1)$ submatrix that you get if you remove row i and column j.
- **matrix_det(m)** which should compute the determinant of a given $d \times d$ matrix m by calculating it recursively.

Unfortunately, these implementations still contain some errors.

a) Define a test class using the **unittest** module that reveals the errors in the above-mentioned functions. (2P)

b) Find the bugs that have been identified by your tests and fix them, such that the resulting code successfully passes the unit tests. (2P)

c) Measure the average time needed to compute the determinants of random matrices of dimensions 1 to 10. (1P)

Good Luck!