Prof. Dr.-Ing. Birgit Vogel-Heuser

Lehrstuhl für Automatisierung und Informationssysteme

Technische Universität München

Exercises for Basic Python

WS 2023/2024

Trainer: Mingxi Zhang, M. Sc.

Exercise 1: Numpy

Read the python documentary for Numpy:
https://numpy.org/doc/stable/user/absolute_beginners.html

Task1: Create functions specified below using Numpy and print the results.

1.1 Create a function *create_array* that creates an array. An input *array_length* is an integer that determines the size of the array. The function should return a Numpy array rand_array containing random integers in the range [0, 20] as a result.

```
def create_array(array_length):



    return rand_array
```

[Figure 1] The function of Task 1.1

1.2 Create a function *append_array* that appends arrays. Inputs *array1* and *array2* are Numpy arrays. The function should return a Numpy array *array3* appending the inputs as a result. All the input arrays must have same shape. The shape of return array3 must be twice the shape of the input array1.

```
def append_array(array1, array2):



    return array3
```

[Figure 2] The function of Task 1.2

1.3 Create a function *sort_array* that sorts values of an array. An input *array* is a Numpy array with the shape (20,). The function should return a Numpy array *array_sym* with the shape (20,). To implement the function *sort_array*, first, sort the values of the input in ascending order and store a result in *array_asc*. Second, sort the input in descending order. Last, add both arrays and return *array_sym*.

```
def sort_array(array):



    return array_sym
```

[Figure 3] The function of Task 1.3

1.4 Create a function *scale* that scales all values by same factor. An input *array_in* is a Numpy array with the shape (20,). The function should return a Numpy array *array_scaled* with the shape (20,) as a result. Since we added the two arrays, values up to 40 are possible. To implement the function *scale*, first, scale all values by the same factor to make the highest value is exactly 20. Last, round all values to 2 decimals.

```
def scale(array_in):



    return array_scaled
```

[Figure 4] The function of Task 1.4

1.5 Create a function *indirect_sort* that sorts columns. An input *array_in* is a Numpy array with the shape (20,). The function should return a Numpy array *matrix* with the shape (2, 10) as a result. To implement the function *indirect_sort*, first, reshape the input *array_in* with the shape (2, 10). Second, create a Numpy array *idx* with the shape (1, 10). This array *idx* has to contain all integers from 1 to 10, but in arbitrary order. Third, insert the newly created array as the second row of the input *array_in*. Last, sort the columns of the matrix by the first row.

```
def indirect_sort(array_in):



    return matrix
```

[Figure 5] The function of Task 1.5

1.6 Create a function *matrix_sum* that computes sum of all elements of an input, sum of its columns, and rows. An input *matrix_in* is a 2D Numpy array with an arbitrary dimension. The function should return results *sum_all, sum_row,* and *sum_column*. To implement the function *matrix_sum*, first, sum all elements of the input *matrix_in* to return *sum_all*. Second, sum each row of the input *matrix_in* to return *sum_row*. Last, sum each column of the input *matrix_in* to return *sum_column*.

```
def matrix_sum(matrix_in):



    return sum_all, sum_row, sum_column
```

[Figure 6] The function of Task 1.6

1.7 Execute all 6 functions and print the results using Numpy_Exercise_Template.py.

Exercise 2: ControlFlow

Read the python documentary for Control Flow Tools chapter 4.1 to 4.6:
https://docs.python.org/3/tutorial/controlflow.html

Tasks:

- Task1:
  - Complete the function *task1(database)*
  - *Database* should only contain positive numbers. All negative numbers should be replaced by 0.
  - Return the modified *database*

- Task2:
  - Complete the function *task2(number)*
  - This function should determine whether the argument *number* is a prime number or not and print the result: *X is (not) a prime number*. Stop computing as soon as possible.
  - Return *True*, if number is a prime, and *False* if not

- Task3:
  - Complete the function *task3()*
  - This function creates an array with three random numbers between 0 and 2
  - Determine whether:
    - All numbers are the same
    - The first and the last number are the same
    - The first two numbers are the same
    - The last two numbers are the same
    - All numbers are different

    and print the result

Exercise 3: Functions And Classes

Read the python documentary for Classes:
https://docs.python.org/3/tutorial/classes.html
Repeat the chapters on functions from part 1.

Tasks:

- Task1:
  - Start from scratch and import numpy
  - Define two classes for geometric objects: *Circle* and *Rectangle* with Radius and Width and Length parameters, respectively. Parameters should be initialized to 5, if no value is given, when creating an object of the class
  - For each class implement the functions *compute_area()* and *compute_perimeter()*, which should return the computed value.
  - Define the function *generate_objects(number)* which is called from main() with *number*=10 and gets the number of objects as input and returns a list of *number* of randomly generated geometric objects with random parameters (int, range 1 to 10)
  - Calculate and print the mean area and the sum of the perimeters of all objects

- Task2:
  - For this task use *FunctionsClasses_Task2_Template.py*
  - It can be shown, that the sequence $a_n = n^2, n \in \mathbb{N}$ can be expressed as a recursive sequence of third order: $a_{n+3} = 3a_{n+2} - 3a_{n+1} + a_n$
  - Write a function *square_numbers(n)*, that calculates n^2 for a given input n using the recursive formula

Exercise 04: Input And Output

Read the python documentary for Input and Output:
https://docs.python.org/3/tutorial/inputoutput.html

Tasks:

- Task1:
  - You are provided with the file *PythonWikipedia_reversed.txt.* Have a look at the file. As you can easily recognize, all words of a line are ordered backwards. (Source: https://en.wikipedia.org/wiki/Python_(programming_language))
  - Complete the function *python_wiki_text()* such that it creates a new file named *PythonWikipedia.txt*, where the text is stored in a readable form. When writing the file don't create whitespaces at the end of a line!
  - Furthermore, the function should count the number of lines, words characters and characters including whitespaces of the original file and print the result.

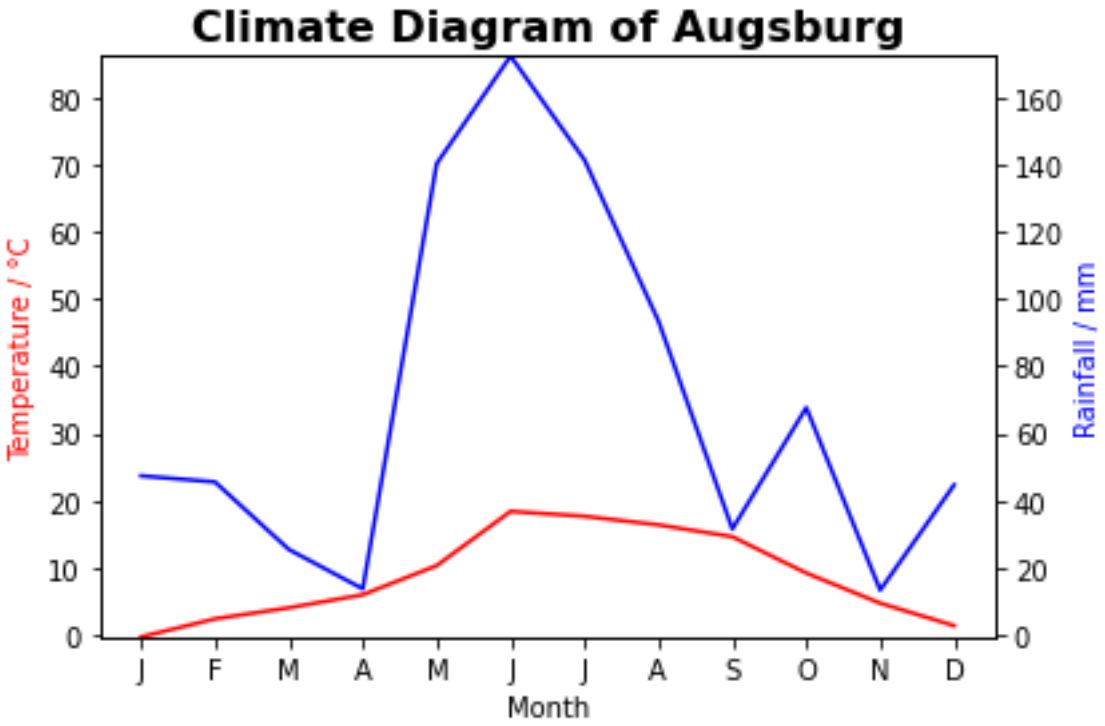
Exercise 05: StudentDatabase

You are provided a database of students *Students.txt* and an example file *Grades_Hotdog_Henry.txt*.

Tasks:

- Start from scratch. **Do not import any libraries! Only use python build-in functions!**
- Create a class Student:
  - Class attributes are:
    - first_name: string
    - last_name: string
    - grades: list of floats
    - bonus: string
    - email: string
    - avg_grade: float
    - passed: bool
  - Create the functions:
    - *compute_grades(self)*: sets *avg_grade* and *passed*; rules: if any single grade is higher than 4.0 the student failed the course, and the average grade is 5.0. If the student passed the course compute the average grade and round to one decimal. If the student has passed the course and achieved the additional homework bonus, there is a bonus of 0.3 for the average grade
    - *create_email(self)*: each student is required to have an email in the format *FirstLetterOfFirstName.LastName@tum.de* (e.g.: *P.Pasta@tum.de*). This function creates the email address and stores it in the class attribute *email*
    - *create_student_file(self)*: this function creates a text file similar to *Grades_Hotdog_Henry.txt*; The last line should specify whether the student has *passed* or *failed* the course
- *Create a function evaluate_database()*:
  - Read the database file
  - For every student create and initialize a new object of type *Student* and store all those objects in a list (Be careful: Your code should work for a variable number of grades)
  - Execute all 3 functions of class Student for every student in the list
  - Evaluate the database and print the result:
    The average grade of the course was X.XX
    XX.X % of the students passed the course
    The best student was FirstName LastName with an average grade of X.X
- Run *evaluate_database()* in the main function

Exercise 06: Climate Diagram Generator



You are provided with three databases containing weather data from different locations in Bavaria. Each file contains exactly one year of measurement data but not starting at 1st of January but at 1st of another month. The raw data can be found on: https://www.dwd.de/DE/leistungen/klimadatendeutschland/klimadatendeutschland.html

Relevant columns:

- JJJJMMDD: date of measurement in format YYYYMMDD
- TM: mean temperature in 2 m height above ground
- SO: Sum of sunshine duration
- RR: Amount of rainfall

Tasks:

- Start from scratch. Import *numpy* and *matplotlib.pyplot*
- Create a global variable *LOC_NAME* for the location. E.g. LOC_NAME = 'Augsburg'
- Create a function *read_text()*:
  - This function reads the database for the location set in LOC_NAME
  - The function returns a tuple *data* containing the variables:
    - date: list of strings containing the date of measurement
    - T_avg: numpy array containing the average temperature for each day

    - sun_time: numpy array containing the sun time of each day in hours
    - rainfall: numpy array containing the amount rainfall of each day in mm
  - All variables should be ordered beginning with the oldest measurements
- Create a function *evaluate_sun_data(data)*:
  - This function analyses the sun time of the location and prints a text like this:

```
In total there were 1772.1 hours of sun in Straubing, which on average were 4.86 hours per day.
The sunniest day was on 2020/06/12 with 14.7 hours of sun.
```

- Create a function *calculate_month(data)*:
  - This function calculates the average temperature and the sum of rain for each month
  - Return two numpy arrays containing the 12 values of mean temperature and the values of accumulated rainfall. Make sure the values are ordered by months starting with January. Remember that not all databases start on the same date!
- Create a function *plot_climate_diagram(temp, rain)*:
  - This function generates a climate diagram with the same layout as the example diagram above:
    - Title: bold, size 16
    - Y-axis labels: Color red / blue
    - Y-axis values: The value for *Rainfall* has always to be twice the value of *Temperature* at the same height. Reason: https://de.wikipedia.org/wiki/Klimadiagramm#Walter/Lieth-Klimadiagramm_(hygrothermisch)
  - Save the climate diagram as 'Climate_Diagram_LOC_NAME.png' to your working directory
- Call all previously created functions