Python Exercises

The following questions are to be prepared till the fourth session of the course. At the beginning of the session, a list is handed through, where each student will tick his/her individual achievements as a basis of grading. Students will be randomly picked from the present ones and have to present (defend) their solution. The student will be interviewed to his/her solution. In case there are clear doubts, that the student can not explain the presented work, no exercise of this sheet will be accounted towards the final grade. The solution will be further discussed with the plenum, for a common learning experience.

Ex. 1 Python Package Create a python package, called `aaip` that holds all of the code for all the following exercises. In this package, each of the following exercises gets a module called `ex1`, `ex2`, `ex3` and so on.

Your package shall contain a function in the first module, called `hello_python_exercises(...)`. This function shall take the following arguments:

- an integer `num_students` and
- a string `student_name`

The function paramater `student_name` shall have a default argument, namely your personal name.

When calling the function, a message shall be printed to the user, that prints (something similar like): "This course has <`num_students`> and one of those is <`student_name`>."

Depending on the arguments provided to the method, different output shall be printed.

To demonstrate your first package and module, create a script outside the scope of the package, import the module and call the function with different parameter.

Ex. 2 Simple File I/O Create a moudle that handles a very simple database of students with several functions:

- a function that opens, reads, and returns the content of a csv file as a list. The default value for the file name is `students_list.csv` and contains in each row an id, a name, and a grade. The functions reads every line and stores each *student* in a dictionary with the keys id, name, and grade. The sum of all students is stored in a database (i.e. a trivial list) that is returned.
- a function, that takes a database as a parameter and ads iteratively new students to that list. The function adds students to the database, as long as no *empty* student name (empty string) is passed. Think about an appropriate way on handling the assignment of new identifiers.
- a search method, that searches for either identifiers or students and outputs the id, name, and grade in a nicely formatted way.
- a method to search for a specific student and change its grade.
- a method to store the database back to a file. The default value for the file name is `students_list.csv` and is compatible to the input file, so this file can be read again.

Test your module by creating a script outside the scope of your package and test through all the methods.

Ex. 3 OOP Create a class `Person` in the new module. The class should have attributes name, age, and address. Implement a method `greet()` that returns a string greeting the person by their name.

Extend the module by adding a class `Student` that inherits from `Person`. The `Student` class should have an additional attribute university and override the `greet()` method to include the university name in the greeting.

Add a factory method / class method `from_string()` (**Attention:** class method, no instance method) to the `Person` class that allows creating a `Person` object from a string input in the format `"name,age,address"`. Modify the module accordingly.

Ex. 4 Complex Number Class with Operator Overloading

Create a `ComplexNumber` class that represents a complex number (plain, do not use the data type complex; represent a complex number with two attributes, real and imaginary) and supports basic arithmetic operations (addition, subtraction, multiplication, division) through operator overloading.

Requirements: it must be possible to add, substract, multiply, and divide complex numbers like integers, like `a+b`, `a-b`, `a*b`, and `a/b`.

Extend the class in such a fashion, so when you print an instance of the class, the complex number is displayed in the form a + bi.

Write a function that takes a list of ComplexNumber objects and returns their sum using a list comprehension.

Hint: for operator overloading have a look at this link from the Python Reference.

Test your module by creating a script outside the scope of your package and test through all the methods.

Ex. 5 File and Directory Management using Python's os Package In this exercise, you will create a Python script that utilizes the `os` package extensively for file and directory management operations. The script will involve creating, modifying, traversing, and removing files and directories. You will also handle file metadata and permissions, and make use of the os package to interact with the operating system. The Task:

(a) **Directory and File Creation**
Write a function `create_project_structure(base_path, project_name)` that creates a directory structure for a new project. The structure should include:
- A root folder with the project name.
- A subfolder named `src/` for source code.
- A subfolder named `docs/` for documentation.
- A file named `README.md` inside the root folder.
- A file named `main.py` inside the src/ folder.

Requirements:
- If the project folder already exists, print a message indicating that.
- Ensure that the directories are created recursively.
- Write placeholder content into `README.md` and `main.py`.

(b) **Directory Traversal**
- Write a function `traverse_directory(path)` that recursively traverses the directory structure starting from path, printing out each file and folder.
- The output should clearly distinguish between files and directories.
- Use `os.walk()` to traverse the directory.

(c) **File Metadata and Permissions** Write a function `file_metadata(file_path)` that:
- Retrieves and prints the file size, last modified time, and creation time.
- Changes the file permissions to make it read-only, then changes it back to be writable.

Use `os.stat()` for retrieving metadata, and `os.chmod()` for changing permissions.

(d) **Find Files by Extension**
- Write a function `find_files_by_extension(path, extension)` that searches for all files with the specified extension (e.g., `.py`, `.md`) within the given directory (and its subdirectories).
- Return a list of full paths to the files found.
- Use `os.walk()` to perform the search.

(e) **Deleting Files and Directories**
- Write a function `delete_files_and_directories(path)` that:
  - Deletes all files in the directory.
  - Deletes all subdirectories.
  - Deletes the directory itself.
- Ensure the function handles nested directories and files.
- Use `os.remove()` to delete files and `os.rmdir()` for directories.

Test your module by creating a script outside the scope of your package and test through all the methods.