Exercises for Python Expert

# Day 1

## Exercise 1.1 - Decorator

Implement a decorator that can be used to debug a function. Whenever the function is called the decorator should print:

* a timestamp when the function was called
* the name of the function that was called
* the arguments and the values of the arguments of the call
* the time it took to execute the function
* the return value

Test the decorator with several functions and various arguments.

## Exercise 1.2 - Dataclass with properties

Build a class to represent a user.

Implement the following (non-public) attributes:

* username
* email (validate if this is a valid e-mail address with a regular expression)
* full name
* password hash (store the hash of a provided password)
* created (a timestamp)
* last\_login (a timestamp that is update at each login)
* status (indicating if the user is active or not)
* token (a random token that can also be used to validate the user)

Implement properties as required for some of the attributes.

Also implement methods to access and work with your user model. Suggestions:

* set\_password()
* validate\_password()
* activate()
* deactivate()
* generate\_token()
* revoke\_token()

## Exercise 1.3 - Project Setup

Build a project directory for your project. Include the following:

* a src directory
* a models directory in the src directory
* a tests directory
* a main.py to start your project.

Also add:

* a readme.md file with a short description of your project and usage instructions
* a config.ini file with settings for your project
* git support

Put you User class from the previous exercise in de models directory.

## Exercise 1.4 - Unittests

Implement unittests in the test directory. Use the unittest library. Test the functionality of the User class.

# Day 2

## Exercise 2.1 - Model persistence

Add a persistence directory to the src directory in your project.

First implement the persistence by using the SQLite library with SQL statements.

Add a module with a UserModel class that uses the User class you have previously created and provides functionality to preform CRUD operations:

* add a new user to the database (INSERT)
* select all users (SELECT & ORDER BY)
* select a user by username (SELECT & WHERE)
* update a user (UPDATE)
* delete a user (DELETE)

## Exercise 2.2 - Model persistence with SQL Alchemy

In a second module in the persistence directory implement a class UserModel similar to the previous exercise but no use SQLAlchemy the provide the CRUD operations

## Exercise 2.3 - TkInter GUI for model

Build a TkInter GUI to add new users. Put all TkInter code a new “gui” directory in your “scr” directory.

# Day 3

## Exercise 3.1 - Flask

Implement a collection of book titles that will be displayed on a webpage. Books can be added or deleted from the list. And the list can be filtered by entering the name of an author or by keywords.

You are free to choose any other type of items to work with. Do not make it to complicated.

Setup a Flask application by adding a web directory to you “src”. In that directory add a “templates” directory and a “static” directory. Add a controller.py file that will implement the provided url endpoints.

In the templates directory setup a base.html file with your basic page layout and a content.html file that will display the information you request.

Implement the flask controller to render your information when the endpoint is invoked.

## Exercise 3.2 - Login page

Add a login page to your project. Validate the username and provided password with the User model you implemented before.

## Exercise 3.3 - Flask RESTful API

Implement a REST endpoint to get information from your application. Use a token to validate if the user is allow to request the information. For example build a weather forecast webpage from a weather API (e.g.openweathermap.org)