Week 8

Learning objectives

You will be able to apply object-oriented programming using the Python programming language:

- Create dataclasses including attributes (with the correct scope of access), properties, methods __init__(), __del__(), __str__(), __repr__(), ...
- Use of class attributes
- Use of class methods (=static methods)
- Association between two classes
- Testing of dataclasses

GitHub

All your solutions from week 8 should be posted on Github. To do so, follow the procedure explained on Leho. After each exercise, you can do a 'commit & push' so that your version on GitHub is always updated. Give an appropriate message each time.
Continue to work on unfinished exercises at home: regularly perform a 'push & commit' command so that all your solutions are available in your online GitHub repository.
When mastering a programming language such as Python, frequent practice is essential and a prerequisite. For this reason, you will find two additional sections in each lab document. These are labelled as follows

Further reading – Doing your own research

This section goes beyond what you have seen this week. Often the exercises are just a little more difficult than what you did in the lab. You will need the Python documentation and Google for this investigation.
In this section you will research a topic that will be covered in theory or lab in the following weeks. These exercises you should also make.

Homework

In this section you will find exercises similar to those you have already done in the laboratory. These exercises have the same level of difficulty as the ones you did in the lab. It is only by doing the exercises "on your own" at home that you will consolidate the material.
Stuck on an exercise? Check the theory and see if you can find a similar exercise you did in the lab. Still not succeeding? Bring your preparation to the Study Morning!

Exercises

In your comments, always mention the number of the exercise!

● B ● ● Exercise 01:

We are going to complete Exercise 6 from Lab Week 6. In the start files you will find the start version of the ShoppingCart class.

- Study the source code. *What are the attributes? What are the properties?*
- Run some test code.

Add the following functionality to the class:

- Add the toString-method: __str__(). The output must be:
  `"There are {<??? Number of products ???>} products in the shopping cart: {??? List of products ???}"`

- Make sure that the + operator can be used:
  - so that two shopping carts can be added together to make a new shopping cart.
  - so that products from another shopping cart can be added into the existing shopping cart.

Test your code.

Example of ouput:

```
Terminal:
Shopping cart 1: There are 4 products in the shopping cart: ['cd1', 'cd2', 'cd3', 'cd4']

Shopping cart 2: There are 2 products in the shopping cart: ['Billy', 'Factum']

***** Shopping cart 3 = Shopping cart 1 + Shopping cart 2 *****
Shopping cart 3: There are 6 products in the shopping cart: ['cd1', 'cd2', 'cd3', 'cd4', 'Billy', 'Factum']

***** Shopping cart 1 +=Shopping cart 2 *****
Shopping cart 1: There are 6 products in the shopping cart: ['cd1', 'cd2', 'cd3', 'cd4', 'Billy', 'Factum']
```

Exercise 02

Start with the start file **Player.py**. Go through the code: check which properties/data attributes are present.

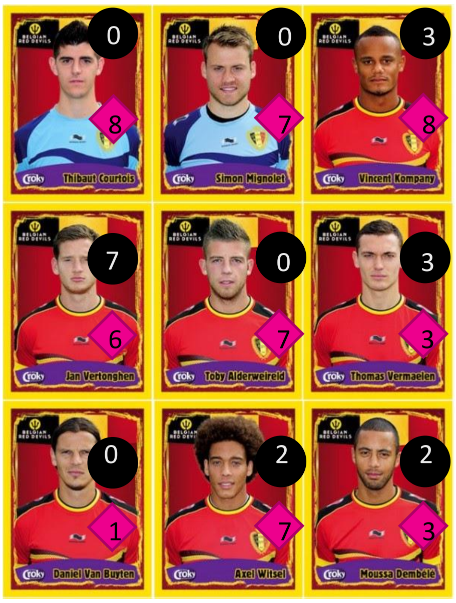
Expand the class with:

- the method "**make_goal**" to increase the number of the individual goals of a player
- a method "……?" so that a player is printed correctly in a list.

Test your code!

Expand the class with:

- the public class-attribute **team_name** is shared by the whole team
- the private class-attribute **number_goals_team**: represent the total number of goals of the whole team. This is the sum of all the individual goals.
  So every time a player scores a new goal, the team's goal total is increased.
- the public static-method **get_goals_team** to return this total number.



● Goals scored by a single player

◆ Value of a player (score on 10)

Team = Red Devils
Total number of goals = 18

Test code

```python
from model.Player import Player

# Test of class attribute & class method (=static method)

def test_players():
    Player.team_name = "Red devils"

    player1 = Player("Thibault", "Courtois", "keeper", 8)
    player2 = Player("Vincent", "Kompany", "striker", 8)
    player3 = Player("Axel", "Witsel", "striker")
    team = [player1, player2, player3]
    print(team)

    print("\nVincent scores!")
    player2.make_goal()
    print(player2)

    print("\nAxel scores!")
    player3.make_goal()
    print(player3)

    print(f"The total number of goals from the team { Player.team_name} is {
          Player.get_goals_team()}")

test_players()
```

Terminal

```
[Player: Courtois, Thibault, Player: Kompany, Vincent, Player: Witsel, Axel]

Vincent scores!
Player: Kompany, Vincent (value: 8/10) goals: 1

Axel scores!
Player: Witsel, Axel (value: 0/10) goals: 1
The total number of goals from the team Red devils is 2
```

Exercise 03

Step 1: create a dataclass **Birthdate** with the day, month and year as attributes.

- Create the required setter and getter properties.
    - Verify the new value in each setter-property.
    - What is a correct value for day/month?
- Program the constructor: __init__()
    - The properties are set using the parameters of the constructor.
- Program the method __str__()
    - The output should be "*day/month/year*".

Test your class!

Step 2:

- Integrate the **equals**-method (__eq__) which is used to check the equality between two birthdates. *When are two birthdates equal?*
- What do you notice when you print a list of birthdates? The __str__() method does not work in this situation? What method should you provide for this?

Test this by creating several objects of this class and comparing them.

Exercise 04

We integrate previous two exercises. We want to store the birthdate (object of the class Birthdate) of each player. This technique is known as **class association**.

- Create a property *birthdate* in the class Player: the datatype is not string but ‘Birthdate’!
- Extend the constructor with an extra (optional) parameter. Give this parameter a default value, i.e. 1 Jan 2024

Now print out each created player's date of birth as well.

Test code

```python
def test_ex4():
    player1 = Player("Thibault", "Courtous", "keeper", 8, Birthdate(11, 5, 1992))
    player2 = Player("Vincent", "Kompany", "striker", 8, Birthdate(10, 4, 1986))
    player3 = Player("Axel", "Witsel", "striker")   # no birthdate --> defaultvalue!

    print("\nThe birthdates of the players are:")
    for item in [player1, player2, player3]:
        print(f"{item} -> Birthdate: {item.birthdate}")

test_ex4()
```

Terminal

```
The birthdates of the players are:
Player: Cortous, Thibault (value: 8/10) goals: 0 -> Birthdate: 11/5/1992
Player: Kompany, Vincent (value: 8/10) goals: 0 -> Birthdate: 10/4/1986
Player: Witsel, Axel (value: 0/10) goals: 0 -> Birthdate: 1/1/2024
```

Exercise 05a

For this exercise, we start with the dataclass **Hotelguest** from previous lab (the class is also present in the source files from week08). Go through the code: check which properties/data attributes are present.

Expand the class with:

- The correct method so that hotel guests are printed correctly in a list.
- The correct method so that we can check when two hotel guests are equal. The equality is checked by last name and first name.
- The correct method so that we can check if a hotel guest is already in a list of hotel guests.

Test your code: **test_hotelguest.py**

Exercise 05b

Now create a new dataclass **Hotel** that contains the name of the hotel and the guest list.

- Create the required setter and getter property for the name.
    - check that the name is not an empty string.
- For the guests: provide only a get property.
- Program the constructor: __init__()
    - One parameter name
    - For the guests: create an empty list in the init method.
- Program the method __str__(): the output will be "*Hotel: name*"

Add the method **check_in**(lastname, firstname): this method takes as parameters the lastname and firstname of the new guest.

- first create a new object of Hotelguest.
- Check first if this new hotelguest does not already exist in the list.
- If not:
    - Add the new hotel guest to the guests list of the hotel.
    - Change the 'is_checked_in' property of the new hotelguest to True.
    - Print the message in the method: "Checked in correctly: <*hotel guest*>"
- If the hotel guest was already present, print an error message.

Add a method function **print_info_guests**() to the Hotel class. For the output: see the test code below.

Test your code!

Test your code. You can use the given testfile.

Test: test_hotel1.py

```
from model.Hotel import Hotel
from model.Hotelguest import Hotelguest

hotel_howest = Hotel("Howest")
print(hotel_howest)
hotel_howest.check_in("Claerhout", "Joerie")
hotel_howest.check_in("Dewitte", "Marie")
hotel_howest.print_info_guests()
```

```
Terminal
Hotel Howest
Checked in correctly:
Hotelguest: Claerhout Joerie - Checked in: True (balance 0 euro)
Checked in correctly:
Hotelguest: Dewitte Marie - Checked in: True (balance 0 euro)
************************************************************
These are the guests present at the hotel Howest:
Hotelguest: Claerhout Joerie - Checked in: True (balance 0 euro)
Hotelguest: Dewitte Marie - Checked in: True (balance 0 euro)
************************************************************
```

●●©● Exercise 05c

Add the private method **__search_hotelguest(lastname,firstname)** to the Hotel class.

- This method takes the last name and first name as parameters.
- Loop over the list of guests and search for the corresponding object hotelguest.
    - If found: return the object
    - If not found: print an error message and return None

Add the method **check_out**(lastname, firstname) to the Hotel class:

- This method takes the guest's name and first name as parameters.
- Use the private method __search_hotelguest() to find the matching guest:
    - If the guest is found: check that their balance is zero. Only then can the guest be removed from the list.
    - If the balance is still not zero: print an error message.
    - If the balance is zero:
        - Change the 'is_checked_in' property of the new hotelguest to False.
        - Remove the hotel guest from the list
    - If the hotel guest is not found, print also an error message.

Test your code!

Test: test_hotel1.py

```
#Exercise 05c
hotel_howest.check_out("Dewitte","Marie")
hotel_howest.check_out("De Gelas", "Johan")
hotel_howest.print_info_guests()
```

```
Terminal
Checks out correctly:
X: Marie DEWITTE
Guest Johan De Gelas has not been found!
************************************************************
These are the guests present at the hotel Howest:
Hotelguest: Claerhout Joerie - Checked in: True (balance 0 euro)
************************************************************
```

Exercise 05d

A hotel guest wishes to pay his bill. Add for this the method **balance_paid_by_guest (name,firstname)** to the class:

- This method takes the guest's last and first name as parameters.
- Use the private method __search_hotelguest() to find the matching guest:
    - If found: set the balance to zero.
    - If not found: print an error message.

Test: test_hotel1.py

```
#Exercise 05d
hotel_howest.guests[0].balance = 75.5             #Joerie drunk a lot last night
hotel_howest.check_out("Claerhout", "Joerie")    #first try

print("\nJoerie pays its debt.")
hotel_howest.balance_paid_by_guest("Claerhout", "Joerie")
hotel_howest.check_out("Claerhout", "Joerie")    #second try
```

```
Terminal
The guest still has an unpaid bill and therefore cannot be checked out yet:
Hotelguest: Claerhout Joerie - Checked in: True (balance 75.5 euro)

Joerie pays its debt.
Checks out correctly:
X: Joerie CLAERHOUT
```

Test your code!

Exercise 05e

Our Hotel class is now almost ready. It would be convenient to store the data of our current guests in a file. For this, we will use hotel_howest.txt (in the doc folder).

Add a new file, **HotelRepository**.py, in the model folder. In this file, define the HotelRepository class. In this class we will add the **static** method **read_hotel_guests**. This method is responsible for reading all hotel guests:

- What parameter(s) does this method take?
- What is the datatype of the return value?

Test the method by printing out the objects read from the file.

After testing:
In the Hotel class, add an extra method called **add_guest**. The parameter for this method is an object of the HotelGuest class. After verifying that the object is not already present, it is added to the guests list.

Use this method to add the objects (read from the file) to your hotel.

Test: test_hotel2.py

```
from model.Hotel import Hotel
from model.Hotelgast import Hotelgast
from model.HotelRepository import HotelRepository

hotel_howest = Hotel("Howest")
print(hotel_howest)

guests = HotelRepository.read_guests("doc/hotel_howest.txt")
for guest in guests:
    hotel_howest.add_guest(guest)

hotel_howest.print_info_guests()
```

```
Terminal
Hotel Howest
******************************************************
These are the guests present at the hotel Howest:
Hotelguest: Walcarius Stijn - Checked in: True (balance 100.0 euro)
Hotelguest: Dewitte Marie - Checked in: True (balance 25.5 euro)
Hotelguest: Berthier Frederiek - Checked in: True (balance 360.0 euro)
Hotelguest: Claerhout Joerie - Checked in: True (balance 2.0 euro)
******************************************************
```

Homework

●●◉● Homework 01

Step 1: create a dataclass **Presenter** with the last name and first name as attributes.

- Create the required setter and getter properties.
    - Verify the new value in each setter-property.
- Program the constructor: __init__()
    - The properties are set using the parameters of the constructor.
- Program the method __str__()
    - The output should be "Presenter: *<last name>, <first name>*".
- Add the equals method
    - Two objects are the same if last name and first name are the same.

Test your class!

Step 2: create a dataclass **Tvprogramme** with following attributes:

- title
- presenter (object of the class Presenter!) => class associaton!
- is_active (Boolean): whether or not it will be broadcast now

Now:

- Create the required properties.
    - The **title** has only a getter property.
    - The **presenter** has a getter and setter property
        - Verify the datatype of the new value in the setter: only objects from the class Presenter are allowed. If a different value is passed, store 'None'
    - is_active has a setter and getter property.
        - The setter property only accepts a value True or False.
        - If a different value is passed, use False.
- Program the constructor: __init__()
    - The parameters are title and presenter
    - The properties are set using the parameters of the constructor.
    - Set the attribute is_active to the value True
- Program the methods __str__() and __repr__()
    - In both cases the output should be "Tvprogramme *<title>* by *<presenter>*".

Test your class!

Step 3: create a dataclass **Tvchannel** with following attributes:

- Name
- Language
- A list of tv-programmes ('programmes')

Now:

- Create the required properties.
    - The **name** has a getter and setter property.
    - The **programmes** has only a getter property.
    - The **language** has a getter and setter property.
        - Only the values 'NL', 'ENG' or 'FR' are allowed.

        - If another value is passed, use the string 'ERROR'
- Program the constructor: __init__()
    - The parameters are name and language
    - The properties are set using the parameters of the constructor.
    - Set the attribute programmes as an empty list.
- Program the methods __str__()
    - The output should be " *<name>* -> *<list of programmes>*"

This class contains an additional method **add_tvprogramme** (Tvprogramme)

- An instance of the class TVprogramme is passed to this method as a parameter.
- After checking that the parameter is indeed a Tvprogramme, adds it to the list of programmes.

Extra:

This class has an additional method **select_random_programme()**.

- This method does not take any parameters.
- This method returns a random Tvprogramme from the list of programmes.

This class contains an additional method **search_inactive_programmes()**.

- This method does not take any parameters.
- This method returns a list of all inactive programmes.

Test your class! You can find some testcode on Leho.

Good luck!