Object Oriented Programming, Exercise 4

Topics: Methods, encapsulation, objects as arguments and attributes, scope of methods

use Git. Make a Git commit at least after every coding task.

Have each coding task in a separate file, so that you can run the code when asked

If you are unsure what to do try first with pseudocode, you get points also if you have written the program with pseudocode!

Code in Python3 and follow the style guide.

Deadline: 27 February 2025, at 20:00

All the code base needed for these exercises are available in https://github.com/BorreBuu/OOP_Python

---

1. **Explain**
    a. encapsulation
    b. client
    c. data attributes
    d. instance

2. **Statistics on numbers**

    part 1: count the numbers

    At this point there is no need to store the numbers themselves in any data structure. It is enough to simply remember how many have been added. The add_number method does take an argument, but there is no need to process the actual value in any way just yet.

    Use the "skeleton" named exercise1_2.py in GitHub for starters.

    After this part the program should work in the following way:

```
stats = NumberStats()
stats.add_number(3)
stats.add_number(5)
stats.add_number(1)
stats.add_number(2)
print("Numbers added:", stats.count_numbers())
```

```
Numbers added: 4
```

    Part 2: the sum and the mean

    Please add the following methods to your class definition:

    the method `get_sum` should return the sum of the numbers added (if no numbers have been added, the method should return 0)

the method `average` should return the mean of the numbers added (if no numbers have been added, the method should return 0)

```
stats = NumberStats()
stats.add_number(3)
stats.add_number(5)
stats.add_number(1)
stats.add_number(2)
print("Numbers added:", stats.count_numbers())
print("Sum of numbers:", stats.get_sum())
print("Mean of numbers:", stats.average())
```

```
Numbers added: 4
Sum of numbers: 11
Mean of numbers: 2.75
```

Part 3: user input

Please write a main program which keeps asking the user for integer numbers until the user types in -1. The program should then print out the sum and the mean of the numbers typed in. You can assume that the user is entering integers.

Your program should use a NumberStats object to keep a record of the numbers added.

NB: you do not need to change the NumberStats class in this part of the exercise. Use an instance of the class to complete this part. (This was talked about in Lecture 2, you can also use this as reference: https://www.geeksforgeeks.org/creating-instance-objects-in-python/)

After this part, the main function should work in the following way:

```
Please type in integer numbers:
4
2
5
2
-1
Sum of numbers: 13
Mean of numbers: 3.25
```

Part 4: Multiple sums

Please add to your main program so that it also counts separately the sum of the even and the odd numbers added.

NB: do not change your NumberStats class definition in this part of exercise, either. Instead, **define three NumberStats objects.** One of them should keep track of all the numbers, another should track the even numbers, and the third should track the odd numbers typed in.

The main function should work in the following way:

```
Please type in integer numbers:
4
2
5
2
-1
Sum of numbers: 13
Mean of numbers: 3.25
Sum of even numbers: 8
Sum of odd numbers: 5
```

---

3. Fastest Car

The exercise template contains a class named Car which represents the features of a car through two attributes: make (str) and top_speed (int).

Please write a function named fastest_car(cars: list) which takes a list of Car objects as its argument.

The function should return the make of the fastest car. You may assume there will always be a single car with the highest top speed. Do not change the list given as an argument or make any changes to the Car class definition.

You can use the code that is available on GitHub to test your program

4. LunchCard

At Linus, the student cafeteria at the TUAS, students can pay for their lunch with a special debit card.

In this exercise you will write a class called LunchCard, with the purpose of emulating the functions provided by the cafeteria's debit card.

Please create a new class named LunchCard.

Part 1 the structure of the new class

First write the constructor for the class. It should take the initial balance available on the card as an argument and save it as an attribute. This is provided for you in the skeleton below.

Next, write a __str__ method, which returns a string containing the balance: "The balance is X euros". The available balance should be printed out with one decimal place precision. Please see the example below for usage.

Here is a skeleton implementation for the class:

```
class LunchCard:
    def __init__(self, balance: float):
        self.balance = balance

    def __str__(self):
        pass
```

A usage example:

```
card = LunchCard(50)
print(card)
```

Prints out the following:

```
The balance is 50.0 euros
```

2. Paying for the lunch

Please implement the following methods in your LunchCard class:

`eat_ordinary` subtracts 2.95 euros from the balance on the card.

`eat_luxury` subtracts 5.90 euros from the balance on the card.

You can use the following main function to test your class:

```
card = LunchCard(50)
print(card)

card.eat_ordinary()
print(card)

card.eat_luxury()
card.eat_ordinary()
print(card)
```

The program should print the following (if I calculated this correctly):

The balance is 50.0 euros

The balance is 47.05 euros

The balance is 38.2 euros

Make sure the balance is never allowed to reach numbers below zero. If the balance goes below zero the lunch is not sold and the price is not subtracted from balance, if you want you can add a notification to the user, but this is not obligatory.

Part 3 depositing money on the card

Implement the `deposit_money` method in your `LunchCard` class.

The method increases the balance on the card by the amount given as an argument.

```
card = LunchCard(10)
print(card)
card.deposit_money(15)
print(card)
card.deposit_money(10)
print(card)
card.deposit_money(200)
print(card)
```

```
The balance is 10.0 euros
The balance is 25.0 euros
The balance is 35.0 euros
The balance is 235.0 euros
```

This method should raise an exception of type ValueError if the user tries to deposit a value under zero.

Here are instructions how to raise exceptions, if you do not remember: https://programming-24.mooc.fi/part-6/3-errors#raising-exceptions

```
card = LunchCard(10)
card.deposit_money(-10)
```

```
File "testi.py", line 3, in lunchcard
ValueError: You cannot deposit an amount of money less than zero
```

Part 4 Multiple cards

Please write a main function which contains the following sequence of events:

Create a lunch card for Peter. The initial balance on the card is 20 euros.

Create a lunch card for Grace. The initial balance on the card is 30 euros.

Peter eats a luxury lunch.

Grace eats an ordinary lunch.

Print out the balance on each card (on separate lines, with the name of the owner at the beginning of the line)

Peter deposits 20 euros

Grace eats the special

Print out the balance on each card (on separate lines, with the name of the owner at the beginning of the line)

Peter eats an ordinary lunch.

Peter eats an ordinary lunch.

Grace deposits 50 euros

Print out the balance on each card (on separate lines, with the name of the owner at the beginning of the line)

The program should print the balance in a following way:

Peter: The balance is 20 euros

Grace: The balance is 30 euros

5. **LunchCard and PaymentTerminal (continue developing the program that you did above)**

After using the Paying for the lunch program, we notice there are some problems with it. The card itself had the knowledge of the prices of the different lunch options and knew to subtract the right amount of money from the balance based on these.

However, if we would like to change the prices or add some new items, we would need to replace all the existing cards with newer versions with the knowledge of new prices and/or items.

This is waste of good cards and money. We obviously need a better version!

Here we make a "stupid" card that only keeps track of the available balance. All more complicated features are going to be contained within another class: the payment terminal.

Part 1: A simpler LunchCard

You can find the template from GitHub lunch_card_and_payment_terminal.py file.

Here the card contains only functionality for finding out the current balance, depositing money on the card and substracting from the balance. Please write the `substract_from_balance(amount)` method.

After this part the program should print the following (the sums are not correct!):

```
Balance 10
Payment successful: True
Balance 2
Payment successful: False
Balance 2
```

Part 2: The payment terminal and dealing with cash payments

In the student cafeteria it is possible to pay with either cash or a LunchCard. A payment terminal is used to handle both cash and card transactions. Let's start with the cash transactions.

You can find the skeleton implementation for PaymentTerminal class from the lunch_card_and_payment_terminal.py file. Please fill in the methods as described in the comments.

If you use the testing code for the Part 2, you could get something like this: (note the prices are wrong!)

```
The change returned was 7.5
The change returned was 2.5
The change returned was 0.0
Funds available at the terminal: 1009.3
Regular lunches sold: 2
Special lunches sold: 1
```

Part 3: Dealing with the card transactions

After we can handle the cash payments, we continue with card transactions. **We will need methods that take a LunchCard as an argument.** The methods reduce the price of the chosen lunch from the card. You can see the outlines of the methods in the lunch_card_and_payment_terminal.py file. Please fill in the methods as described in the comments.

Lunches are sold if the card has enough money. Remember to also increase the number of lunches sold appropriately

The program should print something along the example (the prices are not correct!)

```
The change returned was 7.5
Payment successful: True
Payment successful: False
Payment successful: True
Funds available at the terminal: 1002.5
Regular lunches sold: 2
Special lunches sold: 1
```

Part 4: Depositing money on the card

Lastly, we add a method that lets us deposit money on the card. The card owner pays cash, so the deposited sum is added to the fund available at the terminal.

The program should print something along these lines (the money in terminal may not be the same):

```
Card balance is 2 euros
Payment successful: False
Card balance is 102 euros
Payment successful: True
Card balance is 97.7 euros
Funds available at the terminal: 1100
Regular lunches sold: 0
Special lunches sold: 1
```

6. Box of presents

In this exercise you will practice wrapping presents. You will write two classes: Present and Box. A present has a name and a weight, and a box contains presents. The producer can be a brand name, author, performer, etc.

Part 1: The Present class

Please define the class Present which can be used to represent different kinds of presents. The class definition should contain attributes for the name and the weight (in g) of the present. Instances of the class should work as follows:

```
book = Present("Ta-Nehisi Coates: The Water Dancer", 200)
print("The name of the present:", book.name)
print("The weight of the present:", book.weight)
print("Present:", book)
```

This should print out:

The producer of the present:

The name of the present: Ta-Nehisi Coates: The Water Dancer

The weight of the present: 200

Present: The Water Dancer (200 g)

Part 2: The Box class

Please define the class Box. You should be able to add presents to the box, and the box should keep track of the combined weight of the presents within. The class definition should contain these methods:

- `add_present(self, present: Present)` which adds the present given as an argument to the box. The method has no return value.
- `total_weight(self)` which returns the combined weight of the presents in the box.

You may use the following code to test your class:

```
book = Present("Ta-Nehisi Coates: The Water Dancer", 200)

box = Box()
box.add_present(book)
print(box.total_weight())

cd = Present("Pink Floyd: Dark Side of the Moon", 50)
box.add_present(cd)
print(box.total_weight())
```

This should print out the following:

200

250

7. The shortest person in the room

The exercise template contains the class Person. A person has a name and a height. In this exercise you will implement the class Room. You may add any number of persons to a room, and you may also search for and remove the shortest person in the room.

Part 1: Room

Please define the class Room. It should have a list of persons as an attribute, and also contain the following methods:

- `add(person: Person)` adds the person given as an argument to the room.
- `is_empty()` returns True or False depending on whether the room is empty.
- `print_contents()` prints out the contents of the list of persons in the room.

Please have a look at the following usage example:

```
room = Room()
print("Is the room empty?", room.is_empty())
room.add(Person("Lea", 183))
room.add(Person("Kenya", 172))
room.add(Person("Ally", 166))
room.add(Person("Nina", 162))
room.add(Person("Dorothy", 175))
print("Is the room empty?", room.is_empty())
room.print_contents()
```

The program should have the following output:

```
Is the room empty? True
Is the room empty? False
There are 5 persons in the room, and their combined height is 838 cm.
Lea (183 cm)
Kenya (172 cm)
Ally (166 cm)
Nina (162 cm)
Dorothy (175 cm)
```

Part 2: The shortest person

Please define the method `shortest()` within the Room class definition. The method should return the shortest person in the room it is called on. If the room is empty, the method should return None. The method should not remove the person from the room.

At this point, you can test the program by adding the following code to the previous test code:

```
print("Is the room empty?", room.is_empty())
print("Shortest:", room.shortest())
print()
```

```
room.print_contents()
```

After this, you should have the following output:

Is the room empty? False

Shortest: Nina

There are 5 persons in the room, and their combined height is 858 cm

Lea (183 cm)

Kenya (172 cm)

Ally (166 cm)

Nina (162 cm)

Dorothy (175 cm)

Part 3: Removing a person from the room

Please define the method `remove_shortest()` within the Room class definition. The method should remove the shortest Person object from the room and return the reference to the object. If the room is empty, the method should return None.

At this point, you can test the program by adding the following code to the previous test code:

```
removed = room.remove_shortest()
print(f"Removed from room: {removed.name}")

print()

room.print_contents()
```

After this, you should have the following output:

Removed from room: Nina

There are 4 persons in the room, and their combined height is 696 cm

Lea (183 cm)

Kenya (172 cm)

Ally (166 cm)

Dorothy (175 cm)

You may find this helpful: https://programming-24.mooc.fi/part-4/3-lists#removing-items-from-a-list

8. Recording

Please create a class named Recording which models a single recording. The class should have one private variable: `__length` of type integer.

Please implement the following:

- a constructor which takes the length as an argument
- a getter method length which returns the length of the recording
- a setter method which sets the length of the recording

It should be possible to make use of the class as follows:

```
the_wall = Recording(43)
print(the_wall.length)
the_wall.length = 44
print(the_wall.length)
```

The output should then look something like:

```
43
44
```

9. Weather station

Please create a class named `WeatherStation` which is used to store observations about the weather. The class should have the following public attributes:

- a constructor which takes the name of the station as its argument
- a method named `add_observation(observation: str)` which adds an observation as the last entry in a list
- a method named `latest_observation()` which returns the latest observation added to the list. If there are no observations yet, the method should return an empty string.
- a method named `number_of_observations()` which returns the total number of observations added
- a `__str__` method which returns the name of the station and the total number of observations added as per the example below.

All attributes should be encapsulated, so they can't be directly accessed. It is up to you how you implement the class, as long as the public interface is exactly as described above.

An example of how the class is used:

```
station = WeatherStation("Houston")
station.add_observation("Rain 10mm")
station.add_observation("Sunny")
print(station.latest_observation())

station.add_observation("Thunderstorm")
print(station.latest_observation())
```

```
print(station.number_of_observations())
print(station)
```

The output should be something like the following:

Sunny

Thunderstorm

3

Houston, 3 observations

10. Service charge

Please create a class named `BankAccount` which models a bank account. The class should contain a constructor which takes the name of the owner (str), account number (str) and balance (float) as arguments.

- a method `deposit(amount: float)` for depositing money to the account
- a method `withdraw(amount: float)` for withdrawing money from the account
- a getter method balance which returns the balance of the account

The class should also contain the private method `__service_charge()`, which decreases the balance on the account by one percent. Whenever either of the methods deposit or withdraw is called, this method should also be called. The service charge is calculated and subtracted only after the actual operation is completed (that is, after the amount specified has been added to or subtracted from the balance).

All data attributes within the class definition should be private.

You may use the following code for testing your class:

```
account = BankAccount("Randy Riches", "12345-6789", 1000)
account.withdraw(100)
print(account.balance)
account.deposit(100)
print(account.balance)
```

The output of this program should look something like the following:

891.0

981.09