Python for Engineers 0509-1820, Semester 1 2019-2020

Homework 7

General instructions

- Read the questions **carefully** and make sure your programs work according to the requirements.
- The homework needs to be done individually!
- Read the submission rules on the course web page. All the questions need to be submitted together in the file ex1_012345678.py attached to the homework, after changing the number 012345678 with your ID number (9 digits, including check digit).
- How to write the solution: in this homework, you need to complete the code in the attached outline file.
- Check your code: in order to ensure correctness of your programs and their robustness in the presence of faulty input, for each question run your program with a variety of different inputs, those that are given as examples in the question and additional ones of your choice (check that the output is correct and that the program does not crash).
- Some questions are checked automatically. You thus need to write your solutions only in the specified spaces in the outline file.
- Unless stated otherwise, you can suppose that the input received by the functions is correct.
- You are not allowed to change the names of the objects and variables that already appear in the attached outline file.
- You are not allowed to erase the instructions that appear in the outline file (except the lines that contain the keyword “pass”).
- You are not allowed to use outside libraries (you may not use **import**), unless stated otherwise.
- Final submission date: see course web page.

Question 1

In this question, we will implement the class *Beverage* which represents a beverage that you can order in a restaurant.

Every object of type *Beverage* will contain the following attributes:

- *name*: an attribute of type *string* which represents the name of the beverage
- *price*: an attribute of type *float* which represents the price of the beverage
- *is_diet*: an attribute of type *boolean* which contains *True* if the beverage is a diet beverage and *False* otherwise.

A. Implement the following constructor:

__init__(self, name, price, is_diet)

This constructor receives the required data and saves them in the relevant attributes of the new object that it creates.

- You can assume that the types of the input variables are correct and that the strings are not empty.
- You need to check that the variable *price* is greater than 0, otherwise you need to raise an exception of type *ValueError* with an error message of your choice.

B. Implement the following method in the class *Beverage*:

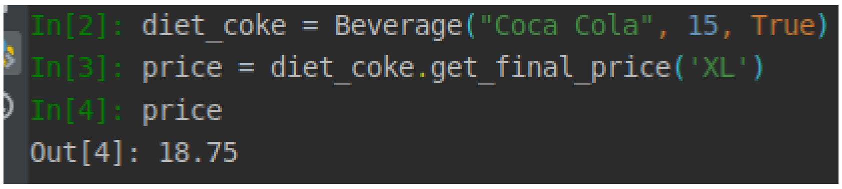
get_final_price(self, size)

This method will return the price of the beverage, according to the required size. The *size* parameter is a string that contains one of the three following values: “Normal”, “Large”, “XL”.

- If the size is “Large”, you need to return the normal price (the one defined in point A above).
- If the size is “XL”, you need to return 125% of the normal price.
- If the size is “Normal”, you need to return 75% of the normal price.
- If the input is different from the strings defined above, you need to raise an exception of type *ValueError* with an error message of your choice.
- We want to be able to also call the method without providing the size of the drink, that it with a default value. A call to the function without the size parameter will behave like a call with the size “Large”. For this purpose, you are allowed to change the definition of the function given above, but not its name.

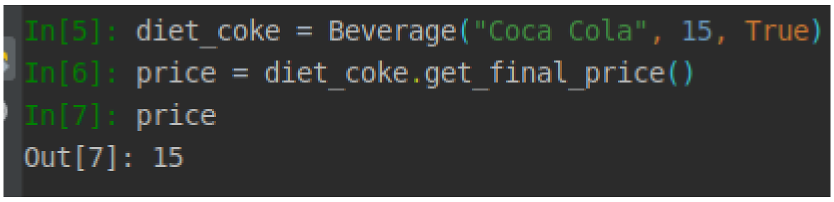
Example of execution:

Normal execution (note that it returns 125% of 15):



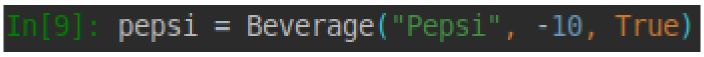
```
In[2]: diet_coke = Beverage("Coca Cola", 15, True)
In[3]: price = diet_coke.get_final_price('XL')
In[4]: price
Out[4]: 18.75
```

Execution without the *size* parameter:

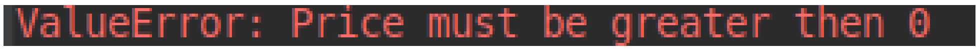


```
In[5]: diet_coke = Beverage("Coca Cola", 15, True)
In[6]: price = diet_coke.get_final_price()
In[7]: price
Out[7]: 15
```

Execution with a negative price:

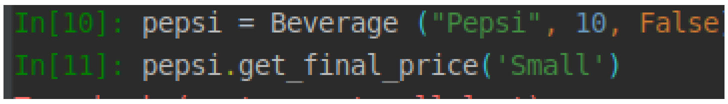


```
In[9]: pepsi = Beverage("Pepsi", -10, True)
```

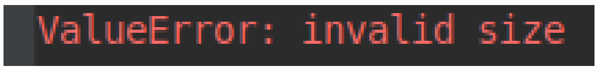


```
ValueError: Price must be greater then 0
```

Execution with a wrong value for *size*:



```
In[10]: pepsi = Beverage ("Pepsi", 10, False
In[11]: pepsi.get_final_price('Small')
```



```
ValueError: invalid size
```

<u>Question 2</u>

In this question we will implement a class *Pizza* which represents a pizza that you can order in a restaurant.

Every object of type *Pizza* will contain the following attributes:

- *name*: an attribute of type *string* which represents the name of the pizza
- *price*: an attribute of type *float* which represents the price of the pizza
- *calories*: an attribute of type *int* which represents the number of calories of the pizza
- *toppings*: an attribute of type *list*, where every item of the list has type *string* and represents one topping of the pizza

A. Implement the following constructor:

__init__(self, name, price, calories, toppings)

This constructor receives the required data and saves them in the relevant attributes of the new object that it creates.

- You can assume that the types of the input variables are correct and that the strings are not empty.
- You need to check that the variables *price* and *calories* are greater than 0, otherwise you need to raise an exception of type *ValueError* with an error message of your choice.

B. Implement the following method in the class *Pizza*:

get_final_price(self, size)

The method will return the price of the pizza depending on the required size. The size parameter is a string that contains one of the three following values: “Personal”, “Family”, “XL”.

- If the size of the pizza is “Family”, you need to return the normal price (the one defined in point A above)
- If the size of the pizza is “XL”, you need to return 125% of the normal price.
- If the size of the pizza is “Personal”, you need to return 60% of the normal price.
- If the input is different from the strings defined above, you need to raise an exception of type *ValueError* with an error message of your choice.
- We want to be able to also call the method without providing the size of the pizza, that it with a default value. A call to the function without the size parameter will behave like a call with the size “Family”. For this purpose, you are allowed to change the definition of the function given above, but not its name.

C. We want to be able to modify the toppings of a pizza.

1. Implement the method:

   add_topping(self, topping, calories, price)

   The method receives the name of the topping (*topping*) as a *string*, the number of calories of the topping (*calories*) as an *int*, and the price of the topping (*price*) as a *float*. The method will add the new topping to the list of existing toppings and will add the price of the new topping to the price of the pizza. It will also add the number of calories of the topping to the total number of calories of the pizza.
   - If the topping already appears among the pizza’s list of toppings, you need to raise an exception of type *ValueError* with an error message of your choice.
   - You can assume that the string *topping* is not empty, that the given price and number of calories are numbers greater than 0.

2. Implement the method:

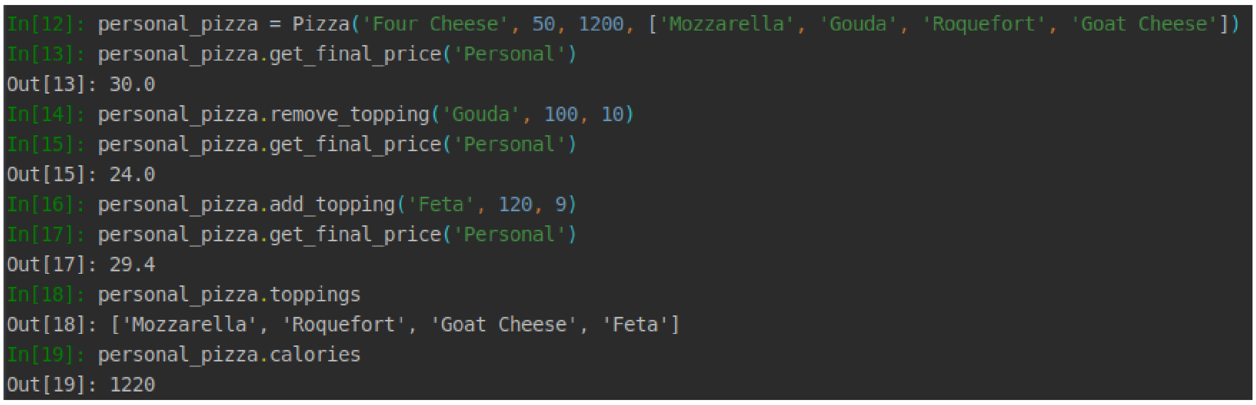
   remove_topping(self, topping, calories, price)

   The method receives the name of the topping (*topping*) as a *string*, the number of calories of the topping (*calories*) as an *int*, and the price of the topping (*price*) as a *float*. The method will remove the given topping from the pizza’s list of toppings and will deduct the price of the topping from the price of the pizza. It will also deduct the topping’s number of calories from the number of calories of the pizza.
   - If the topping does not appear among the pizza’s list of toppings, you need to raise an exception of type *ValueError* with an error message of your choice.
   - You can assume that the string *topping* is not empty, that the given price and number of calories are numbers greater than 0.
   - You also need to raise an exception of type *ValueError* (with an error message of your choice) if the new price or the new number of calories is smaller than 0.

- Note: we assume the toppings are case-sensitive, that is “Tomato” and “tomato” are considered as different toppings.

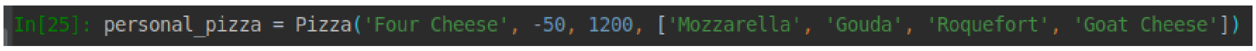
Example of execution:

Normal execution:

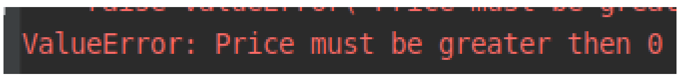


```
In[12]: personal_pizza = Pizza('Four Cheese', 50, 1200, ['Mozzarella', 'Gouda', 'Roquefort', 'Goat Cheese'])
In[13]: personal_pizza.get_final_price('Personal')
Out[13]: 30.0
In[14]: personal_pizza.remove_topping('Gouda', 100, 10)
In[15]: personal_pizza.get_final_price('Personal')
Out[15]: 24.0
In[16]: personal_pizza.add_topping('Feta', 120, 9)
In[17]: personal_pizza.get_final_price('Personal')
Out[17]: 29.4
In[18]: personal_pizza.toppings
Out[18]: ['Mozzarella', 'Roquefort', 'Goat Cheese', 'Feta']
In[19]: personal_pizza.calories
Out[19]: 1220
```

Execution with a negative price:

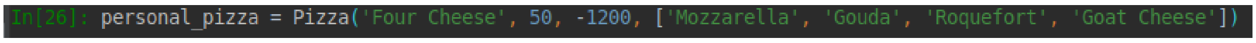


```
In[25]: personal_pizza = Pizza('Four Cheese', -50, 1200, ['Mozzarella', 'Gouda', 'Roquefort', 'Goat Cheese'])
```

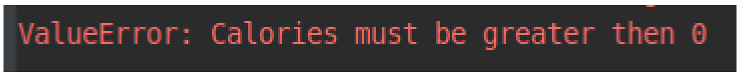


```
ValueError: Price must be greater then 0
```

Execution with a negative number of calories:

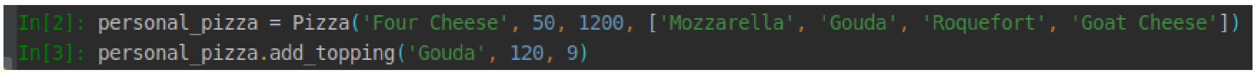


```
In[26]: personal_pizza = Pizza('Four Cheese', 50, -1200, ['Mozzarella', 'Gouda', 'Roquefort', 'Goat Cheese'])
```

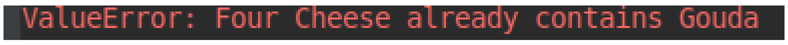


```
ValueError: Calories must be greater then 0
```

Trying to add a topping that already exists:

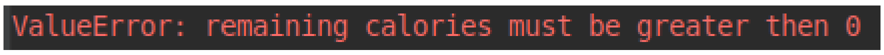


```
In[2]: personal_pizza = Pizza('Four Cheese', 50, 1200, ['Mozzarella', 'Gouda', 'Roquefort', 'Goat Cheese'])
In[3]: personal_pizza.add_topping('Gouda', 120, 9)
```

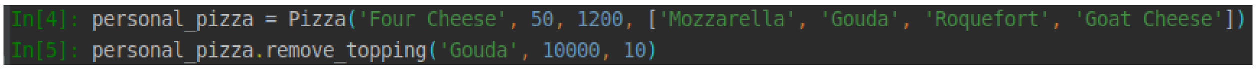


```
ValueError: Four Cheese already contains Gouda
```

Trying to remove a topping with more calories than the total number of calories of the pizza:

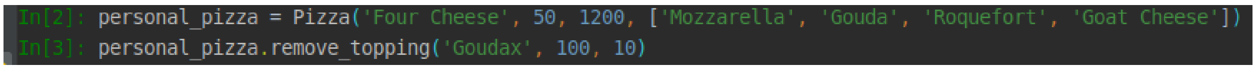


```
ValueError: remaining calories must be greater then 0
```

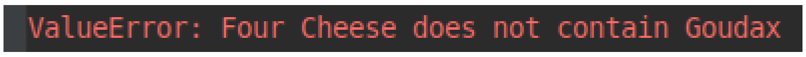


```
In[4]: personal_pizza = Pizza('Four Cheese', 50, 1200, ['Mozzarella', 'Gouda', 'Roquefort', 'Goat Cheese'])
In[5]: personal_pizza.remove_topping('Gouda', 10000, 10)
```

Trying to remove a topping which does not exist in the pizza's list of toppings:



```
In[2]: personal_pizza = Pizza('Four Cheese', 50, 1200, ['Mozzarella', 'Gouda', 'Roquefort', 'Goat Cheese'])
In[3]: personal_pizza.remove_topping('Goudax', 100, 10)
```



```
ValueError: Four Cheese does not contain Goudax
```

Question 3

In this question, we will implement the class *Meal* which represents a meal that you can order at a restaurant.

Every object of type *Meal* will contain the following fields:

1. *beverage*: an attribute of type *Beverage* which represents the meal's beverage
2. *pizza*: an attribute of type *Beverage* which represents the meal's beverage

A. Implement the following constructor:

__init__(self, beverage, pizza)

This constructor receives the required data and saves them in the relevant attributes of the new object that it creates.

- You can assume that the types of the input variables are correct

B. Implement the following method:

get_final_price(self, beverage_size, pizza_size)

The method receives *beverage_size*, which represents the size of the beverage as a string, and *pizza_size*, which represents the size of the pizza as a string. The method returns the sum of the price of the beverage and the price of the pizza.

- Note: you need to use the methods *get_final_price* that we defined for a beverage and a pizza in questions 1 and 2, in order to compute the sum.
- In order not to create confusion among the clients, we will not define default values in this method, as we did in the *get_final_price* methods of questions 1 and 2.

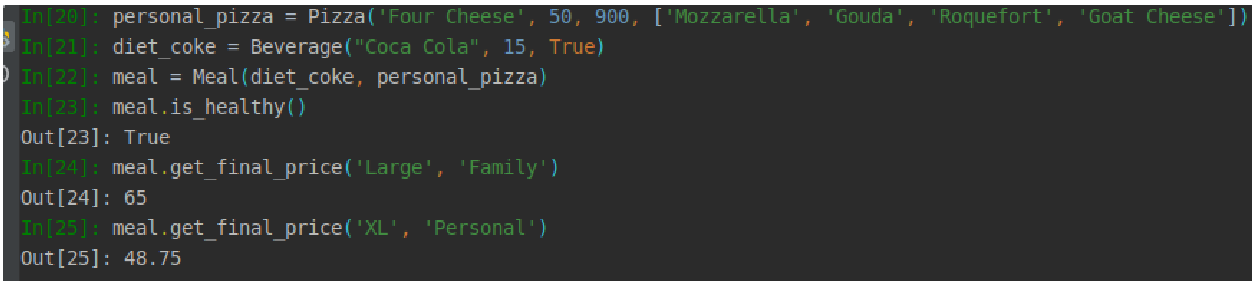
C. Implement the following method:

is_healthy(self)

The method returns a *boolean* value telling if the meal is healthy. We will define a healthy meal as a meal that contains a diet beverage and contains less than 1000 calories in total.

Example of execution:

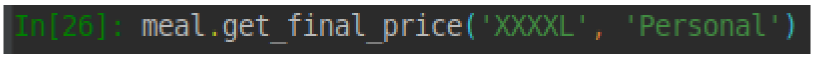
Normal execution:



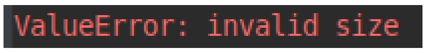
```
In[20]: personal_pizza = Pizza('Four Cheese', 50, 900, ['Mozzarella', 'Gouda', 'Roquefort', 'Goat Cheese'])
In[21]: diet_coke = Beverage("Coca Cola", 15, True)
In[22]: meal = Meal(diet_coke, personal_pizza)
In[23]: meal.is_healthy()
Out[23]: True
In[24]: meal.get_final_price('Large', 'Family')
Out[24]: 65
In[25]: meal.get_final_price('XL', 'Personal')
Out[25]: 48.75
```

In this case, the meal is considered as healthy, because there are less than 1000 calories (900) and the beverage is a diet one.

Execution with a wrong size:



```
In[26]: meal.get_final_price('XXXXL', 'Personal')
```



```
ValueError: invalid size
```

Note that the error here comes from the condition that we imposed in question 1 in the method *Beverage.get_final_price*.