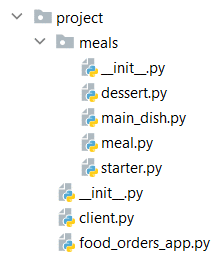
# Python OOP Exam - Food Orders App

*You are given a task to create a basic class application.*

You will be provided with a **skeleton** that includes all the folders and files you will need.

***Note: You are not allowed to change the folder and file structure and change their names!***



# Judge Upload

For the **first 2 problems**, create a **zip** file with the **project** **folder** and **upload it** to the judge system.

For the **last problem**, create a **zip** file with the **test folder** and **upload it** to the judge system.

You do not need to include **in the zip file** your **venv**, **.idea**, **pycache**, and **\_\_MACOSX** (for Mac users), so you do not exceed **the maximum allowed size** of **16.00 KB**.

# Structure (Problem 1) and Functionality (Problem 2)

Our first task is to implement the **structure and functionality** of the classes (properties, methods, inheritance, etc.)

You are **free to add additional attributes** (instance attributes, class attributes, methods, dunder methods, etc.) to simplify your code and increase readability as long as it does not change the project's result according to the requirements and the program works properly.

### Class Client

In the **client.py** file, the class **Client** should be implemented.

### Structure

The class should have the following attributes:

* **phone\_number:** str
  + A string that represents the phone number of the client
  + It must **start with "0" (zero)**, must be **10 characters long**, and must **contain only numbers**.   
    Otherwise, raise a ValueError with the message **"Invalid phone number!"**
* **shopping\_cart: list**
* An **empty** list that will contain **all meals** (objects) **added by the client**
* **bill: float**
* It represents the total amount of money for all meals that the client has added to his shopping cart
* It should be set to **0.0 initially**

### Methods

#### \_\_init\_\_(phone\_number: str)

In the **\_\_init\_\_** method all the needed attributes must be set.

### Class Meal

In the **meal.py** file, the class **Meal** should be implemented. It is a **base class** for any **type of meal,** and it **should not be able to be instantiated**.

### Structure

The class should have the following attributes:

* **name: str**
* A string that represents the name of the meal
* If the name **is an empty string,** raise a **ValueError** with the message **"Name cannot be an empty string!"**
* **price: float**
* A float number that represents the price **per** **one piece** of a meal
* If the price **is less than or equal to 0.0**, raise a **ValueError** with the message **"Invalid price!"**
* **quantity: int**
* An integer that represents the quantity available (to be ordered) for that meal

### Methods

#### \_\_init\_\_(name: str, price: float, quantity: int)

In the **\_\_init\_\_** method all the needed attributes must be set.

#### details()

It returns a string with **information** about the **meal**.

### Class Starter

In the **starter.py** file, the class **Starter** should be implemented. It is a **type of meal**.

If **no quantity** is given, it should be set to **60**.

### Methods

#### \_\_init\_\_(name: str, price: float, quantity: int)

In the **\_\_init\_\_** method all the needed attributes must be set.

#### details()

It should return a string **on one line** in the format shown below. The **price** should be formatted to the **second decimal place**:

**"Starter {name}: {price}lv/piece"**

### Class MainDish

In the **main\_dish.py** file, the class **MainDish** should be implemented. It is a **type of meal**.

If **no quantity** is given, it should be set to **50**.

### Methods

#### \_\_init\_\_(name: str, price: float, quantity: int)

In the **\_\_init\_\_** method all the needed attributes must be set.

#### details()

It should return a string **on one line** in the format shown below. The **price** should be formatted to the **second decimal place**:

**"Main Dish {name}: {price}lv/piece"**

### 5. Class Dessert

In the **dessert.py** file, the class **Dessert** should be implemented. It is a **type of meal**.

If **no quantity** is given, it should be set to **30**.

### Methods

#### \_\_init\_\_(name: str, price: float, quantity: int)

In the **\_\_init\_\_** method all the needed attributes must be set.

#### details()

It should return a string **on one line** in the format shown below. The **price** should be formatted to the **second decimal place**:

**"Dessert {name}: {price}lv/piece"**

### 6. Class FoodOrdersApp

In the **food\_orders\_app.py** file, the class **FoodOrdersApp** should be implemented. It will contain **all the functionality** of the project.

### Structure

The class should have the following attributes:

* **menu: list**
  + An **empty** list that will contain **all the meals** (objects)
* **clients\_list: list**
  + An **empty** list that will contain **all the clients** (objects)

### Methods

#### \_\_init\_\_()

In the **\_\_init\_\_** method all the needed attributes must be set.

#### register\_client(client\_phone\_number: str)

* **Creates a client (object) and adds it to the client list and returns the message "Client {phone\_number} registered successfully."**
* If a client with the **same phone number** is already **registered**, raise an **Exception** with the message **"The client has already been registered!"**

#### add\_meals\_to\_menu(\*meals: Meal)

* This method **adds all the meals** (objects) given **to the menu list**.
* If one or more of the provided objects are **NOT** **meals** (not a **"Starter"**, a **"MainDish"**, or a **"Dessert"**) **ignore them** and **keep adding only the meals**.
* Note: you will always be given meals with different names.

#### show\_menu()

* It should return the **details() for** **each meal** on the menu on **separate lines**
* If there are **less than 5 meals on the menu**, raise an **Exception** with the message **"The menu is not ready!"**

#### add\_meals\_to\_shopping\_cart(client\_phone\_number: str, \*\*meal\_names\_and\_quantities)

***The client attempts to order food.*** *All clients can add* ***any meal that is******on the menu******if there is enough quantity****. You will be given a dictionary with the meal names as keys and the quantity the client wants for each meal as values.*

* Adds the meals (objects) to the **client's shopping cart**.It also **increases the client's bill** with the price for the concrete meal by the quantity added and **decreases the meal quantity on the menu**. Returns the message **"Client {client\_phone\_number} successfully ordered {meal\_names} for {client\_bill}lv."**
  + **The meal names refer to all meal names on the client's shopping cart. They should be separated by a comma and a space ", ".**
  + **The bill is the total amount of money** for all meals on the **client's shopping cart**. It **should be formatted to the second decimal point.**
* **First, the client can only add meals to his/ her cart when the menu is ready. If there are less than 5 meals on the menu,** raise an **Exception** with the message "**The menu is not ready!"**
* If the client with the provided phone number is **not registered in the app**,he/she should be **registered automatically** with the provided phone number, then he/she can **continue with the order**.
* If a **meal** with the given name is **not on the menu**,raise an **Exception** with the message: **"{meal\_name} is not on the menu!"**
  + **If the exception is raised, the client could NOT make the order at all (none of the meals should be added to the client's shopping cart, and the bill should not be increased).**
* **If there isn't enough quantity of a meal to be added to the cart, you must raise an Exception with the following message: "Not enough quantity of {meal\_type}: {meal\_name}!"**
  + **The meal types are "Starter", "MainDish" and "Dessert"**
  + **If the exception is raised, the client could NOT make the order at all (none of the meals should be added to the client's shopping cart, and the bill should not be increased).**

#### cancel\_order(client\_phone\_number: str)

*You will always be given an existing client phone number.*

* The client decides to cancel his/ her order. This method **removes** **all the meals** that the client has added to his/her shopping cart and **resets the bill to 0** and returns the message: "Client {phone\_number} successfully canceled his order.". Keep in mind that you should update the quantity of the meals on the menu list after the cancellation.
* If there are no meals on the client's shopping cart, raise an Exception with the message "There are no ordered meals!"

#### finish\_order(client\_phone\_number: str)

*You will always be given an existing client phone number.*

* The client **pays the bill** and **receives the ordered meals**. This method **removes** **all the meals** that the client has added to his/ her cart and **resets the bill to 0**. Then, it returns the following message: "**Receipt #{receipt\_id} with total amount of {total\_paid\_money} was successfully paid for {client\_phone\_number}.**"
  + The receipt\_id is automatically generated each time a bill is paid (regardless of the client). It starts from 1 and increases by 1 with each new paid bill.
  + The total paid money should be formatted to the second decimal place.
* If there are no meals on the client's shopping cart, raise an Exception with the message "There are no ordered meals!"

#### \_\_str\_\_()

#### This method should return a string with the following information:

"**Food Orders App has {number\_of\_listed\_meals} meals on the menu and {number\_of\_clients} clients.**"

* **Note:** The number of listed meals is the **number of added-to-the-menu meals** (objects), **NOT** their total quantity.

## Examples

|  |
| --- |
| **Input** |
| from project.food\_orders\_app import FoodOrdersApp  from project.meals.starter import Starter  from project.meals.dessert import Dessert  from project.meals.main\_dish import MainDish  food\_orders\_app = FoodOrdersApp()  print(food\_orders\_app.register\_client("0899999999"))  french\_toast = Starter("French toast", 6.50, 5)  hummus\_and\_avocado\_sandwich = Starter("Hummus and Avocado Sandwich", 7.90)  tortilla\_with\_beef\_and\_pork = MainDish("Tortilla with Beef and Pork", 12.50, 12)  risotto\_with\_wild\_mushrooms = MainDish("Risotto with Wild Mushrooms", 15)  chocolate\_cake\_with\_mascarpone = Dessert("Chocolate Cake with Mascarpone", 4.60, 17)  chocolate\_and\_violets = Dessert("Chocolate and Violets", 5.20)  print(food\_orders\_app.add\_meals\_to\_menu(  french\_toast, hummus\_and\_avocado\_sandwich,  tortilla\_with\_beef\_and\_pork,  risotto\_with\_wild\_mushrooms,  chocolate\_cake\_with\_mascarpone,  chocolate\_and\_violets))  print(food\_orders\_app.show\_menu())  food = {"Hummus and Avocado Sandwich": 5,  "Risotto with Wild Mushrooms": 1,  "Chocolate and Violets": 4}  print(food\_orders\_app.add\_meals\_to\_shopping\_cart('0899999999', \*\*food))  additional\_food = {"Risotto with Wild Mushrooms": 2,  "Tortilla with Beef and Pork": 2}  print(food\_orders\_app.add\_meals\_to\_shopping\_cart('0899999999', \*\*additional\_food))  print(food\_orders\_app.finish\_order("0899999999"))  print(food\_orders\_app) |
| **Output** |
| Client 0899999999 registered successfully.  None  Starter French toast: 6.50lv/piece  Starter Hummus and Avocado Sandwich: 7.90lv/piece  Main Dish Tortilla with Beef and Pork: 12.50lv/piece  Main Dish Risotto with Wild Mushrooms: 15.00lv/piece  Dessert Chocolate Cake with Mascarpone: 4.60lv/piece  Dessert Chocolate and Violets: 5.20lv/piece  Client 0899999999 successfully ordered Hummus and Avocado Sandwich, Risotto with Wild Mushrooms, Chocolate and Violets for 75.30lv.  Client 0899999999 successfully ordered Hummus and Avocado Sandwich, Risotto with Wild Mushrooms, Chocolate and Violets, Risotto with Wild Mushrooms, Tortilla with Beef and Pork for 130.30lv.  Receipt #1 with total amount of 130.30 was successfully paid for 0899999999.  Food Orders App has 6 meals on the menu and 1 clients. |

# Task 3: Unit Tests

You will **be provided with another skeleton** for this problem. **Open** the **new skeleton** as a **new project** and write tests for the **ShoppingCart** class. The class will have some methods, fields, and one constructor, all of them working properly. You are **NOT ALLOWED** to change any class. Cover the whole class with unit tests to make sure that the class is working as intended. Submit **only the test** folder.