# Exercise: Classes and Instances

Problems for exercise and homework for the [Python OOP Course @SoftUni](https://softuni.bg/courses/python-oop). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1937>

## Point

Create a class called **Point**. Upon initialization it should receive **x** and **y** (numbers). Create **3 instance methods**:

* **set\_x(new\_x)** - changes the x value of the point
* **set\_y(new\_y)** - changes the y value of the point
* **distance(x, y)** - returns the distance between the point and the provided coordinates

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| p = Point(2, 4)  p.set\_x(3)  p.set\_y(5)  print(p.distance(10, 2)) | 7.615773105863909 |

## Circle

Create a class called **Circle**. Upon initialization it should receive a **radius** (number). Create a class attribute called **pi** which should be equal to **3.14**. Create **3 instance methods**:

* **set\_radius(new\_radius)** - changes the radius
* **get\_area()** - returns the area of the circle
* **get\_circumference()** - returns the circumference of the circle

The area should be rounded to the 2nd decimal.

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| circle = Circle(10)  circle.set\_radius(12)  print(circle.get\_area())  print(circle.get\_circumference()) | 452.16  75.36 |

## Account

Create a class called **Account**. Upon initialization it should receive **id** (number), **name** (string), **balance** (number; **optional**; **0** by default). The class should also have **3 instance methods**:

* **credit(amount)** - add the amount to the balance and **return** the new balance
* **debit(amount)** - if the amount is **less** than or **equal** to the balance, **reduce** the balance by the amount and **return** the new balance. Otherwise return **"Amount exceeded balance"**
* **info()** - returns **"User {name} with account {id} has {balance} balance"**

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| account = Account(1234, "George", 1000)  print(account.credit(500))  print(account.debit(1500))  print(account.info()) | 1500  0  User George with account 1234 has 0 balance |
| account = Account(5411256, "Peter")  print(account.debit(500))  print(account.credit(1000))  print(account.debit(500))  print(account.info()) | Amount exceeded balance  1000  500  User Peter with account 5411256 has 500 balance |

## Employee

Create class **Employee**. Upon initialization it should receive **id** (number), **first\_name** (string), **last\_name** (string), **salary** (number). Create **3 more instance methods**:

* **get\_full\_name()** - returns **"{first\_name} {last\_name}"**
* **get\_annual\_salary()** - returns the salary for **12 months**
* **raise\_salary(amount)** - **increase the salary** by the given amount and **return the new salary**

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| employee = Employee(744423129, "John", "Smith", 1000)  print(employee.get\_full\_name())  print(employee.raise\_salary(500))  print(employee.get\_annual\_salary()) | John Smith  1500  18000 |

## Time

Create a class called **Time**. Upon initialization it should receive **hours**, **minutes** and **seconds** (numbers). The class should also have **class attributes** **max\_hours** equal to **23**, **max\_minutes** equal to **59** and **max\_seconds** equal to **59**. You should also create **3 instance methods**:

* **set\_time(hours, minutes, seconds)** - update the time
* **get\_time()** - returns **"{hh}:{mm}:{ss}"**
* **next\_second()** - update the time with one second (use the **class attributes** for validation) and return the new time (using the **get\_time()** method)

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| time = Time(9, 30, 59)  print(time.next\_second()) | 09:31:00 |
| time = Time(10, 59, 59)  print(time.next\_second()) | 11:00:00 |
| time = Time(23, 59, 59)  print(time.next\_second()) | 00:00:00 |

## Pizza Delivery

Create a class called **PizzaDelivery**. Upon initialization it should receive **name**(string), **price**(float) and **ingredients** (dict). The class should also havean attribute **ordered** set to false by default. You should also create **3 instance methods**:

* **add\_extra(ingredient: str, quantity: int, ingredient\_price: float)**:
  + if we already **have this ingredient** in our pizza **increase the ingredient quantity** with the given one and **update the pizza price** by adding the amount for the additional ingredients
  + if we **don't have this ingredient** in our pizza, we should **add it** and **update the pizza price**
* **remove\_ingredient(ingredient: str, quantity: int, ingredient\_price: float):**
  + if we **don't have this ingredient** in our pizza, we should **return** the following message **"Wrong ingredient selected! We do not use {ingredient} in {name}!"**
  + if we **have the ingredient**, but we try to remove **more than we have available** we should **return** the following message **"Please check again the desired quantity of {ingredient}!"**, otherwise **remove** the given quantity of the ingredient and update the pizza price
* **pizza\_ordered()** – set the attribute **ordered** to **True** and **return** the following message **"You've ordered pizza {name} prepared with {all ingredients and their quantities separated with coma and space} and the price will be {price}lv."**. Please have in mind that once the pizza is ordered no further changes are allowed. We should return the following message if the customer tries to change it: **"Pizza {name} already prepared and we can't make any changes!"**

|  |
| --- |
| **Test Code** |
| Margarita = PizzaDelivery('Margarita', 11, {'cheese': 2, 'tomatoes': 1})  Margarita.add\_extra('mozzarella', 1, 0.5)  Margarita.add\_extra('cheese', 1, 1)  Margarita.remove\_ingredient('cheese', 1, 1)  print(Margarita.remove\_ingredient('bacon', 1, 2.5))  print(Margarita.remove\_ingredient('tomatoes', 2, 0.5))  Margarita.remove\_ingredient('cheese', 2, 1)  print(Margarita.pizza\_ordered())  print(Margarita.add\_extra('cheese', 1, 1)) |
| **Output** |
| Wrong ingredient selected! We do not use bacon in Margarita!  Please check again the desired quantity of tomatoes!  You've ordered pizza Margarita prepared with cheese: 0, tomatoes: 1, mozzarella: 1 and the price will be 9.5lv.  Pizza Margarita already prepared and we can't make any changes! |

## Library

Create class called **Library,** where the information regarding the users and books rented/available will be stored. Create another one called **User,** where the information for each of the library users will be stored: user id, username and **file** with **records of the books rented by the current user**.

### Class Library

Upon initialization you won't receive anything. The class should havethe following attributes:

* **user\_records** – empty list which will store the users (users objects) of the library
* **books\_available** – empty dictionary which will keep information regarding the authors (keys) and the books available for each of the authors (list)
* **rented\_books** – empty dictionary with usernames for keys and another dictionary as value in which the book names will be the keys and days to return the value (**{usernames: {book names: days left}})**.

You should also create **3 instance methods**:

* **add\_user(user: User)**:
  + add the user if we do not have him/her in the library records already, otherwise return the message **"User with id = {user\_id} already registered in the library!"**
* **remove\_user(user: User):**
  + remove the user from the library records if available, otherwise return the message **"We could not find such user to remove!"**
* **change\_username(user\_id: int, new\_username: str):**
  + change the username with the new provided and return the message **"Username successfully changed to: {new\_username} for userid: {user\_id}"** if there is a record with the same user id in the library and the username is different than the provided one.
  + If the username is the same for this id return the following message **"Please check again the provided username - it should be different than the username used so far!"**.
  + If there is no record for the provided id return **"There is no user with id = {user\_id}!"**

### Class User

Upon initialization it should receive **user\_id**(int) and **username**(string). The class should also havean attribute **books** which will be an empty list at the beginning. You should also create **3 instance methods**:

* **get\_book(author: str, book\_name: str, days\_to\_return: int, library: Library)**:
  + if the **book is available** in the library add it to the **books list** for this user, **update the library records (rented\_books and available\_books dicts)** and return the following message: **"{book\_name} successfully rented for the next {days\_to\_return} days!"**
  + if it's **already rented** return the following message **"The book "{book\_name}" is already rented and will be available in {days\_to\_return provided by the user rented the book} days!"**
* **return\_book(author:str, book\_name:str, library: Library):**
  + if the **book is in the user's books list return it in the library** (update **books\_available and rented\_books** class attributes) and **remove it from the books list** for this user
  + otherwise **return** the following message **"{username} doesn't have this book in his/her records!"**
* **info()** – **return** a string containing the books currently rented by the user in ascending order separated by comma and space.
* You should also override the **\_\_string\_\_** method in order to get a string in the following format **"{user\_id}, {username}, {books}"**

***Note:*** Please submit a zip file, containing a separate file for each of the classes, with the class names provided in the problem description and include them in project module in order to be able to make proper imports.

|  |
| --- |
| **Test Code** |
| from project.library import Library from project.user import User  user = User(12, 'Peter')  library = Library()  library.add\_user(user)  print(library.add\_user(user))  library.remove\_user(user)  print(library.remove\_user(user))  library.add\_user(user)  print(library.change\_username(2, 'Igor'))  print(library.change\_username(12, 'Peter'))  print(library.change\_username(12, 'George'))  [print(f'{user\_record.user\_id}, {user\_record.username}, {user\_record.books}') for user\_record in library.user\_records]  library.books\_available.update({'J.K.Rowling': ['The Chamber of Secrets',  'The Prisoner of Azkaban',  'The Goblet of Fire',  'The Order of the Phoenix',  'The Half-Blood Prince',  'The Deathly Hallows']})  user.get\_book('J.K.Rowling', 'The Deathly Hallows', 17, library)  print(library.books\_available)  print(library.rented\_books)  print(user.books)  print(user.get\_book('J.K.Rowling', 'The Deathly Hallows', 10, library))  print(user.return\_book('J.K.Rowling', 'The Cursed Child', library))  user.return\_book('J.K.Rowling', 'The Deathly Hallows', library)  print(library.books\_available)  print(library.rented\_books)  print(user.books) |
| **Output** |
| User with id = 12 already registered in the library!  We could not find such user to remove!  There is no user with id = 2!  Please check again the provided username - it should be different than the username used so far!  Username successfully changed to: George for userid: 12  12, George, []  {'J.K.Rowling': ['The Chamber of Secrets', 'The Prisoner of Azkaban', 'The Goblet of Fire', 'The Order of the Phoenix', 'The Half-Blood Prince']}  {'George': {'The Deathly Hallows': 17}}  ['The Deathly Hallows']  The book "The Deathly Hallows" is already rented and will be available in 17 days!  George doesn't have this book in his/her records!  {'J.K.Rowling': ['The Chamber of Secrets', 'The Prisoner of Azkaban', 'The Goblet of Fire', 'The Order of the Phoenix', 'The Half-Blood Prince', 'The Deathly Hallows']}  {'George': {}}  [] |