# Exercise: Classes and Objects

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>.

## Vet

Create a class called **Vet**. Upon initialization it should receive a **name** (string). It should also have an **instance attribute** called **animals** (empty list by default). There should also be **2 class attributes**: **animals** (empty list) which will store the total amount of **animals for all vets**; **space** (5 by default). You should create **3 instance methods**

* **register\_animal(animal\_name)**
  + If there **is space** in the vet clinic add the animal to **both animals' lists** and return a message: **"{name} registered in the clinic"**
  + Otherwise return **"Not enough space"**
* **unregister\_animal(animal\_name)**
  + If the animal is **in the clinic**, **remove** it from **both animals** lists and return **"{animal} unregistered successfully"**
  + Otherwise, return **"{animal} not in the clinic"**
* **info()** – returns: **"{vet\_name1} has {number\_animals} animals. {space\_left\_in\_clinic} space left in clinic"**

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| peter = Vet("Peter")  george = Vet("George")  print(peter.register\_animal("Tom"))  print(george.register\_animal("Cory"))  print(peter.register\_animal("Fishy"))  print(peter.register\_animal("Bobby"))  print(george.register\_animal("Kay"))  print(george.unregister\_animal("Cory"))  print(peter.register\_animal("Silky"))  print(peter.unregister\_animal("Molly"))  print(peter.unregister\_animal("Tom"))  print(peter.info())  print(george.info()) | Tom registered in the clinic  Cory registered in the clinic  Fishy registered in the clinic  Bobby registered in the clinic  Kay registered in the clinic  Cory unregistered successfully  Silky registered in the clinic  Molly not in the clinic  Tom unregistered successfully  Peter has 3 animals. 1 space left in clinic  George has 1 animals. 1 space left in clinic |

## 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 with the new values
* **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 |

## 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 |

## Pizza Delivery

Create a class called **PizzaDelivery**. Upon initialization it should receive **name**(string), **price**(float) and **ingredients** (dict). The class should also havean instance 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 ingredient price
  + if we **do not 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 **do not have this ingredient** in our pizza, we should **return** the following message **"Wrong ingredient selected! We do not use {ingredient} in {pizza\_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 by removing the ingredient price
* **make\_order()** - set the attribute **ordered** to **True** and **return** the following message **"You've ordered pizza {pizza\_name} prepared with {ingredient: quantity** (separated with", ")**} and the price will be {price}lv."**. 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.make\_order())  print(margarita.add\_extra('cheese', 1, 1)) 11+ 0.5 + 1 – 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 10.5lv.  Pizza Margarita already prepared and we can't make any changes! |

**Note: For the next problems you should 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 module named project to be able to make proper imports.**

## Todo List

You are tasked to create **two classes**: **a Task** class and **a Section** class.



The **Task** class should receive a **name** (string), and a **due\_date** (str) upon initialization. The Task also has **two attributes**: **comments** (empty list) and **completed** set to **False** by default.

The **Task** class should also have **five methods**:

* **change\_name(new\_name: str)**
  + Change **the name of the task** and return **the new name**.
  + If the new name **is the same as the current name**, return **"Name cannot be the same."**
* **change\_due\_date(new\_date: str)** 
  + Change **the due date of the task** and return **the new date**.
  + If the new **date is the same as the current date**, return **"Date cannot be the same."**
* **add\_comment(comment: str)**
  + Add a comment to the task.
* **edit\_comment(comment\_number: int, new\_comment: str)**
  + The **comment number** value represents the **index** of the comment we want to edit. You should **change** **the comment** and **return all the comments**, separated **by comma and space (", ")**
  + If the comment number **is out of range**, return **"Cannot find comment."**
* **details()**
  + Return the task's details in this format:

**"Name: {task\_name} - Due Date: {due\_date}"**

The **Section** class should receive a **name** (string) upon initialization. The Task also has **one instance attribute**: **tasks** (empty list)

The Section class should also have **four methods**:

* **add\_task(new\_task: Task)**
  + Add a **new task** to the collection. **Return "Task {task details} is added to the section"**
  + If the task **is in the collection**, return **"Task is already in the section {section\_name}"**
* **complete\_task(task\_name: str)** 
  + Change the task to completed(**True**). Return **"Completed task {task\_name}"**
  + If the task is not found, return **"Could not find task with the name {task\_name}"**
* **clean\_section()**
  + Removes all the **completed tasks** and returns **"Cleared {amount of removed tasks} tasks."**
* **view\_section()**
  + Return information about the section and its tasks in this format:

**"Section {section\_name}:**

**{details of the first task}**

**{details of the second task}**

**...**

**{details of the n task}"**

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| task = Task("Make bed", "27/05/2020")  print(task.change\_name("Go to University"))  print(task.change\_due\_date("28.05.2020"))  task.add\_comment("Don't forget laptop")  print(task.edit\_comment(0, "Don't forget laptop and notebook"))  print(task.details())  section = Section("Daily tasks")  print(section.add\_task(task))  second\_task = Task("Make bed", "27/05/2020")  section.add\_task(second\_task)  print(section.clean\_section())  print(section.view\_section()) | Go to University  28.05.2020  Don't forget laptop and notebook  Name: Go to University - Due Date: 28.05.2020  Task Name: Go to University - Due Date: 28.05.2020 is added to the section  Cleared 0 tasks.  Section Daily tasks:  Name: Go to University - Due Date: 28.05.2020  Name: Make bed - Due Date: 27/05/2020 |

## Guild System

You are tasked to create **two classes**: **a Player** class and **a Guild** class.



The **Player** class should receive a **name** (string), **hp** (int) and **mp** (int) upon initialization. The **Player** also has 2 instance attributes: **skills** (empty dictionary by initialization – will contain the skills of each player and its mana cost) and **guild** set to **"Unaffiliated"** by default.

The Player class should also have **two methods**:

* **add\_skill(skill\_name, mana\_cost)**
  + Add the skill and the corresponding mana cost to the dictionary of skills. Return **"Skill {skill\_name} added to the collection of the player {player\_name}"**
  + If the skill is already in the collection, return **"Skill already added"**
* **player\_info()** 
  + Returns the player's information, including his/her skills, in this format:

**"Name: {player\_name}  
 Guild: {guild\_name}  
 HP: {hp}**

**MP: {mp}**

**==={skill\_name\_1} - {skill\_mana\_cost}**

**==={skill\_name\_2} - {skill\_mana\_cost}**

**...**

**==={skill\_name\_N} - {skill\_mana\_cost}"**

The **Guild** class receive a **name** {string}. The **Player** should also have one instance attribute **players** (empty list by initialization which will contain the players of the guild). The class also has 3 methods:

* **assign\_player(player: Player)**
  + Add the player to the guild. Return **"Welcome player {player\_name} to the guild {guild\_name}"**.Remember to change the player's guild in the player class.
  + If the player is already in the guild, return **"Player {player\_name} is already in the guild."**
  + If the player is in another guild, return **"Player {player\_name} is in another guild."**
* **kick\_player(player\_name: String)**
  + Remove the player from the guild. Return **"Player {player\_name} has been removed from the guild."**. Remember to change the player's guild in the player class to **"Unaffiliated"**.
  + If the is not a player with that name in the guild, return **"Player {player\_name} is not in the guild."**
* **guild\_info()** 
  + Returns the guild's information, including the players in the guild, in this format:

**"Guild: {guild\_name}  
 {player's info}"**

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| player = Player("George", 50, 100)  print(player.add\_skill("Shield Break", 20))  print(player.player\_info())  guild = Guild("UGT")  print(guild.assign\_player(player))  print(guild.guild\_info()) | Skill Shield Break added to the collection of the player George  Name: George  Guild: Unaffiliated  HP: 50  MP: 100  ===Shield Break - 20  Welcome player George to the guild UGT  Guild: UGT  Name: George  Guild: UGT  HP: 50  MP: 100  ===Shield Break - 20 |

## Spoopify

You are tasked to create **three classes**: **a Song** class, **an Album** class, and **a Band** class.



The **Song** class should receive a **name** (string), **length** (float) and **single** (bool) upon initialization. It has **one** method:

* **get\_info()**
  + Returns the information of the song in this format: **"{song\_name} - {song\_length}"**

The **Album** class should receive a **name** (string) and **songs(one, many or none)** **as arguments** upon initialization. It also has an **instance** **attribute published** (**False** by default). It has **four** methods:

* **add\_song(song: Song)**
  + Adds the **song to the album**. Return **"Song {song\_name} has been added to the album {name}."**
  + If the song is **single**, return **"Cannot add {song\_name}. It's a single"**
  + If the album is **published**, return **"Cannot add songs. Album is published."**
  + If the song is **already added**, return **"Song is already in the album."**
* **remove\_song(song\_name: str)**
  + Removes the song with the given name and return **"Removed song {song\_name} from album {album\_name}."**
  + If the song is not in the album, return **"Song is not in the album."**
  + If the album is published, return **"Cannot remove songs. Album is published."**
* **publish()**
  + Publish the album (set to **True**) and return **"Album {name} has been published."**
  + If the album is published, return **"Album {name} is already published."**
* **details()**
  + Returns the information of the album, with the songs in it, in this format:

**"Album {name}**

**== {first\_song\_info}**

**== {second\_song\_info}**

**…**

**== {n\_song\_info}"**

The **Band** class should receive a **name** (string) upon initialization. It also has an **attribute albums** (**empty list**).

The class has **three** methods:

* **add\_album(album: Album)**
  + Adds an **album to the collection** and returns **"Band {band\_name} has added their newest album {album\_name}."**
  + If the album **is already added**, return **"Band {band\_name} already has {album\_name} in their library."**
* **remove\_album(album\_name: str)**
  + Removes the album from the collection and returns **"Album {name} has been removed."**
  + If the album is **published**, return **"Album has been published. It cannot be removed."**
  + If the album is **not in the collection**, return **"Album {name} is not found."**
* **details()**
  + Returns the information of the band, with their albums, in this format:

**"Band {name}**

**{album details}**

**...**

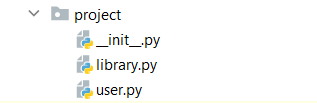
**{album details}"**

### Examples

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| song = Song("Running in the 90s", 3.45, False)  print(song.get\_info())  album = Album("Initial D", song)  second\_song = Song("Around the World", 2.34, False)  print(album.add\_song(second\_song))  print(album.details())  print(album.publish())  band = Band("Manuel")  print(band.add\_album(album))  print(band.remove\_album("Initial D"))  print(band.details()) | Running in the 90s - 3.45  Song Around the World has been added to the album Initial D.  Album Initial D  == Running in the 90s - 3.45  == Around the World - 2.34  Album Initial D has been published.  Band Manuel has added their newest album Initial D.  Album has been published. It cannot be removed.  Band Manuel  Album Initial D  == Running in the 90s - 3.45  == Around the World - 2.34 |

## 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

In the **library.py** create class **Library**. Upon initialization it will not receive anything, but it should havethe following instance 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 (key: str) and the books available for each of the authors (list of strings)
* **rented\_books** - empty dictionary which will keep information regarding the usernames (key: str) and nested dictionary as value in which will keep information regarding the book names (key: str) and days left before returning the book to the library (int) - (**{usernames: {book names: days to return}})**.

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

* **add\_user(user: User)**:
  + Add the user if we do not have him/her in the library's user 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 we have him/her in the library's user records
  + 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. Change his username in the **rented\_books** dictionary as well (if present).
  + If the new 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

In the **user.py** create class **User**. Upon initialization it should receive **user\_id** (int) and **username** (string). The class should also havean instance 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 the book is **not available** in the library, **do nothing**
  + if it is **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 **\_\_str\_\_** method to get a string in the following format **"{user\_id}, {username}, {list of rented books}"**

|  |
| --- |
| **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': {}}  [] |
| **Test Code** |
| from library import Library  from user import User  user = User(12, 'Peter')  library = Library()  library.add\_user(user)  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', 10, library)  print(user) |
| **Output** |
| 12, Peter, ['The Deathly Hallows'] |