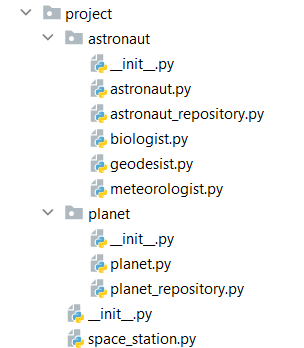
# Python OOP Retake Exam – 23.08.2021



*There is a space station, which navigates astronauts to collect items from foreign planets. The astronauts with different professional speacialties have different survival capabilities in the open space according to their essential needs, like the need for oxygen. Your task is to send them on missions and collect items from the different planets.*

You will be provided with a **skeleton**, which includes all the folders and files that 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 name **project** and upload it to the judge system.

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

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

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

### Astronaut

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

### Structure

The class should have the following attributes:

**from** abc **import** ABC, abstractmethod  
  
  
**class** Astronaut(ABC):  
 **def** \_\_init\_\_(self, name: str, oxygen: int):  
 self.name = name  
 self.oxygen = oxygen  
 self.backpack = []

* **name**: **str**
  + If the name **is an empty string or whitespace,** raise a **ValueError** with message: **"Astronaut name cannot be empty string or whitespace!"**
* **оxygen**: **int**
  + The oxygen of an astronaut in units
* backpack: an empty **list**
  + In the backpack each astronaut will collect items **while on a mission**
* @property  
  **def** name(self):  
   **return** self.\_name  
    
  @name.setter  
  **def** name(self, value):  
   **if** value == **"" or** value.isspace():  
   **raise** ValueError(**"Astronaut name cannot be empty string or whitespace!"**)  
   self.\_name = value

#### Methods

##### \_\_init\_\_(name: str, oxygen: int)

The **\_\_init\_\_** method should have a **name**, a given amount of **oxygen**, and a **backpack**.

##### breathe()

**Each time** an astronaut **takes a breath** their oxygen decreases by **10 units**. Note: some types of astronauts need **more** **oxygen units** while breathing.

@abstractmethod  
**def** breathe(self):  
 self.oxygen -= **10**

##### increase\_oxygen(amount: int)

**Increases** the oxygen with the given **amount**.

**def** increase\_oxygen(self, amount: int):  
 self.oxygen += amount

### Biologist

In the file **biologist.py** the class **Biologist** should be implemented. The biologist is a type of astronaut. Each biologist has **70 initial units of oxygen** and each time they take a breath their oxygen is **decreased** by **5 units**.

**from** project.astronaut.astronaut **import** Astronaut  
  
  
**class** Biologist(Astronaut):  
 **def** \_\_init\_\_(self, name: str):  
 super().\_\_init\_\_(name, **70**)  
  
 **def** breathe(self):  
 self.oxygen -= **5**

### Geodesist

In the file **geodesist.py** the class **Geodesist** should be implemented. The geodesist is a type of astronaut. Each geodesist has **50 initial units of oxygen**.

**from** project.astronaut.astronaut **import** Astronaut  
  
  
**class** Geodesist(Astronaut):  
 **def** \_\_init\_\_(self, name: str):  
 super().\_\_init\_\_(name, **50**)  
  
 **def** breathe(self):  
 self.oxygen -= **10**

### Meteorologist

In the file **meteorologist.py** the class **Meteorologist** should be implemented. The meteorologist is a type of astronaut. Each meteorologist has **90 initial units of oxygen** and each time they take a breath their oxygen is **decreased** by **15 units**.

**from** project.astronaut.astronaut **import** Astronaut  
  
  
**class** Meteorologist(Astronaut):  
 **def** \_\_init\_\_(self, name: str):  
 super().\_\_init\_\_(name, **90**)  
  
 **def** breathe(self):  
 self.oxygen -= **15**

### AstronautRepository

In the **astronaut\_repository.py** file the class **AstronautRepository** should be implemented. It is a repository for the astronauts that are on the Space Station.

**class** AstronautRepository:

#### Structure

The class should have the following attributes:

* astronauts: an empty **list** of **astronauts**
* **def** \_\_init\_\_(self):  
   self.astronauts = []

#### Methods

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

The **\_\_init\_\_** method should have astronauts.

##### add(astronaut: Astronaut)

* Adds an astronaut.
* **def** add(self, astronaut):  
   self.astronauts.append(astronaut)

##### remove(astronaut: Astronaut)

* Removes an astronaut from the collection.
* **def** remove(self, astronaut):  
   **if** astronaut **in** self.astronauts:  
   self.astronauts.remove(astronaut)

##### find\_by\_name(name: str)

* Returns an astronaut with that name if he/ she exists.
* **def** find\_by\_name(self, name: str):  
   find\_astronaut = [a **for** a **in** self.astronauts **if** a.name == name]  
   **if** find\_astronaut:
* **return** find\_astronaut[**0**]

### Planet

In the **planet.py** file the class Planet should be implemented. It is an implementation for a planet, and it holds information about the items that can be found on its surface.

### Structure

The class should have the following attributes:

**class** Planet:  
 **def** \_\_init\_\_(self, name: str):  
 self.name = name  
 self.items = []

* **name**: **str**
  + If the name **is an empty string or whitespace,** raise a **ValueError** with message: **"Planet name cannot be empty string or whitespace!"**
* **items**: an empty **list** of strings holding each item that could be found on that planet
* @property  
  **def** name(self):  
   **return self.\_\_name**@name.setter  
  **def** name(self, value):  
   **if** value == **"" or** value.isspace():  
   **raise** ValueError(**"Planet name cannot be empty string or whitespace!"**)  
   self.\_\_name = value

#### Methods

##### \_\_init\_\_(name: str)

The **\_\_init\_\_()** method should have a **name** and **items**.

### PlanetRepository

In the **planet\_repository.py** file the class **PlanetRepository** should be implemented. It is a repository for planets that await to be explored.

#### Structure

The class should have the following attributes:

* planets: an empty **list** of **planets**
* **from** project.planet.planet **import** Planet  
    
    
  **class** PlanetRepository:  
   **def** \_\_init\_\_(self):  
   self.planets = []

##### Methods

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

The **\_\_init\_\_** method should have an empty list of planets.

##### add(planet: Planet)

* Adds a planet for exploration.
* **def** add(self, planet: Planet):  
   self.planets.append(planet)

##### remove(planet: Planet)

* Removes a planet from the collection.
* **def** remove(self, planet: Planet):  
   **if** planet **in** planets:  
   self.planets.remove(planet)

##### find\_by\_name(name: str)

* Returns a planet with that name if it exists.
* **def** find\_by\_name(self, name: str):  
   find\_planet = [p **for** p **in** self.planets **if** p.name == name]  
   **if** find\_planet:  
   **return** find\_planet[0]

### SpaceStation

In the **space\_station.py** file the class **SpaceStation** should be implemented.

#### Structure

The class should have the following attributes:

* planet\_repository: a new repository created for each space station
* astronaut\_repository: a new repository created for each space station
* **from** project.planet.planet\_repository **import** PlanetRepository
* **from** project.astronaut.biologist **import** Biologist  
  **from** project.astronaut.geodesist **import** Geodesist  
  **from** project.astronaut.meteorologist **import** Meteorologist  
  **from** project.planet.planet **import** Planet  
  **from** project.planet.planet\_repository **import** PlanetRepository
* **class** SpaceStation:  
   **def** \_\_init\_\_(self):  
   self.planet\_repository = PlanetRepository()  
   self.astronaut\_repository = AstronautRepository()  
   self.number\_of\_successful\_missions = **0** self.number\_of\_not\_completed\_missions = **0**

##### Methods

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

The **\_\_init\_\_** method should have a **planet\_repository** and an **astronaut\_repository**.

#### add\_astronaut(astronaut\_type: str, name: str)

* **Creates** an **astronaut** with the given **name** of the given **type, adds** him/ her to the repository and **returns** the following message: **"Successfully added {astronaut\_type}: {astronaut\_name}."**
* If an astronaut with that name is already in the repository returns: **"{astronaut\_name} is already added."**
* The valid astronaut types are **"Biologist"**, **"Geodesist"** and **"Meteorologist"**. If the astronaut type is invalid, raise an **Exeption** with message: **"Astronaut type is not valid!"**
* **def** add\_astronaut(self, astronaut\_type: str, name: str):  
    
   **if** self.astronaut\_repository.find\_by\_name(name):  
   **return f"**{name} **is already added."  
   if** astronaut\_type == **"Biologist"**:  
   self.astronaut\_repository.astronauts.append(Biologist(name))  
   **return f"Successfully added** {astronaut\_type}**:** {name}**."  
   elif** astronaut\_type == **"Geodesist"**:  
   self.astronaut\_repository.astronauts.append(Geodesist(name))  
   **return f"Successfully added** {astronaut\_type}**:** {name}**."  
   elif** astronaut\_type == **"Meteorologist"**:  
   self.astronaut\_repository.astronauts.append(Meteorologist(name))  
   **return f"Successfully added** {astronaut\_type}**:** {name}**."  
   else**:  
   **raise** Exception(**"Astronaut type is not valid!"**)

#### add\_planet(name: str, items: str)

* **Creates** a **planet** with the provided **name** and **items** (**single string** with words, separated by **", "**), adds it to the repository and **returns** the following message: **"Successfully added Planet: {planet\_name}."**
* If a planet with that name is already in the repository returns: **"{planet\_name} is already added."**
* **def** add\_planet(self, name: str, items: str):  
    
   **if** self.planet\_repository.find\_by\_name(name):  
   **return f"**{name} **is already added."** planet = Planet(name)  
   planet.items = items.split(**", "**)  
   self.planet\_repository.planets.append(planet)  
   **return f"Successfully added Planet:** {name}**."**

#### retire\_astronaut(name: str)

* Retires the astronaut from the space station by removing him/her from the repository and returns the following message: **"Astronaut {astronaut\_name} was retired!"**
* If an astronaut with that name doesn't exist, **raise Exception** with the following message: **"Astronaut {astronaut\_name} doesn't exists!"**
* **def** retire\_astronaut(self, name: str):  
   astronaut = self.astronaut\_repository.find\_by\_name(name)  
   **if not** astronaut:  
   **raise** Exception(**f"Astronaut** {name} **doesn't exists!"**)  
    
   self.astronaut\_repository.astronauts.remove(astronaut)  
   **return f"Astronaut** {name} **was retired!"**

#### recharge\_oxygen()

* There is **no capacity** **limit**.
* **def** recharge\_oxygen(self):  
   **for** astronaut **in** self.astronaut\_repository.astronauts:  
   astronaut.increase\_oxygen(**10**)

#### send\_on\_mission(planet\_name: str)

* If the **planet does not exist**, raise an **Exception** with the following message: **"Invalid planet name!"**
* You should start by choosing the astronauts that are most suitable for the mission:
* You should pick **up to 5 astronauts** with the **highest amount of oxygen** among the ones with oxygen **above 30 units**.
* If you **don't have any** **suitable** **astronauts**, raise an **Exception** with the following message: **"You need at least one astronaut to explore the planet!"**
* The astronauts start **going out in open space** one by one, sorted in **descending order by the amount of oxygen they have**:
  + An astronaut lands on a planet and **starts collecting its items one by one starting from the last one in the collection**. Each time he/she **finds an item** he/she **takes a breath**.
  + If an astronaut **run out of** **oxygen,** he/ she should return to the space station and the **next astronaut starts exploring.**
* **A mission is successful when all the items from the planet are collected:**
  + If it is **successful**, return the following message, with the **name of the explored planet** and the **number** of the **astronauts** that **had gone out in open space**: **"Planet: {planet\_name} was explored. {astronauts} astronauts participated in collecting items."**
  + Otherwise, return: **"Mission is not completed."**
* **def** send\_on\_mission(self, planet\_name: str):  
   find\_planet = [p **for** p **in** self.planet\_repository.planets **if** p.name == planet\_name]  
   **if not** find\_planet:  
   **raise** Exception(**"Invalid planet name!"**)  
    
   sorted\_astronaut = []  
   **for** astronaut **in** self.astronaut\_repository.astronauts:  
   **if** astronaut.oxygen > **30**:  
   sorted\_astronaut.append(astronaut)  
    
   **if not** sorted\_astronaut:  
   **raise** Exception(**"You need at least one astronaut to explore the planet!"**)  
   choosing\_astronaut = []  
   **for** astronaut **in** sorted(sorted\_astronaut, key=**lambda** x: -x.oxygen):  
   **if** len(choosing\_astronaut) < **5**:  
   choosing\_astronaut.append(astronaut)  
   **else**:  
   **break** planet = find\_planet[**0**]  
   counter = **0  
    
   for** astronaut **in** choosing\_astronaut:  
   **if** planet.items:  
   counter += **1  
   for** item **in** planet.items[::-**1**]:  
    
   astronaut.backpack.append(item)  
   planet.items.remove(item)  
   astronaut.breathe()  
   **if** astronaut.oxygen <= **0**:  
   **break  
   else**:  
   self.number\_of\_successful\_missions += **1  
   return f"Planet:** {planet\_name} **was explored.** {counter} **astronauts participated in collecting items."** self.number\_of\_not\_completed\_missions += **1  
   return "Mission is not completed."**

#### report()

* Returns an information about the **number of successful missions**, the number of **not completed missions**, and an informationabout **all the astronauts in the space station**. If an astronaut **doesn't have items in the backpack**, return **"none"** instead:

**"{number\_of\_successful\_missions} successful missions!**

**{number\_of\_not\_completed\_missions} missions were not completed!**

**Astronauts' info:**

**Name: {astronaut\_name1}**

**Oxygen: {astronaut\_oxygen1}**

**Backpack items: {bag\_item1, bag\_item2, bag\_item3, …, bag\_itemn \ "none"}**

**...**

**Name: {astronaut\_nameN}**

**Oxygen: {astronaut\_oxygenN}**

**Backpack items: {bag\_item1, bag\_item2, bag\_item3, …, bag\_itemn \ "none"}"**

**def** report(self):  
 result = [**f"**{self.number\_of\_successful\_missions} **successful missions!"**,  
 **f"**{self.number\_of\_not\_completed\_missions} **missions were not completed!"**,  
 **f"Astronauts' info:"**]  
 **for** astronaut **in** self.astronaut\_repository.astronauts:  
 result.extend([**f"Name:** {astronaut.name}**"**,  
 **f"Oxygen:** {astronaut.oxygen}**"**,  
 **f"Backpack items:** {**', '**.join(astronaut.backpack) **if** astronaut.backpack **else 'none'**}**"**])  
  
 **return '**\n**'**.join(result)

# Problem 3. Unit Tests

You will **be provided with another skeleton** for this problem. You should **write tests** for the **Library** class. It will have some **properties**, **methods**, and a **constructor**, all of them **working properly**. You are **NOT ALLOWED** to change the class. Cover the whole class with unit tests to make sure that the class is working as intended. Submit **only the test** folder.