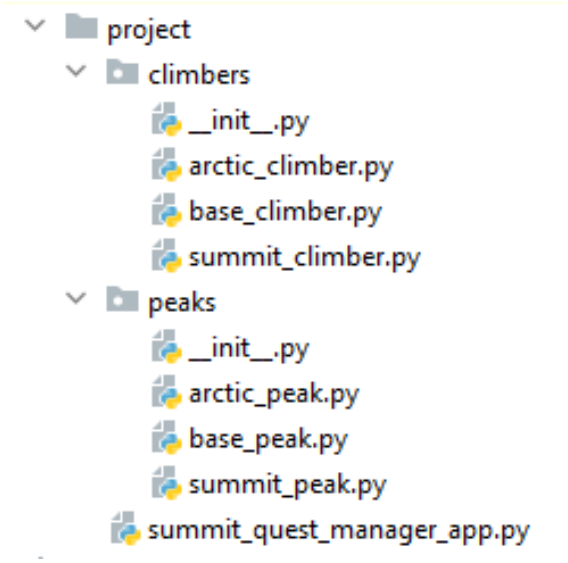
Python OOP Retake Exam - 19 December 2023

In the picturesque mountain village of Summitville, an air of excitement sweeps through the crisp alpine air as climbers from far and wide prepare for the annual SummitQuest Challenge! Adventurous souls, ranging from seasoned Arctic climbers to daring summit enthusiasts, gather to conquer the towering peaks that surround the village. Each peak presents its own set of challenges, from the icy cliffs of Mount Everest to the rugged faces of K2.

The peaks are not only a test of physical strength but a showcase of strategic prowess, as climbers carefully select their gear to navigate the treacherous terrains. The village buzzes with anticipation as climbers register for the challenge and add their dream summits to their wish lists.

As the sun rises over the majestic peaks, the SummitQuest Manager App comes to life, tracking climbers' progress, ensuring they have the required gear, and capturing the thrill of each conquest. The challenge is not just about reaching the summits; it's about the camaraderie among climbers, the breathtaking views, and the sheer joy of overcoming nature's obstacles.

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



Judge Upload

For the **first two 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 **your venv, .idea, pycache, and __MACOSX (for Mac users) in the zip file**, so you do not exceed **the maximum allowed size** of **`16.00` KB**.

Structure (Problem 1) and Functionality (Problem 2)

Our task is to implement all the classes' structure and functionality (properties, methods, inheritance, abstraction, 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 final result in accordance with its requirements so that the program works properly.

1. Class BasePeak

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

Structure

The class should have the following attributes:

- **`name: str`**
  - The value represents the **name of the peaks**.
  - If the name is **less than 2 symbols**, raise a **`ValueError`** with the message: **`"Peak name cannot be less than 2 symbols!"`**
- **`elevation: int`**
  - Represents the **elevation of each peak**.
  - If the peak elevation is **below 1500m**, raise a **`ValueError`** with the message: **`"Peak elevation cannot be below 1500m."`**
- **`difficulty_level: calculate_difficulty_level()`**
  - Essentially, it **sets the difficulty level** of the peak based on the **calculation performed** in the **`calculate_difficulty_level()`** method.

Methods

__init__(name: str, elevation: int)

- In the **`__init__`** method, all the needed attributes must be set.

get_recommended_gear()

- **Returns** a list of recommended gear.
- Keep in mind that **each type of peak** has specific requirements for the gear.

calculate_difficulty_level()

- **Returns** the **difficulty level** depending on the peak's elevation.
- Keep in mind that **each type of peak** can implement the **method differently**.

2. Class ArcticPeak

In the **`arctic_peak.py`** file, the class **`ArcticPeak`** should be implemented. The Arctic peak is a **type of peak**.

Methods

__init__(name: str, elevation: int)

- In the **`__init__`** method, all the needed attributes must be set.

get_recommended_gear()

- The recommended gear to climb the Arctic peak is

  `["Ice axe", "Crampons", "Insulated clothing", "Helmet"]`

calculate_difficulty_level()

- The difficulty level is **`"Extreme"`** if the **peak elevation is higher than 3000**.
- The difficulty level is **`"Advanced"`** if the **peak elevation is between 2000 and 3000 (both inclusive)**.

3. Class SummitPeak

In the **`summit_peak.py`** file, the class **`SummitPeak`** should be implemented. The Summit peak is a **type of peak**.

Methods

__init__(name: str, elevation: int)

- In the **`__init__`** method, all the needed attributes must be set.

get_recommended_gear()

- The recommended gear to climb the Summit peak is

  `["Climbing helmet", "Harness", "Climbing shoes", "Ropes"]`

calculate_difficulty_level()

- The difficulty level is **`"Extreme"`** if the **peak elevation is higher than 2500**.
- The difficulty level is **`"Advanced"`** if the **peak elevation is between 1500 and 2500 (both inclusive)**.

4. Class BaseClimber

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

Structure

The class should have the following attributes:

- **`name: str`**
  - The value represents the **name of the climber**.
  - If the name is **an empty string or contains only white spaces**, raise a **`ValueError`** with the message: **`"Climber name cannot be null or empty!"`**
- **`strength: float`**
  - Represents the **strength that each climber** has based on their **type**.
  - If the **strength falls below or equals 0**, raise a **`ValueError`** with the message: **`"A climber cannot have negative strength or strength equal to 0!"`**
- **`conquered_peaks: list`**

- It will store a sequence of peaks conquered by each climber.

- **`is_prepared: bool`**
  - The property indicates if the climber is prepared to climb a certain peak. Its **initial value is `True`**, representing that the climber has the required gear.

Methods

__init__(name: str, strength: float)

- In the **`__init__`** method, all the needed attributes must be set.

can_climb()

- The method checks whether the **climber has enough strength required to attempt a climb**. It returns **True** if the climber's **strength is greater than or equal to the specified minimum strength** to climb.
- Keep in mind that each **type of climber** has a **different minimum strength**, and it will implement the **method differently**.

climb(peak : BasePeak)

- The method takes a peak parameter that represents the peak the climber is trying to conquer. The **climber's strength** is **reduced by a specific amount** based on the peak's difficulty level, and a **list is collected of conquered peaks** by each climber.
- Keep in mind that **each type of climber** can implement the **method differently**.

rest ()

- The method **increases** the climber's **strength by 15 points**.

__str__()

- Returns a **`string`** with **information** about the **`climber`** in the format below.
  - The strength should be rounded to the **first decimal place**.
  - The conquered peaks should be separated by a comma and a space **`(", ")`**

`"{type of climber}: /// Climber name: {name} * Left strength: {strength} * Conquered peaks: {conquered_peaks} ///"`

5. Class ArcticClimber

In the **`arctic_climber.py`** file, the class **`ArcticClimber`** should be implemented. The Arctic climber is a **type of climber**. Each climber has an **initial strength of 200.**

Methods

__init__(name: str)

- In the **`__init__`** method, all the needed attributes must be set.

can_climb ()

- The Arctic climber can attempt the climb only if his or her **strength is greater than or equal to 100**.

climb(peak : BasePeak)

- The method takes a **peak parameter** and performs the following actions:

- The climber's **`strength`** will be **reduced by 20, multiplied by 2 if the peak difficulty level is `"Extreme"`** otherwise, it will **decrease the strength by 20, multiplied by 1.5**.
- The **peak name** will be added to the **conquered peaks list**.

6. Class SummitClimber

In the **`summit_climber.py`** file, the class **`SummitClimber`** should be implemented. The Summit climber is a **type of climber**. Each climber has an **initial strength of 150.**

Methods

__init__(name: str)

- In the **`__init__`** method, all the needed attributes must be set.

can_climb ()

- The Summit climber can attempt the climb only if his or her **strength is greater than or equal to 75**.

climb(peak : BasePeak)

- The method takes a **peak parameter** and performs the following actions:
  - The climber's **`strength`** will be **reduced by 30, multiplied by 1.3 if the peak difficulty level is `"Advanced"`** otherwise, it will **decrease the strength by 30, multiplied by 2.5**.
  - The **peak name** will be added to the **conquered peaks list**.

7. Class SummitQuestManagerApp

In the **`summit_quest_manager_app.py`** file, the class **`SummitQuestManagerApp`** should be implemented. It will contain the functionality of the project.

Structure

The class should have the following attributes:

- **`climbers: list`**
  - An empty list to store all climber objects registered for the Summit Quest.
- **`peaks: list`**
  - An empty list to store all peak objects that are part of the wish list to climb.

Methods

__init__()

- In the **`__init__`** method, all the needed attributes must be set.

register_climber(climber_type: str, climber_name: str)

The method **creates** a climber of the given type and **adds** it to the **`climbers`** collection.

- If the climber's type is **not valid**, return the following message:

  `"{climber_type} doesn't exist in our register."`
- If a climber with the same **name** is already added to the list, **do not duplicate records**, return the following message:

`"{climber_name} has been already registered."`

- If none of the above cases is reached, the **climber** is successfully created. Store the climber in the appropriate collection and return it:

  `"{climber_name} is successfully registered as a {climber_type}."`

- **Valid types** of climbers are: `"ArcticClimber"` and `"SummitClimber"`

peak_wish_list(peak_type: str, peak_name: str, peak_elevation: int)

The method **creates** a peak of the given type and **adds** it to the `peaks` collection. The method is responsible for **allowing a new peak to climb**.

- First, check if the **peak type** is valid, and if **not**, return the following message:

  `"{peak_type} is an unknown type of peak."`

- If the above case is not reached, create the correct type of **peak** and add it to the appropriate collection. Return the following message:
  `"{peak_name} is successfully added to the wish list as a {peak_type}."`
- **Valid types** of peaks are: `"ArcticPeak"` and `"SummitPeak"`.

check_gear(climber_name: str, peak_name: str, gear: List[str]):

The method is responsible for verifying if every **climber has the required gear for each peak**.

- If the climber has **the same set of gear** as the **recommended one**, then return the following message:

  `"{climber_name} is prepared to climb {peak_name}."`

- If the climber's **gear set lacks even one of the recommended items** for the peak, set `is_prepared` to `False` and return the following message:

  `"{climber_name} is not prepared to climb {peak_name}. Missing gear: {missing gear separated by ', ', sorted alphabetically}."`

perform_climbing(climber_name: str, peak_name: str):

The method is responsible for allowing the **climber to conquer a specific peak**:

- **Climber Validation:**
  - Validates the existence of a climber with the given `climber_name` in the collection of registered `climbers`.
  - If **no climber is found**, the method returns the message:

    `"Climber {climber_name} is not registered yet."`

- **Peak Validation:**
  - Validates the existence of a peak with the given `peak_name` in the **wish list**.
  - If **no peak is found**, the method returns the message:

    `"Peak {peak_name} is not part of the wish list."`

- **Climb Check:**
  - Checks if the climber's `can_climb` and `is_prepared` are both `True`, indicating that a climb can be performed. Returns the following message:

`"{climber_name} conquered {peak_name} whose difficulty level is {difficulty_level}."`

- Check if the **climber is not prepared** to climb, then return the following message:

  `"{climber_name} will need to be better prepared next time."`

- Otherwise, the climber will need some rest. Returns the message:

  `"{climber_name} needs more strength to climb {peak_name} and is therefore taking some rest."`

get_statistics ()

Returns detailed information about the **Summit Quest**. Only climbers who **successfully manage to conquer peaks** should be included. They need to be ordered by the **conquered peak in descending order**. If there is more than one climber with the same number of conquered peaks, **then order them by climber name alphabetically**. When there is **more than one conquered peak in the list**, they need to be **ordered alphabetically** as well. The static information should follow the format each on the new line:

`"Total climbed peaks: {how many peaks were conquered}`

`**Climber's statistics:**`

`{climber₁}`

`{climber₂}`

`...`

`{climberₙ}"`

Examples

Input

`# Create an instance of SummitQuestManagerApp`
`climbing_app = SummitQuestManagerApp()`
 	
`# Register climbers`
`print(climbing_app.register_climber("ArcticClimber", "Alice"))`
`print(climbing_app.register_climber("SummitClimber", "Bob"))`
`print(climbing_app.register_climber("ExtremeClimber", "Dave"))`
`print(climbing_app.register_climber("ArcticClimber", "Charlie"))`
`print(climbing_app.register_climber("ArcticClimber", "Alice"))`
`print(climbing_app.register_climber("SummitClimber", "Eve"))`
`print(climbing_app.register_climber("SummitClimber", "Frank"))`
 	
`# Add peaks to the wish list`
`print(climbing_app.peak_wish_list("ArcticPeak", "MountEverest", 4000))`
`print(climbing_app.peak_wish_list("SummitPeak", "K2", 3000))`
`print(climbing_app.peak_wish_list("ArcticPeak", "Denali", 2500))`
`print(climbing_app.peak_wish_list("UnchartedPeak", "MysteryMountain", 2000))`
 	
`# Prepare climbers for climbing`
`print(climbing_app.check_gear("Alice", "MountEverest", ["Ice axe", "Crampons", "Insulated clothing", "Helmet"]))`

```
print(climbing_app.check_gear("Bob", "K2", ["Climbing helmet", "Harness", "Climbing shoes",
"Ropes"]))
print(climbing_app.check_gear("Charlie", "Denali", ["Ice axe", "Crampons"]))

# Perform climbing
print(climbing_app.perform_climbing("Alice", "MountEverest"))
print(climbing_app.perform_climbing("Bob", "K2"))
print(climbing_app.perform_climbing("Kelly", "Denali"))
print(climbing_app.perform_climbing("Alice", "K2"))
print(climbing_app.perform_climbing("Alice", "MysteryMountain"))
print(climbing_app.perform_climbing("Eve", "MountEverest"))
print(climbing_app.perform_climbing("Charlie", "MountEverest"))
print(climbing_app.perform_climbing("Frank", "K2"))
print(climbing_app.perform_climbing("Frank", "Denali"))
print(climbing_app.perform_climbing("Frank", "MountEverest"))

# Get statistics
print(climbing_app.get_statistics())
```

Output

```
Alice is successfully registered as a ArcticClimber.
Bob is successfully registered as a SummitClimber.
ExtremeClimber doesn't exist in our register.
Charlie is successfully registered as a ArcticClimber.
Alice has been already registered.
Eve is successfully registered as a SummitClimber.
Frank is successfully registered as a SummitClimber.
MountEverest is successfully added to the wish list as a ArcticPeak.
K2 is successfully added to the wish list as a SummitPeak.
Denali is successfully added to the wish list as a ArcticPeak.
UnchartedPeak is an unknown type of peak.
Alice is prepared to climb MountEverest.
Bob is prepared to climb K2.
Charlie is not prepared to climb Denali. Missing gear: Helmet, Insulated clothing.
Alice conquered MountEverest whose difficulty level is Extreme.
Bob conquered K2 whose difficulty level is Extreme.
Climber Kelly is not registered yet.
Alice conquered K2 whose difficulty level is Extreme.
Peak MysteryMountain is not part of the wish list.
Eve conquered MountEverest whose difficulty level is Extreme.
Charlie will need to be better prepared next time.
Frank conquered K2 whose difficulty level is Extreme.
Frank conquered Denali whose difficulty level is Advanced.
Frank needs more strength to climb MountEverest and is therefore taking some rest.
Total climbed peaks: 3
**Climber's statistics:**
ArcticClimber: /// Climber name: Alice * Left strength: 120.0 * Conquered peaks: K2,
MountEverest ///
SummitClimber: /// Climber name: Frank * Left strength: 51.0 * Conquered peaks: Denali, K2 ///
SummitClimber: /// Climber name: Bob * Left strength: 75.0 * Conquered peaks: K2 ///
SummitClimber: /// Climber name: Eve * Left strength: 75.0 * Conquered peaks: MountEverest ///
```

Task 3: Unit Tests (100 points)

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