Exercise: First Steps in OOP

Problems for exercise and homework for the Python OOP Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/1935.

1. Shop

Create a class called **Shop**. Upon initialization, it should receive a **name** (string) and **items** (list). Create a method called **get_items_count()** which should return the **number of items** in the store.

Examples

Test Code	Output
`shop = Shop("My Shop", ["Apples", "Bananas", "Cucumbers"])` `print(shop.get_items_count())`	3

2. Hero

Create a class called **Hero**. Upon initialization, it should receive a **name** (string) and **health** (number). Create two additional methods:

- **defend(damage)** - reduce the given **damage** from the hero's health:
  - if the **health** becomes 0 or less, **set** it **to 0** and **return "{name} was defeated"**
- **heal(amount)** - **increase the health** of the hero with the given amount

Examples

Test Code	Output

`hero = Hero("Peter", 100)`
`print(hero.defend(50))`
`hero.heal(50)`
`print(hero.defend(99))`
`print(hero.defend(1))`
	None
	None
	Peter was defeated

3. Employee

Create class **Employee**. Upon initialization, it should receive **id** (number), **first_name** (string), **last_name** (string), and **salary** (number). Create **3 additional instance methods**:

- **get_full_name()** - returns **"{first_name} {last_name}"**
- **get_annual_salary()** - returns the total salary for **12 months**
- **raise_salary(amount)** - **increases the salary** by the given amount and **returns 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

4. Cup

Create a class called **Cup**. Upon initialization, it should receive **`size`** (integer) and **`quantity`** (an integer representing **how much liquid** is in it).

The class should have **two methods**:

- **`fill(quantity)`** that will **increase** the amount of liquid in the cup with the given **quantity** (if there is **space** in the cup, **otherwise ignore**).
- **`status()`** that will **return** the **amount** of **free space** left in the cup.

Submit only the class in the judge system. Do not forget to test your code.

Examples

Test Code	Output

`cup = Cup(100, 50)`
`print(cup.status())`
`cup.fill(40)`
`cup.fill(20)`
`print(cup.status())`
	`50`
	`10`

5. Flower

Create a class called **`Flower`**. Upon initialization, the class should receive a **`name`** (string) and a **`water_requirements`** (number). The flower should also have an instance attribute called **`is_happy`** (**`False`** by default). Add **two additional methods** to the class:

- **`water(quantity)`** - it will water the flower. **Each time** check if the quantity is **greater than or equal** to the required. If it is - the flower becomes happy (set **`is_happy`** to **`True`**).
- **`status()`** - it should return **`"{name} is happy"`** if the flower **is happy**, otherwise it should return **`"{name} is not happy"`**.

Submit only the class in the judge system.

Examples

Test Code	Output

`flower = Flower("Lilly", 100)`
`flower.water(50)`
`print(flower.status())`
`flower.water(60)`
`print(flower.status())`
`flower.water(100)`
`print(flower.status())`
	`Lilly is not happy`
	`Lilly is not happy`
	`Lilly is happy`

6. Steam User

Create a class called **`SteamUser`**. Upon initialization, it should receive a **`username`** (string) and **`games`** (list). It should also have an **attribute** called **`played_hours`** (**`0`** by default). Add **three methods** to the class:

- **`play(game, hours)`**
  - If the **game** is in the **game list, increase** the **`played_hours`** by the given hours and return **`"{username} is playing {game}"`**
  - Otherwise, return **`"{game} is not in library"`**

- **buy_game(game)**
  - If the game **is not** in the game list, **add it** and return **"{username} bought {game}"**
  - Otherwise, return **"{game} is already in your library"**
- **status()** - returns the following:
  "{username} has {games_count} games. Total play time: {played_hours}"

Submit only the class in the judge system.

Examples

Test Code	Output

`user = SteamUser("Peter", ["Rainbow Six Siege", "CS:GO", "Fortnite"])`
`print(user.play("Fortnite", 3))`
`print(user.play("Oxygen Not Included", 5))`
`print(user.buy_game("CS:GO"))`
`print(user.buy_game("Oxygen Not Included"))`
`print(user.play("Oxygen Not Included", 6))`
`print(user.status())`
	`Peter is playing Fortnite`
	`Oxygen Not Included is not in library`
	`CS:GO is already in your library`
	`Peter bought Oxygen Not Included`
	`Peter is playing Oxygen Not Included`
	`Peter has 4 games. Total play time: 9`

7. Programmer

Create a class called **Programmer**. Upon initialization, it should receive - **name** (string), **language** (string), **skills** (integer). The class should have **two methods**:

- **watch_course(course_name, language, skills_earned)**
  - If the programmer's **language** is **the same as** the **one on the course, increase his skills** with the given amount and return a message **"{name} watched {course_name}"**.
  - Otherwise return **"{name} does not know {language}"**.
- **change_language(new_language, skills_needed)**
  - If the programmer **has the skills** and the **new language is not the same as his**, **change** his language to the new one and return **"{name} switched from {previous_language} to {new_language}"**.
  - If the programmer **has the skills**, but the given **language is equal** to his return **"{name} already knows {language}"**.
  - In the last case, the programmer does **not have enough skills**, so return **"{name} needs {needed_skills} more skills"** and **do not change his language**.

Submit only the class in the judge system.

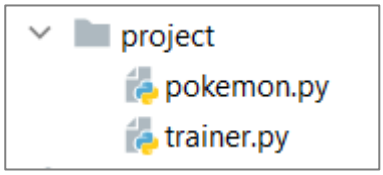
Examples

Test Code	Output

`programmer = Programmer("John", "Java", 50)`
`print(programmer.watch_course("Python Masterclass", "Python", 84))`
`print(programmer.change_language("Java", 30))`
`print(programmer.change_language("Python", 100))`
`print(programmer.watch_course("Java: zero to hero", "Java", 50))`
`print(programmer.change_language("Python", 100))`
`print(programmer.watch_course("Python Masterclass", "Python", 84))`
	`John does not know Python`
	`John already knows Java`
	`John needs 50 more skills`
	`John watched Java: zero to hero`
	`John switched from Java to Python`
	`John watched Python Masterclass`

8. Pokemon Battle*

Note: For this problem, 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 a module named project.



You are tasked to create **two classes**: **a Pokemon** class in the **pokemon.py** file and **a Trainer** class in the **trainer.py** file.

The **Pokemon** class should receive a **name** (string) and **health** (int) upon initialization. It should also have a method called **pokemon_details** that returns **the information about the pokemon: "{pokemon_name} with health {pokemon_health}"**

The **Trainer** class should receive a **name** (string). The Trainer should also have an attribute **pokemons** (list, empty by default). The Trainer has **three methods**:

- **add_pokemon(pokemon: Pokemon)**
  - Adds the **pokemon to the collection** and **returns "Caught {pokemon_name} with health {pokemon_health}"**. **Hint**: use the pokemon's details method.
  - If the pokemon is already in the collection, returns **"This pokemon is already caught"**
  - **Hint**: to import the **Pokemon** class, you should add **"from project.pokemon import Pokemon"**
- **release_pokemon(pokemon_name: string)**
  - Checks if you have a pokemon with that name and **removes it from the collection**. In the end, returns **"You have released {pokemon_name}"**
  - If there **is no pokemon** with that name in the collection, returns **"Pokemon is not caught"**
- **trainer_data()**
  - The method returns the information about the trainer and his pokemon's collection in the format:

```
"Pokemon Trainer {trainer_name}
 Pokemon count {the amount of pokemon caught}
 - {pokemon_details1}
 ...
 - {pokemon_detailsN}"
```

Examples

Test Code	Output

`pokemon = Pokemon("Pikachu", 90)`
`print(pokemon.pokemon_details())`
`trainer = Trainer("Ash")`
`print(trainer.add_pokemon(pokemon))`
`second_pokemon = Pokemon("Charizard", 110)`
`print(trainer.add_pokemon(second_pokemon))`
`print(trainer.add_pokemon(second_pokemon))`
`print(trainer.release_pokemon("Pikachu"))`
`print(trainer.release_pokemon("Pikachu"))`
`print(trainer.trainer_data())`
	`Pikachu with health 90`
	`Caught Pikachu with health 90`
	`Caught Charizard with health 110`
	`This pokemon is already caught`
	`You have released Pikachu`
	`Pokemon is not caught`
	`Pokemon Trainer Ash`
	`Pokemon count 1`
	`- Charizard with health 110`