

# Exercise: Classes and Objects

Problems for exercise and homework for the [Python Fundamentals Course @SoftUni](#).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.org/Contests/1734>.

**Note: Submit only the classes in the judge system for all tasks. Test your classes with your own code to see if they work correctly.**

## 1. Storage

Create a **class Storage**. The `__init__` method should accept **one parameter** - the **capacity** of the storage. The Storage class should also have an attribute called **storage** - empty list, where all the items will be stored.

The class should have two additional methods:

- `add_product(product: str)` - adds the product in the storage **if there is enough space** for it
- `get_products()` - returns the storage list

### Example

Test Code	Output
<pre>storage = Storage(4) storage.add_product("apple") storage.add_product("banana") storage.add_product("potato") storage.add_product("tomato") storage.add_product("bread") print(storage.get_products())</pre>	<pre>["apple", "banana", "potato", "tomato"]</pre>

## 2. Weapon

Create a **class Weapon**. The `__init__` method should receive a number of **bullets** (integer). Create an attribute called **bullets** to store that number. The class should also have the following methods:

- `shoot()`
  - If there are bullets in the weapon, **reduce** them **by 1** and return a message **"shooting..."**
  - If there are **no bullets** left, return: **"no bullets left"**
- `__repr__()`
  - Returns **"Remaining bullets: {amount\_of\_bullets}"**
  - You can read more about the method here: [link](#)

### Example

Test Code	Output
<pre>weapon = Weapon(5) print(weapon.shoot()) print(weapon.shoot()) print(weapon) print(weapon.shoot()) print(weapon.shoot()) print(weapon.shoot()) print(weapon.shoot()) print(weapon)</pre>	<pre>shooting... shooting... Remaining bullets: 3 shooting... shooting... shooting... no bullets left Remaining bullets: 0</pre>

### 3. Catalogue

Create a **class Catalogue**. The `__init__` method should accept the **name** of the catalogue (string). Each catalogue should also have an **attribute** called **products**, an empty **list**. The class should also have **three more methods**:

- `add_product(product_name: str)` - adds the product to the products' list
- `get_by_letter(first_letter: str)` - returns a **list** containing only the products that start with the given letter
- `__repr__` - returns the catalogue info in the following format:  
"Items in the {name} catalogue:  
{item1}  
{item2}  
...  
{itemN}"

The items should be **sorted alphabetically in ascending order**.

#### Example

Test Code	Output
<pre>catalogue = Catalogue("Furniture") catalogue.add_product("Sofa") catalogue.add_product("Mirror") catalogue.add_product("Desk") catalogue.add_product("Chair") catalogue.add_product("Carpet") print(catalogue.get_by_letter("C")) print(catalogue)</pre>	<pre>["Chair", "Carpet"] Items in the Furniture catalogue: Carpet Chair Desk Mirror Sofa</pre>

### 4. Town

Create a **class Town**. The `__init__` method should receive the **name** of the town (string). Each town has a **latitude** - "0°N" upon initialization and a **longitude** - "0°E" upon initialization. It should also have **3 more methods**:

- `set_latitude(latitude)` - sets an **latitude**
- `set_longitude(longitude)` - sets an **longitude**
- `__repr__` - returns a representation of the object in the following string format:  
"Town: {name} | Latitude: {latitude} | Longitude: {longitude}"

#### Example

Test Code	Output
<pre>town = Town("Sofia") town.set_latitude("42° 41' 51.04\" N") town.set_longitude("23° 19' 26.94\" E") print(town)</pre>	<pre>Town: Sofia   Latitude: 42° 41' 51.04" N   Longitude: 23° 19' 26.94" E</pre>

### 5. Class

Create a **class Class**. The `__init__` method should receive the **name** of the class. Each class should also have **2 empty lists** - **students** and **grades**. Create a **class attribute** `__students_count` equal to **22**. The class should also have **3 additional methods**:

- **add\_student(name: str, grade: float)** - adds the **student** and the **grade** in the two lists if there is **free space** in the class
- **get\_average\_grade()** - returns the **average** of all existing **grades** formatted to the **second decimal point** (as a **number**)
- **\_\_repr\_\_** - returns the string (**single line**):

"The students in {class\_name}: {students}. Average grade: {average\_grade}"

The students must be separated by a comma and a space: ", ".

## Example

Test Code	Output
<pre>a_class = Class("11B") a_class.add_student("Peter", 4.80) a_class.add_student("George", 6.00) a_class.add_student("Amy", 3.50) print(a_class)</pre>	<pre>The students in 11B: Peter, George, Amy. Average grade: 4.77</pre>

## 6. Inventory

Create a **class Inventory**. The **\_\_init\_\_** method should accept only the **\_\_capacity: int** (private attribute) of the inventory. You can read more about private attributes [here](#). Each inventory should also have an attribute called **items - empty list**, where all the items will be stored. The class should also have **3 methods**:

- **add\_item(item: str)** - adds the item in the inventory if there is space for it. Otherwise, returns **"not enough room in the inventory"**
- **get\_capacity()** - returns the value of **\_\_capacity**
- **\_\_repr\_\_()** - returns **"Items: {items}.\nCapacity left: {left\_capacity}"**. The items should be separated by ", "

## Example

Test Code	Output
<pre>inventory = Inventory(2) inventory.add_item("potion") inventory.add_item("sword") print(inventory.add_item("bottle")) print(inventory.get_capacity()) print(inventory)</pre>	<pre>not enough room in the inventory 2 Items: potion, sword. Capacity left: 0</pre>

## 7. Articles

Create a **class** called **Article**. The **\_\_init\_\_** method should accept **3 arguments: title: str, content: str, and author: str**. The class should also have **4 methods**:

- **edit(new\_content: str)** - changes the old content to the new one
- **change\_author(new\_author: str)** - changes the old author with the new one
- **rename(new\_title: str)** - changes the old title with the new one
- **\_\_repr\_\_()** - returns the following string **"{title} - {content}: {author}"**

## Example

Test Code	Output
-----------	--------

<pre> article = Article(     "Highest Recorded Temperature",     "Temperatures across Europe are unprecedented, according to scientists.",     "Ben Turner" ) article.edit(     "Syracuse, a city on the coast of the Italian island of Sicily, registered temperatures of 48.8 degrees Celsius" ) article.rename(     "Temperature in Italy" ) article.change_author(     "B. T." ) print(article) </pre>	<p>Temperature in Italy - Syracuse, a city on the coast of the Italian island of Sicily, registered temperatures of 48.8 degrees Celsius: B. T.</p>
--	---

## 8. \* Vehicle

Create a **class Vehicle**. The **\_\_init\_\_** method should receive a **type**, a **model**, and a **price**. You should also set an **owner** to **None**. The class should have the following methods:

- **buy(money: int, owner: str)**
  - If the person **has enough money** and the vehicle has **no owner**, returns: **"Successfully bought a {type}. Change: {change}"** and **sets the owner** to the given one
  - If the **money is not enough**, return: **"Sorry, not enough money"**
  - If the car **already has an owner**, return: **"Car already sold"**
- **sell()**
  - If the car **has an owner**, set it to **None** again.
  - Otherwise, return: **"Vehicle has no owner"**
- **\_\_repr\_\_()**
  - If the vehicle **has an owner**, returns: **"{model} {type} is owned by: {owner}"**.
  - Otherwise, return: **"{model} {type} is on sale: {price}"**

## Example

Test Code	Output
<pre> vehicle_type = "car" model = "BMW" price = 30000 vehicle = Vehicle(vehicle_type, model, price) print(vehicle.buy(15000, "Peter")) print(vehicle.buy(35000, "George")) print(vehicle) vehicle.sell() print(vehicle) </pre>	<p>Sorry, not enough money  Successfully bought a car. Change: 5000.00  BMW car is owned by: George  BMW car is on sale: 30000</p>

## 9. \* Movie

Create a **class Movie**. The **\_\_init\_\_** method should receive a **name** and a **director**. It should also have a default value of an attribute called **watched** set to **False**. There should also be a class attribute **\_\_watched\_movies** which will keep track of the number of all the watched movies. The class should have the following methods:

- **change\_name(new\_name: str)** - changes the name of the movie
- **change\_director(new\_director: str)** - changes the director of the movie
- **watch()** - change the **watched** attribute to **True** and **increase** the **total watched** movies class attribute (if the movie is **not already watched**)
- **\_\_repr\_\_()** - returns "Movie name: {name}; Movie director: {director}. Total watched movies: {\_\_watched\_movies}"

### Example

Test Code	Output
<pre>first_movie = Movie("Inception", "Christopher Nolan") second_movie = Movie("The Matrix", "The Wachowskis") third_movie = Movie("The Predator", "Shane Black") first_movie.change_director("Me") third_movie.change_name("My Movie") first_movie.watch() third_movie.watch() first_movie.watch() print(first_movie) print(second_movie) print(third_movie)</pre>	<pre>Movie name: Inception; Movie director: Me. Total watched movies: 2 Movie name: The Matrix; Movie director: The Wachowskis. Total watched movies: 2 Movie name: My Movie; Movie director: Shane Black. Total watched movies: 2</pre>