# Exercise: Classes and Objects

Problems for exercise and homework for the [Python Fundamentals Course @SoftUni](https://softuni.bg/trainings/3450/programming-fundamentals-with-python-september-2021).

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.***

## 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** |
| 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()) | ["apple", "banana", "potato", "tomato"] |

## 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](https://www.journaldev.com/22460/python-str-repr-functions)

### Example

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| 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) | shooting...  shooting...  Remaining bullets: 3  shooting...  shooting...  shooting...  no bullets left  Remaining bullets: 0 |

## 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** |
| 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) | ["Chair", "Carpet"]  Items in the Furniture catalogue:  Carpet  Chair  Desk  Mirror  Sofa |

## 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** |
| town = Town("Sofia")  town.set\_latitude("42° 41\' 51.04\" N")  town.set\_longitude("23° 19\' 26.94\" E")  print(town) | Town: Sofia | Latitude: 42° 41' 51.04" N | Longitude: 23° 19' 26.94" E |

## 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** |
| 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) | The students in 11B: Peter, George, Amy. Average grade: 4.77 |

## 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](https://www.tutorialsteacher.com/python/private-and-protected-access-modifiers-in-python). 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** |
| inventory = Inventory(2)  inventory.add\_item("potion")  inventory.add\_item("sword")  print(inventory.add\_item("bottle"))  print(inventory.get\_capacity())  print(inventory) | not enough room in the inventory  2  Items: potion, sword.  Capacity left: 0 |

## 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** |
| 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) | Temperature in Italy - Syracuse, a city on the coast of the Italian island of Sicily, registered temperatures of 48.8 degrees Celsius: B. T. |

## \* 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** |
| vehicle\_type = "car"  model = "BMW"  price = 30000  vehicle = Vehicle(vehicle\_type, model, price)  vehicle.buy(15000, "Peter")  vehicle.buy(35000, "George")  print(vehicle)  vehicle.sell()  print(vehicle) | Sorry, not enough money  Successfully bought a car. Change: 5000.00  BMW car is owned by: George  BMW car is on sale: 30000 |

## \* 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** |
| 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) | 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 |