# Lab: Attributes and Methods

Problems for in-class lab for the [Python OOP Course @SoftUni](https://softuni.bg/courses/python-oop).   
Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/2430>

## Store

Create a class called **Store**. Upon initialization it should receive a **name (str)**, **type (str)**, **capacity (int)**. The store should also have an **attribute** called **items** (**dictionary** that stores **name** of an item and its **quantity**). The class should have **4 methods**:

* **from\_size (name:str, type:str, size:int)** – a **new instance** should be created with capacity which is **50% of the size**
* **add\_item(item\_name:str)** – adds **1** to the quantity of the given **item**. On **success**, the method should **return "{item\_name} added to the store"**. If the addition is **not possible**, the following message should be returned **"Not enough capacity in the store"**
* **remove\_item(item\_name:str, amount:int)** – **removes** the given amount from the **item**. On **success**, it should return **"{count} {item\_name} removed from the store"**. **Otherwise**, the method should return **"Cannot remove {count} {item\_name}"**
* **\_\_repr\_\_()** - returns a string representation in the format **"{store\_name} of type {store\_type} with capacity {store\_capacity}"**

### Examples

|  |
| --- |
| **Test Code** |
| first\_store = Store("First store", "Fruit and Veg", 20)  second\_store = Store.from\_size("Second store", "Clothes", 500)  print(first\_store)  print(second\_store)  print(first\_store.add\_item("potato"))  print(second\_store.add\_item("jeans"))  print(first\_store.remove\_item("tomatoes", 1))  print(second\_store.remove\_item("jeans", 1)) |
| **Output** |
| First store of type Fruit and Veg with capacity 20  Second store of type Clothes with capacity 250  potato added to the store  jeans added to the store  Cannot remove 1 tomatoes  1 jeans removed from the store |

## Integer

Create a class called **Integer**. Upon initialization it should receive a single parameter **value** (**int**). It should have **4 methods**:

* **from\_float(value)** - creates a **new instance** by **flooring** the provided floating number. If the value is **not a float** return a message **"value is not a float"**
* **from\_roman(value)** – creates a **new instance** by converting the **roman** number (**as string**) to an integer
* **from\_string(value)** - creates a **new instance** by converting the **string** to an integer (if the value **cannot be converted**, return a message **"wrong type"**)
* **add(integer:Integer)** – adds the values of the **two instances** and returns the result (if the integer is **not an instance of Integer**, return the message **"number should be an Integer instance"**)

### Examples

|  |
| --- |
| **Test Code** |
| first\_num = Integer(10)  second\_num = Integer.from\_roman("IV")  print(Integer.from\_float("2.6"))  print(Integer.from\_string(2.6))  print(first\_num.add(second\_num)) |
| **Output** |
| value is not a float  wrong type  14 |

## Calculator

Create a class called **Calculator** that has the following **static methods**:

* **add(\*args)** – **sums** all the arguments passed to the function and **returns the result**
* **multiply(\*args)** – **multiplies** all the numbers and **returns the result**
* **divide(\*args)** – **divides** all the numbers and returns the **result**
* **subtract(\*args)** – **subtracts** all the numbers and returns the **result**

### Examples

|  |
| --- |
| **Test Code** |
| print(Calculator.add(5, 10, 4))  print(Calculator.multiply(1, 2, 3, 5))  print(Calculator.divide(100, 2))  print(Calculator.subtract(90, 20, -50, 43, 7)) |
| **Output** |
| 19  30  50.0  70 |

## Hotel Rooms

In a folder called **project** create two files: **hotel.py** and **room.py**

In the **room.py** file create a class called **Room**. Upon **initialization** it should receive a **number** (**int**) and a **capacity** (**int**). It should also have an **attribute** called **guests** (**0** upon initialization) and **is\_taken** (**False** upon initialization). The class should have **2 methods**:

* **take\_room(people)** – if the room is **not taken**, and there is **enough space**, the guests take the room. Otherwise, the method should return **"Room number {number} cannot be taken"**
* **free\_room()** – if the room **is taken**, free it. Otherwise, return **"Room number {number} is not taken"**

In the **hotel.py** file create a class called **Hotel**. Upon initialization it should receive a **name** (**str**). It should also have 2 **more attributes**: **rooms** (empty **list** of rooms) and **guests** (**0** upon initialization). The class should have **5 more methods**:

* **from\_stars(stars\_count)** – creates a new instance with name **"{stars\_count} stars Hotel"**
* **add\_room(room)** – add the room to the list of rooms
* **take\_room(room\_number, people)** – find the room with that **number** and try to **accommodate** the **guests** in the room
* **free\_room(room\_number)** – find the room with that **number** and **free it**
* **print\_status()** – prints information about the hotel in the following format:

**Hotel {name} has {guests} total guests  
Free rooms: {numbers of all free rooms separated by comma and space}**

**Taken rooms: {numbers of all taken rooms separated by comma and space}**

### Examples

|  |
| --- |
| **Test Code** |
| from project.hotel import Hotel  from project.room import Room  hotel = Hotel.from\_stars(5)  first\_room = Room(1, 3)  second\_room = Room(2, 2)  third\_room = Room(3, 1)  hotel.add\_room(first\_room)  hotel.add\_room(second\_room)  hotel.add\_room(third\_room)  hotel.take\_room(1, 4)  hotel.take\_room(1, 2)  hotel.take\_room(3, 1)  hotel.take\_room(3, 1)  hotel.print\_status() |
| **Output** |
| Hotel 5 stars Hotel has 3 total guests  Free rooms: 2  Taken rooms: 1, 3 |