Lab: Class and Static Methods

Problems for in-class lab for the Python OOP Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/2430.

1. 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 (starting from the first one) and returns the **result**
- **`subtract(*args)`** - **subtracts** all the numbers (starting from the first one) 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`

2. Shop

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

- **`small_shop(name: str, type: str)`** - a **new shop with a capacity of 10** should be created
- **`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 shop"`.** If the addition is **not possible**, the following message should be returned **`"Not enough capacity in the shop"`**
- **`remove_item(item_name:str, amount:int)`** - **removes** the given amount from the **item**. On **success**, it should return **`"{amount} {item_name} removed from the shop"`**. **Otherwise**, the method should return **`"Cannot remove {amount} {item_name}"`**
    - If the item **quantity** reaches **0**, the **item** should be **removed from the items' dictionary**.
- **`__repr__()`** - returns a string representation in the format **`"{shop_name} of type {shop_type} with capacity {shop_capacity}"`**

Examples

Test Code

`fresh_shop = Shop("Fresh Shop", "Fruit and Veg", 50)`
`small_shop = Shop.small_shop("Fashion Boutique", "Clothes")`
`print(fresh_shop)`

```
print(small_shop)

print(fresh_shop.add_item("Bananas"))
print(fresh_shop.remove_item("Tomatoes", 2))

print(small_shop.add_item("Jeans"))
print(small_shop.add_item("Jeans"))
print(small_shop.remove_item("Jeans", 2))
print(small_shop.items)
```

Output

```
Fresh Shop of type Fruit and Veg with capacity 50
Fashion Boutique of type Clothes with capacity 10
Bananas added to the shop
Cannot remove 2 Tomatoes
Jeans added to the shop
Jeans added to the shop
2 Jeans removed from the shop
{}
```

3. Integer

Create a class called **`Integer`**. Upon initialization, it should receive a single parameter **`value`** (**int**). It should have **3 additional methods**:

- **`from_float(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"`**)

Examples

Test Code

```
first_num = Integer(10)
print(first_num.value)

second_num = Integer.from_roman("IV")
print(second_num.value)

print(Integer.from_float("2.6"))
print(Integer.from_string(2.6))
```

Output

```
10
4
value is not a float
wrong type
```

4. 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** by default) and **is_taken** (**False** by default). The class should have **2 additional 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** by default). The class should have **5 more methods**:

- **from_stars(stars_count: int)** - creates a new instance with name **"{stars_count} stars Hotel"**
- **add_room(room: Room)** - adds the room to the list of rooms
- **take_room(room_number, people)** - finds the room with that **number** and tries to **accommodate** the **guests** in the room
- **free_room(room_number)** - finds the room with that **number** and tries to **free it**
- **status()** - **returns** 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)

print(hotel.status())
```

Output

```
Hotel 5 stars Hotel has 3 total guests
Free rooms: 2
Taken rooms: 1, 3
```