# Lab: Classes and Objects

Problems for in-class lab for the Python Fundamentals Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/1733.

## 1. Comment

Create a class with the name **"Comment"**. The **__init__** method should accept **3 parameters**:

- **username**
- **content**
- **likes** (optional, 0 by default)

Use the **exact names** for your variables

***Note: there is no input/output for this problem. Test the class yourself and submit only the class***

### Example

| Test Code | Output |
|---|---|
| comment = Comment("user1", "I like this book")<br>print(comment.username)<br>print(comment.content)<br>print(comment.likes) | user1<br>I like this book<br>0 |

### Hint

Create a class named **"Comment"**. Create the **__init__** method and pass it the **three parameters**:

```
01-comment.py
class Comment:
    def __init__(self, username, content, likes=0):
        self.username = username
        self.content = content
        self.likes = likes
```

- we set the initial value of the likes to be 0

Test the class with the provided test code:

```
comment = Comment("user1", "I like this book")
print(comment.username)
print(comment.content)
print(comment.likes)
```

## 2. Party

Create a class **Party** that only has an attribute **people – empty list**. The **__init__** method should **not accept** any **parameters**. You will be given **names** of people (on separate lines) until you receive the command "**End**". Use the created class to solve this problem. After you receive the "**End**" command, print **2 lines**:

- **"Going: {people}" -** the people should be separated by comma and space **", "**.
- **"Total: {total_people_going}"**

***Note: submit all of your code, including the class***

## Example

| Input | Output |
|---|---|
| Peter<br>John<br>Katy<br>End | Going: Peter, John, Katy<br>Total: 3 |
| Sam<br>Eddy<br>Edd<br>Kris<br>End | Going: Sam, Eddy, Edd, Kris<br>Total: 4 |

## Hint

Start by creating the **`class Party`**:

```
02-party.py
1  class Party:
2      def __init__(self):
3          self.people = []
4
```

Create an **instance** of the class:

```
6  party = Party()
```

Create a loop that reads input and adds it to the people until you receive "**End**":

```
8  line = input()
9  while line != "End":
10     party.people.append(line)
11     line = input()
```

Finally, print the result:

```
13 print(f"Going: {', '.join(party.people)}")
14 print(f"Total: {len(party.people)}")
```

# 3. Email

Create **class `Email`**. The **`__init__`** method should receive **`sender`**, **`receiver`** and a **`content`**. It should also have a default set to **`False` attribute** called **`is_sent`**. The class should have **two additional methods**:

- **`send()`** - sets the **`is_sent`** attribute to **`True`**
- **`get_info()`** - returns the following string: "**`{sender} says to {receiver}: {content}. Sent: {is_sent}`**"

You will receive some information (separated by a single space) until you receive the command "**`Stop`**". The first element will be the **sender**, the second one – the **receiver,** and the third one – the **content**. On the **final line,** you will be given the **indices** of the **sent emails** separated by comma and space "**`, `**".

Call the **`send()`** method for the given indices of emails. For each email, call the **`get_info()`** method.

***Note: submit all of your code, including the class***

## Example

| Input | Output |
|---|---|
| Peter John Hi,John<br>John Peter Hi,Peter!<br>Katy Lilly Hello,Lilly<br>Stop<br>0, 2 | Peter says to John: Hi,John. Sent: True<br>John says to Peter: Hi,Peter!. Sent: False<br>Katy says to Lilly: Hello,Lilly. Sent: True |
| Anna, Bella, Hi<br>Sam, Dany, Okey<br>Felix, Mery, Bye<br>Stop<br>0 | Anna, says to Bella,: Hi. Sent: True<br>Sam, says to Dany,: Okey. Sent: False<br>Felix, says to Mery,: Bye. Sent: False |

## Hint

First, we create the Email class with the **`__init__`** method and the **2 other methods**:

```
03-email.py
1  class Email:
2      def __init__(self, sender, receiver, content):
3          self.sender = sender
4          self.receiver = receiver
5          self.content = content
6          self.is_sent = False
7
8      def send(self):
9          self.is_sent = True
10
11     def get_info(self):
12         return f"{self.sender} says to {self.receiver}: {self.content}. Sent: {self.is_sent}"
```

- The **`is_sent`** attribute is not passed to the function. It is set **automatically** to **`False`**
- The **`send()`** method does not accept parameters since it always sets the **`is_sent`** attribute to **`True`**
- The **`get_info()`** method also does not accept parameters. It just returns a **string representation** of the object

We read the input until we receive **`"Stop"`**. Then, for each input, we create an **Email** and add it to the emails' list:

```
15  emails = []
16
17  line = input()
18  while line != "Stop":
19      tokens = line.split(" ")
20      sender = tokens[0]
21      receiver = tokens[1]
22      content = tokens[2]
23      email = Email(sender, receiver, content)
24      emails.append(email)
25      line = input()
```

We read the indices of the sent emails, loop through them, and call the **`send()`** method for each of the emails at those indices:

```
27  send_emails = list(map(lambda x: int(x), input().split(", ")))
28
29  for x in send_emails:
30      emails[x].send()
```

Finally, we print each of the emails:

```
32  for email in emails:
33      print(email.get_info())
```

## 4. Zoo

Create a **class Zoo**. It should have a **class attribute** called **__animals** that stores the **total count of the animals** in the zoo. The **__init__** method should only receive the **name** of the zoo. There you should also create **3 empty lists** (**mammals, fishes, birds**). The class should also have **2 more methods**:

- **add_animal(species, name)** - based on the species, adds the name to the corresponding list
- **get_info(species)** - based on the species returns a string in the following format:
  **"{Species} in {zoo_name}: {names}**
  **Total animals: {total_animals}"**

On the **first line,** you will receive the **name** of the zoo. On the **second line,** you will receive number **n**. On the following **n lines** you will receive animal info in the format: **"{species} {name}"**. **Add** the animal to the **zoo** to the **corresponding list**. The species could be **"mammal"**, **"fish"**, or **"bird"**.

On the **final line,** you will receive a **species**.

At the end, print the info for that species and the total count of animals in the zoo.

### Example

| Input | Output |
|---|---|
| Great Zoo<br>5<br>mammal lion<br>mammal bear<br>fish salmon<br>bird owl<br>mammal tiger<br>mammal | Mammals in Great Zoo: lion, bear, tiger<br>Total animals: 5 |
| Blah<br>1<br>mammal bear<br>mammal | Mammals in Blah: bear<br>Total animals: 1 |

### Hint

Start by creating the class and the __init__ method:

04-zoo.py

```
class Zoo:
    __animals = 0

    def __init__(self, name):
        self.name = name
        self.mammals = []
        self.fishes = []
        self.birds = []
```

- The underscores in front of the animal's attribute is used to express that it is private. It is not meant to be used outside the class.

Then, create the other two methods for adding and getting the info:

```
    def add_animal(self, species, name):
        if species == "mammal":
            self.mammals.append(name)
        elif species == "fish":
            self.fishes.append(name)
        elif species == "bird":
            self.birds.append(name)

        Zoo.__animals += 1

    def get_info(self, species):
        result = ""
        if species == "mammal":
            result += f"Mammals in {self.name}: {', '.join(self.mammals)}\n"
        elif species == "fish":
            result += f"Fishes in {self.name}: {', '.join(self.fishes)}\n"
        elif species == "bird":
            result += f"Birds in {self.name}: {', '.join(self.birds)}\n"

        result += f"Total animals: {Zoo.__animals}"
        return result
```

- We check the species type inside the methods.

Finally, implement the logic for reading the input and printing the result:

```
zoo_name = input()
zoo = Zoo(zoo_name)
count = int(input())

for i in range(count):
    animal = input().split(" ")
    species = animal[0]
    name = animal[1]
    zoo.add_animal(species, name)

info = input()
print(zoo.get_info(info))
```

# 5. Circle

Create a **class `Circle`**. In the **`__init__`** method, the circle should only receive **one parameter** - its **`diameter`**. Create a class attribute called **`__pi`** that is equal to **3.14**. The class should also have the following methods:

- **`calculate_circumference()`** - returns the circumference of the circle
- **`calculate_area()`** - returns the area of the circle
- **`calculate_area_of_sector(angle)`** - gives the central angle in degrees, returns the area that fills the sector

***Notes: Search the formulas on the internet. Name your methods and variables exactly as in the description! Submit only the class. Test your class before submitting it!***

## Example

| Test Code | Output |
|---|---|
| `circle = Circle(10)`<br>`angle = 5`<br><br>`print(f"{circle.calculate_circumference():.2f}")`<br>`print(f"{circle.calculate_area():.2f}")`<br>`print(f"{circle.calculate_area_of_sector(angle):.2f}")` | `31.40`<br>`78.50`<br>`1.09` |

## Hint

First, create the **Circle** class, set the attribute **`__pi`**, and create the **`__init__`** method:

```
1  class Circle:
2      __pi = 3.14
3
4      def __init__(self, diameter):
5          self.diameter = diameter
6          self.radius = diameter / 2
```

- We will be given the **diameter** so that the **radius** will be the **diameter divided** by **2**

Create the first method that calculates the **circumference**:

```
7      def calculate_circumference(self):
8          return Circle.__pi * self.diameter
```

After that, create the method that calculates and returns the **area** of the circle:

```
10     def calculate_area(self):
11         return Circle.__pi * self.radius * self.radius
```

Finally, create the method that calculates the **area of a particular sector**:

```
13     def calculate_area_of_sector(self, angle):
14         return (angle/360) * Circle.__pi * self.radius * self.radius
```

Write some code to **test** your class before you submit it.