# Lab: Classes and Objects

Problems for in-class lab for the [Python Fundamentals Course @SoftUni](https://softuni.bg/trainings/3368/python-fundamentals-may-2021).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/Practice/Index/1733>.

## Comment

Create a class with 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")  print(comment.username)  print(comment.content)  print(comment.likes) | user1  I like this book  0 |

### Hint

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



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

Test the class with the provided test code



## 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***

class Party:  
 def \_\_init\_\_(self):  
 self.people = []  
  
  
party = Party()  
  
line = input()  
  
while line != 'End':  
 party.people.append(line)  
 line = input()  
  
print(f"Going: {', '.join(party.people)}")  
print(f"Total: {len(party.people)}")

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter  John  Katy  End | Going: Peter, John, Katy  Total: 3 |

### Hint

Start by creating the **class Party**:



Create an **instance** of the class:



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



Finally, print the result:



## 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 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***

class Email:  
 def \_\_init\_\_(self, sender, receiver, content):  
 self.sender = sender  
 self.receiver = receiver  
 self.content = content  
 self.is\_sent = False  
  
 def send(self):  
 self.is\_sent = True  
  
 def get\_info(self):  
 return f"{self.sender} says to {self.receiver}: " \  
 f"{self.content}. Sent: {self.is\_sent}"  
  
  
emails = []  
  
data = input()  
  
while data != "Stop":  
 sender, receiver, content = data.split()  
 email = Email(sender, receiver, content)  
 emails.append(email)  
 data = input()  
  
indexes = input().split(', ')  
indexes\_ints = list(map(int, indexes))  
  
for x in indexes\_ints:  
 emails[x].send()  
  
for e in emails:  
 print(e.get\_info())

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter John Hi,John  John Peter Hi,Peter!  Katy Lilly Hello,Lilly  Stop  0, 2 | Peter says to John: Hi,John. Sent: True  John says to Peter: Hi,Peter!. Sent: False  Katy says to Lilly: Hello,Lilly. Sent: True |

### Hint

First, we create the Email class with the **\_\_init\_\_** method and the **2 other methods**:



* 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:



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



Finally, we print each of the emails:



## 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 next **n lines** you will receive animal info in the format: **"{species} {name}**". **Add** the animal to the **zoo** to the **corresponding list**. The **"species"** command will 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.

class Zoo:  
 \_\_animals = 0  
  
 def \_\_init\_\_(self, name):  
 self.name = name  
 self.mammals = []  
 self.fishes = []  
 self.birds = []  
  
 def get\_animals\_count(self):  
 return self.\_\_animals  
  
 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)  
  
 self.\_\_animals += 1  
  
 def get\_info(self, species):  
 if species == "mammal":  
 return f"Mammals in {self.name}: {', '.join(self.mammals)}"  
 elif species == "fish":  
 return f"Fishes in {self.name}: {', '.join(self.fishes)}"  
 elif species == "bird":  
 return f"Birds in {self.name}: {', '.join(self.birds)}"  
  
  
name = input()  
n = int(input())  
zoo = Zoo(name)  
  
for \_ in range(n):  
 species, name = input().split()  
 zoo.add\_animal(species, name)  
  
searched\_species = input()  
print(zoo.get\_info(searched\_species))  
print(f"Total animals: {zoo.get\_animals\_count()}")

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Great Zoo  5  mammal lion  mammal bear  fish salmon  bird owl  mammal tiger  mammal | Mammals in Great Zoo: lion, bear, tiger  Total animals: 5 |

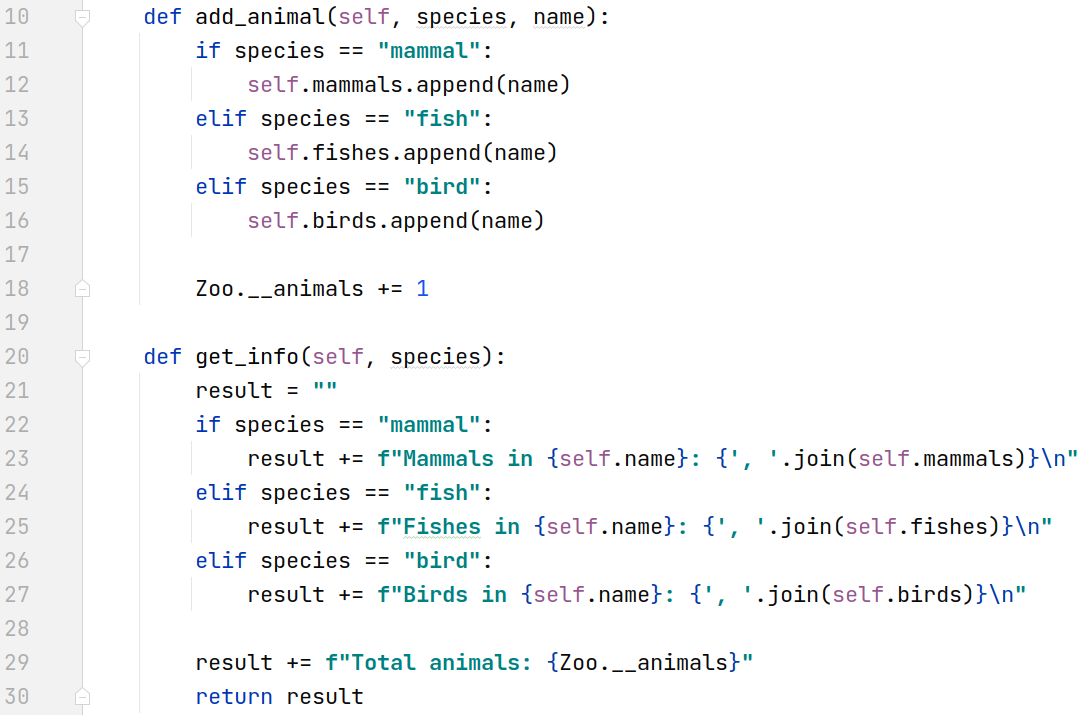
### Hint

Start by creating the class and the \_\_init\_\_ method:



* The underscores in front of the animals attribute is used to express that it is private and is not meant to be used outside the class.

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



* We check the species type inside the methods.

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



## 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

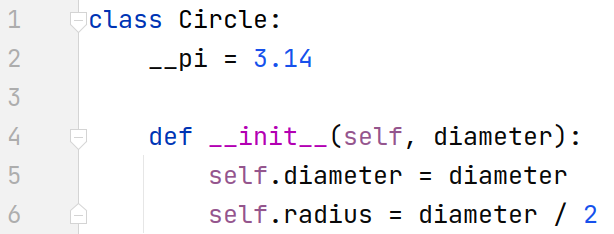
***Notes: Search the formulas in the internet. Name your methods and variables exactly as in the description! Submit only the class. Test your class before submitting!***

### Example

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| circle = Circle(10)  angle = 5  print(f"{circle.calculate\_circumference():.2f}")  print(f"{circle.calculate\_area():.2f}")  print(f"{circle.calculate\_area\_of\_sector(angle):.2f}") | 31.40  78.50  1.09 |

### Hint

First, create the **Circle** class, set the attribute **\_\_pi** and create the **\_\_init\_\_** method:



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

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



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



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



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

class Circle:  
 \_\_pi = 3.14  
  
 def \_\_init\_\_(self, diameter):  
 self.diameter = diameter  
 self.radius = diameter / 2  
  
 def calculate\_circumference(self):  
 return Circle.\_\_pi \* self.diameter  
  
 def calculate\_area(self):  
 return Circle.\_\_pi \* self.radius \* self.radius  
  
 def calculate\_area\_of\_sector(self, angle):  
 return (angle/360) \* Circle.\_\_pi \* self.radius \* self.radius

# Submit only the class in Judge (judge.softuni.bg)  
circle = Circle(10)  
angle = 5  
  
print(f"{circle.calculate\_circumference():.2f}")  
print(f"{circle.calculate\_area():.2f}")  
print(f"{circle.calculate\_area\_of\_sector(angle):.2f}")