App Development Project
Week 7, Lesson 1

👀 **Looking Forward** 👀
At the end of this lesson, you will be able to: • Define an object. • Identify the objects required by the project and their relationships. • Create a **User class** (model) to aid in storage, retrieval and usage of User **data**.

1. Activity 1: What is an Object?

THINK ABOUT IT

Let us recall the demo flask project from Week 1, Lesson 1. Are you able to identify the classes and their corresponding information (attributes) and functionality (methods)?

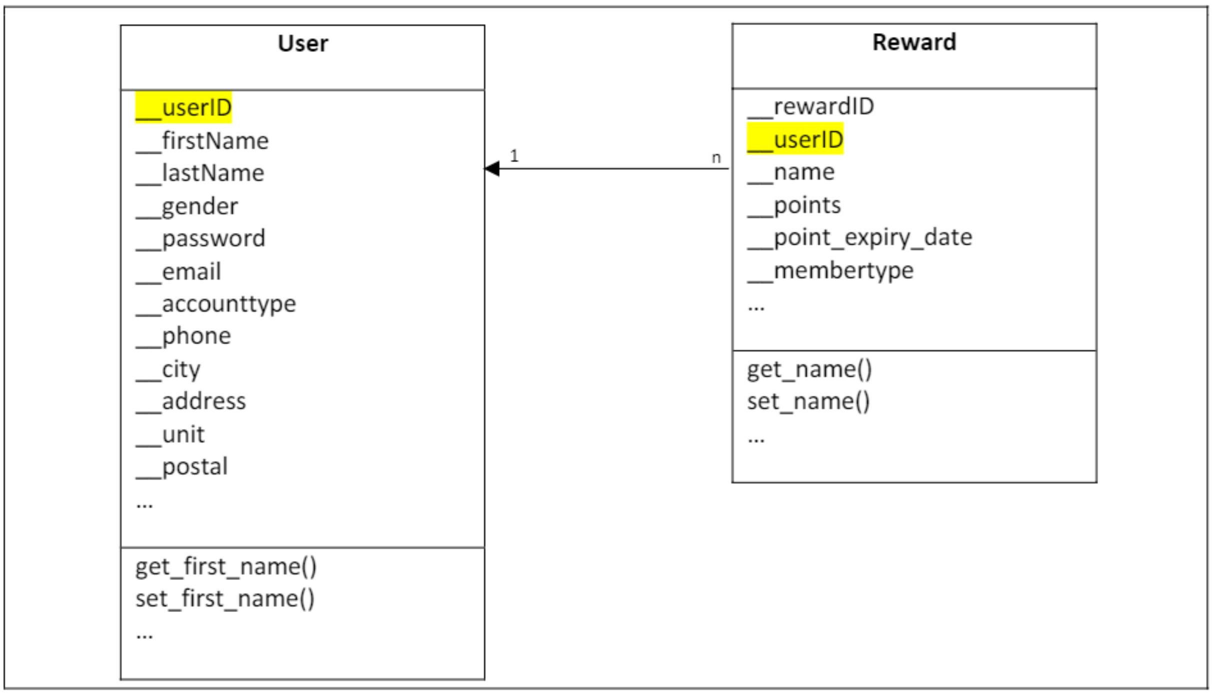
One example would be a User class as defined below. Remember that classes are blueprints that specify the data attributes and methods from which we create objects. From the User class, we can then create for example admin, customer, vendor etc. user objects.

User
__userID __firstName __lastName __gender __password __email __accounttype __phone __city __address __unit __postal …

get_user_id()
set_user_id()
…

How many more classes can you think of?

Next, there are relationships between the classes. For example, a customer object may have many reward objects associated with it, perhaps in the form of discount vouchers. We can then use __user_id to identify who this reward object belongs to whenever a new reward object is created.


User
__userID
__firstName
__lastName
__gender
__password
__email
__accounttype
__phone
__city
__address
__unit
__postal
...
get_first_name()
set_first_name()
...
1
n
Reward
__rewardID
__userID
__name
__points
__point_expiry_date
__membertype
...
get_name()
set_name()
...

2. Activity 2: Discuss on Project Idea and Objects

🤔 THINK ABOUT IT 🤔

Within your teams, let's now think about the project that you are going to do. As you define your classes, there are 2 main questions that you need to ask yourselves:

1. What kind of objects will you be creating from these classes in your project?
2. What is the relationship between these objects?

You are encouraged to draw the class diagram for your project to provide clarity on how your data will eventually be stored/persisted, and to establish a common understanding among your team members. You will learn more about how your data is persisted in the next activity.

More information on class diagrams here:

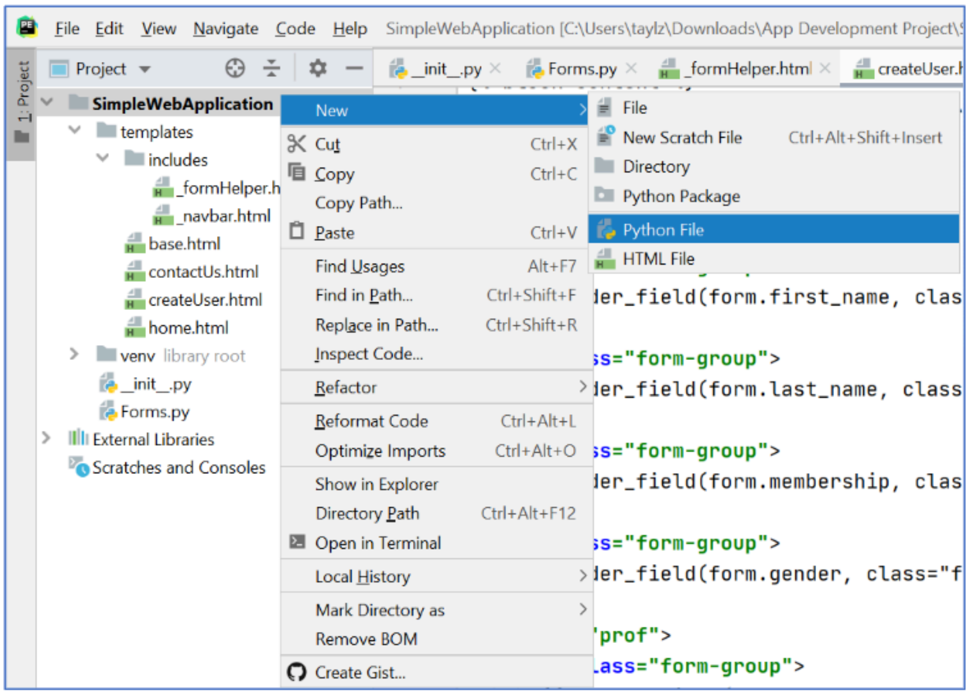
- https://www.tutorialspoint.com/uml/uml_class_diagram.htm

3. Activity 3: Start the Project

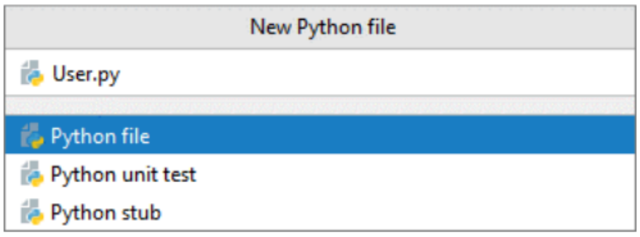
3.1 Create Classes for Your SimpleWebApplication

3.1.1 Create a New User Class

Step 1: Right-click on the **SimpleWebApplication** folder and select **New > Python File** to create a new Python File called **User.py**.



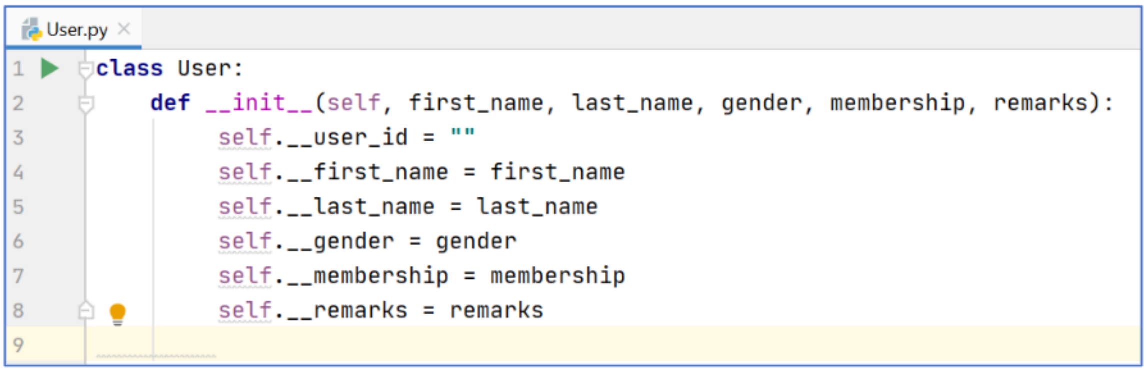
```
der_field(form.first_name, clas
ss="form-group">
der_field(form.last_name, class
ss="form-group">
der_field(form.membership, clas
ss="form-group">
der_field(form.gender, class="f
'prof">
.ass="form-group">
```



Step 2: Create a **User class** with the following **private** attributes:

1. __user_id
2. __first_name
3. __last_name
4. __gender
5. __membership
6. __remarks

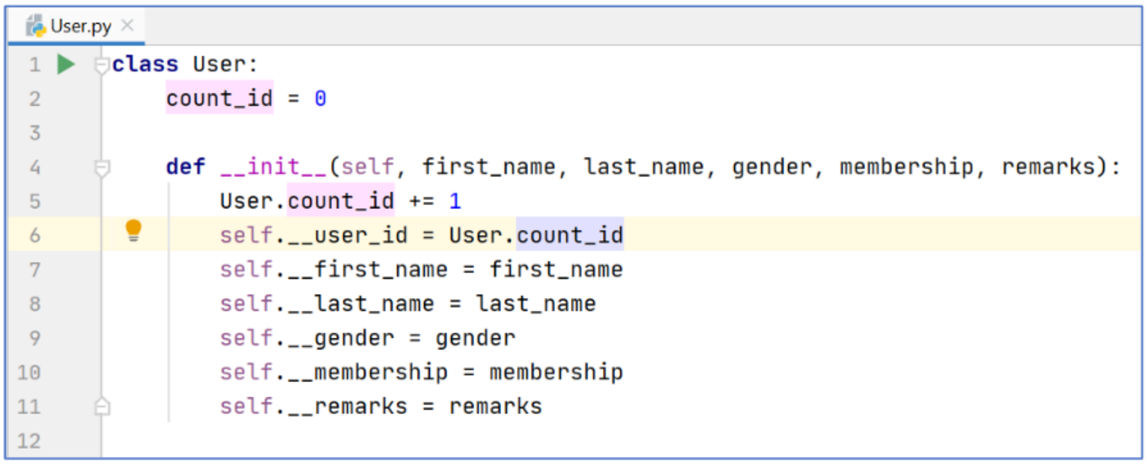
The **initializer** needs to take in **parameters** for each of the private attributes except **__user_id**.



User.py

```python
class User:
    def __init__(self, first_name, last_name, gender, membership, remarks):
        self.__user_id = ""
        self.__first_name = first_name
        self.__last_name = last_name
        self.__gender = gender
        self.__membership = membership
        self.__remarks = remarks

```

Step 3: Create a **class attribute** called **count_id** to be used as a counter. Increment the **count_id** class attribute and use it to initialize the **__user_id** data attribute.



User.py

```python
class User:
    count_id = 0

    def __init__(self, first_name, last_name, gender, membership, remarks):
        User.count_id += 1
        self.__user_id = User.count_id
        self.__first_name = first_name
        self.__last_name = last_name
        self.__gender = gender
        self.__membership = membership
        self.__remarks = remarks

```

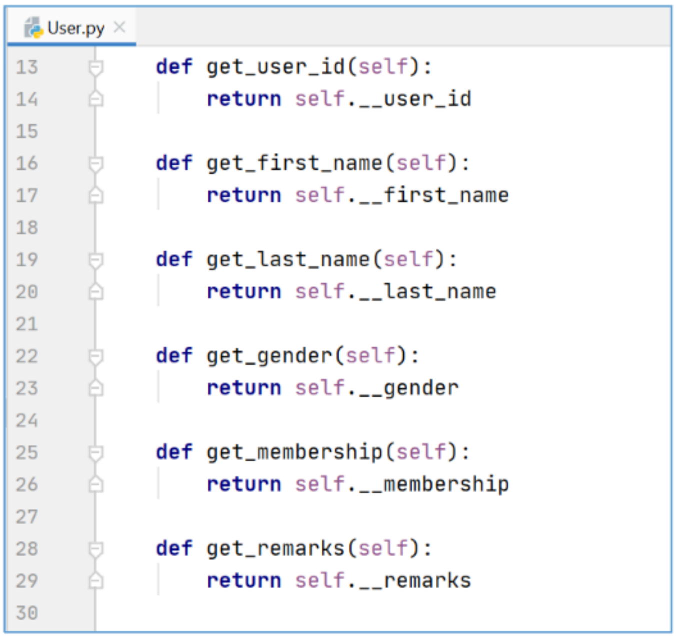
THINK ABOUT IT

Notice that the User class' initializer method does not require **user_id** be taken in as a parameter. This is because, **count_id** will be used to create a **primitive** form of **auto-increment** for the **__user_id**.

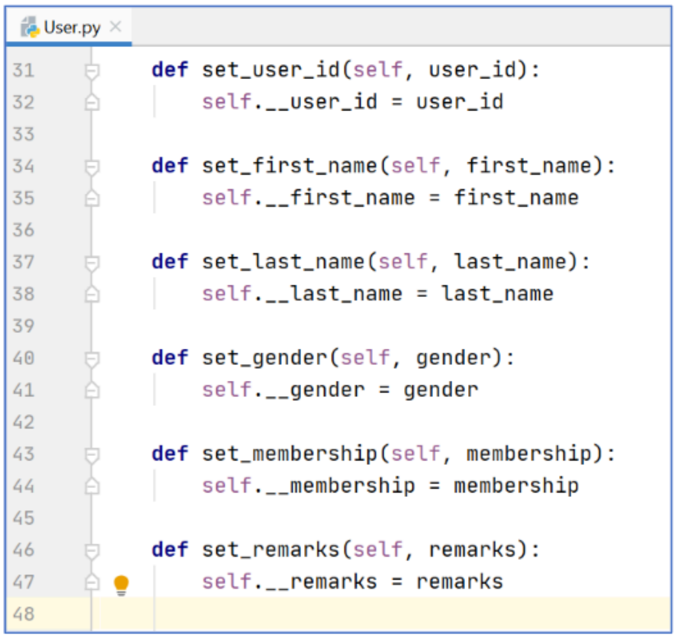
One obvious limitation it has is that every time you **restart** the web application, the **count_id resets to 0**. Once the **count_id** resets, the next newly created User's **__user_id** will start from **1** again and **overwrite** any User that previously had **__user_id == 1**.

Can you think of a better alternative?

Step 4: Create the **accessor** and **mutator methods** for each of the private attributes.


User.py

```python
    def get_user_id(self):
        return self.__user_id

    def get_first_name(self):
        return self.__first_name

    def get_last_name(self):
        return self.__last_name

    def get_gender(self):
        return self.__gender

    def get_membership(self):
        return self.__membership

    def get_remarks(self):
        return self.__remarks

```


User.py

```python
    def set_user_id(self, user_id):
        self.__user_id = user_id

    def set_first_name(self, first_name):
        self.__first_name = first_name

    def set_last_name(self, last_name):
        self.__last_name = last_name

    def set_gender(self, gender):
        self.__gender = gender

    def set_membership(self, membership):
        self.__membership = membership

    def set_remarks(self, remarks):
        self.__remarks = remarks

```

~ End ~