PD LAB
ASSIGNMENT - 10

Name: Raunak Thanawala
Registration Number: 231070051
Branch: Computer Engineering
Batch: 3

Aim:-

To study Classes and Objects in python

Theory:-

1. **Classes and Objects**
    - Class:
        - A class is a blueprint or template for creating objects.
        - It defines the properties and behaviors that the objects created from it will have.
    - Object:
        - An instance of a class.

    - An object is a specific realization of a class, containing actual values for the attributes defined by the class.

2. **Attributes**
    - Class Attributes:
        - Variables defined within a class but outside any method.
        - They are shared by all instances of the class.
    - Instance Attributes:
        - Variables that are specific to each instance of a class.
        - Typically, they are defined within the __init__ method, which is called when an object is created.
3. **Methods**
    - Instance Methods:
        - Functions defined inside a class that operate on instances of the class.
        - They are usually used to manipulate instance attributes.
    - Class Methods:
        - Defined using the @classmethod decorator.

        - They receive the class itself as the first parameter (cls) and are used to manipulate class attributes or create alternative constructors.
    - Static Methods:
        - Defined using the @staticmethod decorator.
        - They do not take self or cls as a parameter and act as utility functions within a class.

4. Constructors

- Initializer (__init__):
    - A special method called when an object is instantiated.
    - Used to initialize instance attributes.

5. Destructors

- Destructor (__del__):
    - A special method called when an object is about to be destroyed.
    - Used for cleanup tasks before the object is removed from memory.

6. Encapsulation

- The concept of restricting access to certain data and methods within a class.

- In Python, this is achieved by prefixing attributes or methods with a single or double underscore, marking them as protected or private, respectively.

7. **Inheritance**

- A mechanism allowing a new class to inherit attributes and methods from an existing class.
- This enables code reuse and establishes a relationship between classes (such as a parent-child or superclass-subclass relationship).

8. **Polymorphism**

- The ability to define methods in different ways across different classes or instances.
- It allows objects of different types to be treated uniformly based on shared methods or interfaces.

9. **Abstraction**

- Hiding implementation details and exposing only essential information and behaviors.
- Abstraction is achieved by using abstract classes or interfaces that define common behaviors without specifying implementation.

10. **Magic Methods (Dunder Methods)**

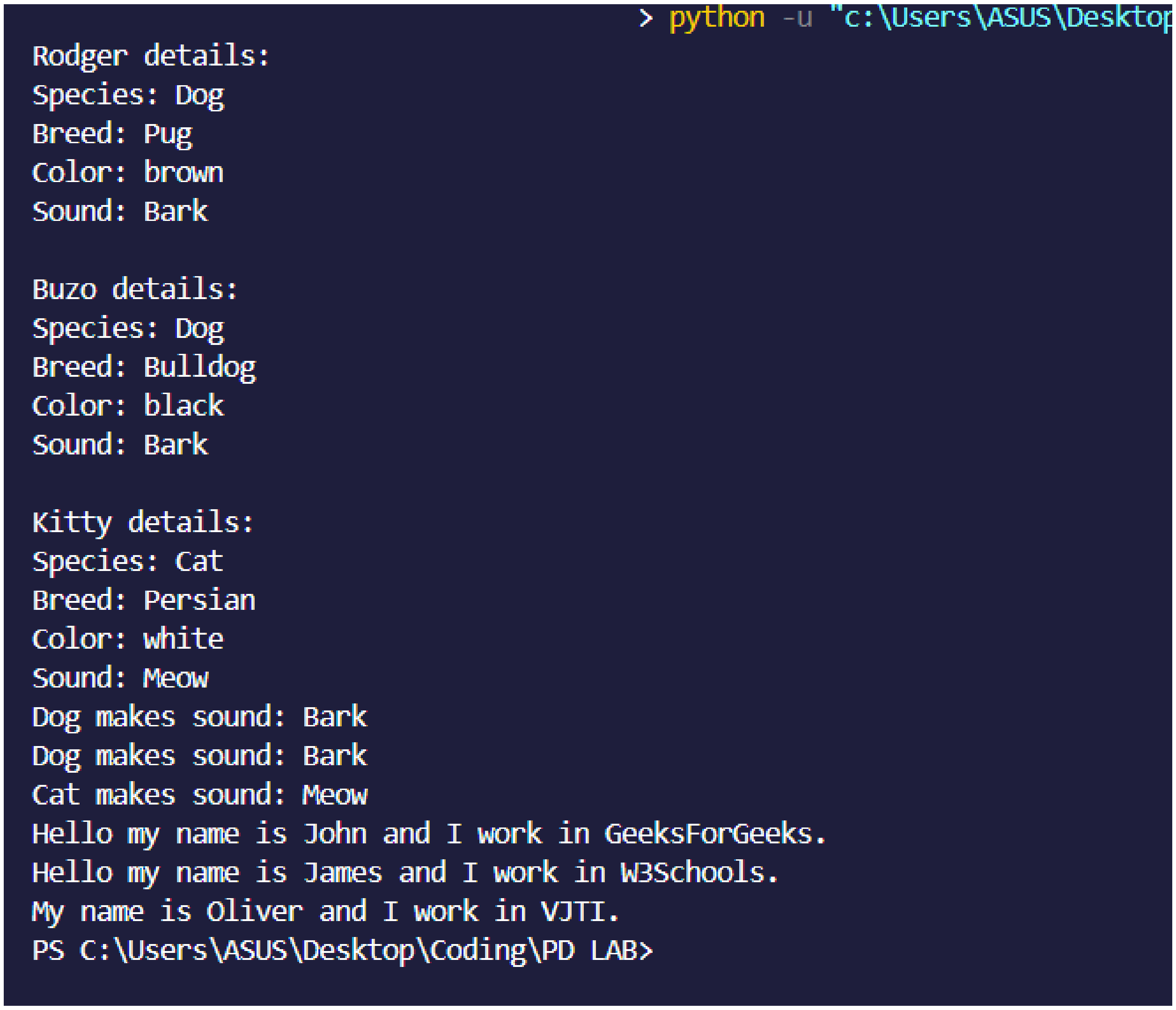
- Special methods surrounded by double underscores, such as __init__, __str__, and __len__.
- These methods allow customization of object behavior in Python, enabling operators and functions to work with user-defined objects.

Code:

```python
class Pet:
    def __init__(self, species):
        self.species = species

    def make_sound(self):
        return "Generic Sound"

class Dog(Pet):
    def __init__(self, breed, color):
        super().__init__("Dog")
        self.breed = breed
        self.color = color

    def make_sound(self):
        return "Bark"

class Cat(Pet):
    def __init__(self, breed, color):
        super().__init__("Cat")
        self.breed = breed
        self.color = color

    def make_sound(self):
        return "Meow"

Rodger = Dog("Pug", "brown")
Buzo = Dog("Bulldog", "black")
```

```python
Kitty = Cat("Persian", "white")

print("Rodger details: ")
print("Species:", Rodger.species)
print("Breed:", Rodger.breed)
print("Color:", Rodger.color)
print("Sound:", Rodger.make_sound())

print("\nBuzo details: ")
print("Species:", Buzo.species)
print("Breed:", Buzo.breed)
print("Color:", Buzo.color)
print("Sound:", Buzo.make_sound())

print("\nKitty details: ")
print("Species:", Kitty.species)
print("Breed:", Kitty.breed)
print("Color:", Kitty.color)
print("Sound:", Kitty.make_sound())

pets = [Rodger, Buzo, Kitty]
for pet in pets:
    print(f"{pet.species} makes sound: {pet.make_sound()}")


class GFG:
    def __init__(self, name, company):
        self.name = name
        self.company = company

    def show(self):
        print("Hello my name is " + self.name+" and I" + " work in 
"+self.company+".")

obj = GFG("John", "GeeksForGeeks")
obj.show()

class Geek:
    def __init__(somename, name, company):
        somename.name = name
        somename.company = company

    def show(somename):
        print("Hello my name is " + somename.name + " and I work in
"+somename.company+".")

obj = Geek("James", "W3Schools")
obj.show()

class MyClass:
    pass

class Strin:
    def __init__ (self, name, college):
        self.name = name
        self.college = college

    def __str__ (self):
        return f"My name is {self.name} and I work in {self.college}."

obj = Strin("Oliver" , "VJTI")
print(obj)
```

OUTPUT:



```
                                        > python -u "c:\Users\ASUS\Desktop
Rodger details:
Species: Dog
Breed: Pug
Color: brown
Sound: Bark

Buzo details:
Species: Dog
Breed: Bulldog
Color: black
Sound: Bark

Kitty details:
Species: Cat
Breed: Persian
Color: white
Sound: Meow
Dog makes sound: Bark
Dog makes sound: Bark
Cat makes sound: Meow
Hello my name is John and I work in GeeksForGeeks.
Hello my name is James and I work in W3Schools.
My name is Oliver and I work in VJTI.
PS C:\Users\ASUS\Desktop\Coding\PD LAB>
```

CONCLUSION:

Thus we have written a program that shows classes and objects illustrated in python.
We have also given a brief explanation about the classes and objects in python.