CLASS and INHERITANCE

OUTLINE

- Class and Inheritance
- Introduction to database

Introduction to OOPs in Python

Python is a multi-paradigm programming language. Meaning, it supports different programming approach.

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

- attributes
- behaviour

Parrot for example has,

- name, age, color are attributes
- singing, dancing are behaviour

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

Introduction to OOPs in Python

In Python, the concept of OOP follows some basic principles:

Inheritance	A process of using details from a new class without modifying existing class.
Encapsulation	Hiding the private details of a class from other objects.
Polymorphism	A concept of using common operation in different ways for different data input.

Classes

A class is a blueprint for the object.

We can think of class as an sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, parrot is an object.

We define a class using the keyword class

For example :

```
class Parrot:

    pass
```

Here, we use class keyword to define an empty class Parrot. From class, we construct instances. An instance is a specific object created from a particular class.

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint, template or plan for the object.

The example for object of parrot class can be:

```
obj = Parrot()
```

Here, obj is object of class Parrot. Suppose we have details of parrot. Now, we are going to show how to build the class and objects of parrot.

EXAMPLE

Creating Class and Object in Python

```
class Parrot:
# class attribute
   species = "bird"
# instance attribute
   def __init__(self, name, age):
                self.name = name
                self.age = age
# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)
```

EXAMPLE

Creating Class and Object in Python

```
# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))
# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

Method

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

EXAMPLE

Creating Methods in Python

```
class Parrot:
    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age
    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)
    def dance(self):
        return "{} is now dancing".format(self.name)
    # instantiate the object
blu = Parrot("Blu", 10)
    # call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Constructors in PYTHON

Class functions that begins with double underscore (__) are called special functions as they have special meaning.

Of one particular interest is the __init__() function. This special function gets called whenever a new object of that class is instantiated.

This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.

EXAMPLE

Constructor

```
def __init__(self,name,num,init):
    self.name = name
    self.num = num
    self.bal = init
```

Classes
INSTANTIATION

Class *instantiation* uses function notation. Just pretend that the class object is a parameter less function that returns a new instance of the class. For example (assuming the above class):

```
x = MyClass()
```

creates a new *instance* of the class and assigns this object to the local variable x.

INHERITANCE

Inheritance enable us to define a class that takes all the functionality from parent class and allows us to add more.

It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

EXAMPLE

Inheritance

```
# parent class
class Bird:
    def __init__(self):
        print("Bird is ready")
    def whoisThis(self):
        print("Bird")
    def swim(self):
        print("Swim faster")
```

EXAMPLE

Inheritance

```
# child class
class Penguin(Bird):
    def __init__(self):
        # call super() function
        super().__init__()
        print("Penguin is ready")
    def whoisThis(self):
        print("Penguin")
    def run(self):
        print("Run faster")
peggy = Penguin()
peggy.whoisThis()
peggy.swim()
peggy.run()
```

Multiple INHERITANCE

Python supports a form of multiple inheritance as well. A class definition with multiple base classes looks like this:

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    . . .
    <statement-N>
```

ENCAPSULATION

Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single " _ " or double " __".

EXAMPLE

Data Encapsulation in Python

```
class Computer:
    def __init__(self):
        self.__maxprice = 900
    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))
    def setMaxPrice(self, price):
        self.__maxprice = price
```

EXAMPLE

Data Encapsulation in Python

```
c = Computer()
c.sell()
# change the price
c.__maxprice = 1000
c.sell()
# using setter function
c.setMaxPrice(1000)
c.sell()
```

POLYMORPHISM

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.

EXAMPLE

Polymorphism in Python

```
class Parrot:
    def fly(self):
        print("Parrot can fly")
    def swim(self):
        print("Parrot can't swim")

class Penguin:
    def fly(self):
        print("Penguin can't fly")
    def swim(self):
        print("Penguin can swim")
```

EXAMPLE

Polymorphism in Python

```
# common interface
def flying_test(bird):
    bird.fly()
#instantiate objects
obj = Parrot()
obj2 = Penguin()
# passing the object
flying_test(obj)
flying_test(obj2)
```