सी-डैक CDAC अॅक्टस acts
Authorized Training Centre

Object Oriented Programming with Python

Object-Oriented Programming

- Object-Oriented Programming(OOP) is an approach of looking at a problem using models which are organized around the real-world concepts.
- The fundamental construct of Object-Oriented Programming (OOP) is object which combines both data structure and behavior as a single entity.

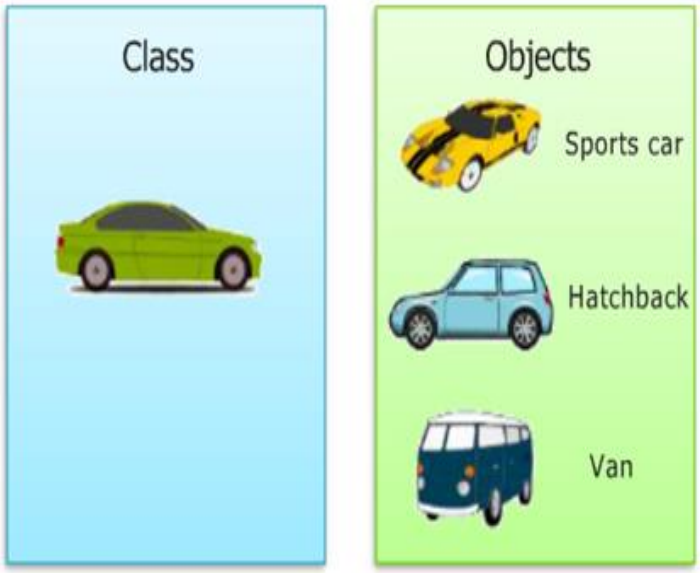
Features of object-oriented programming are:-

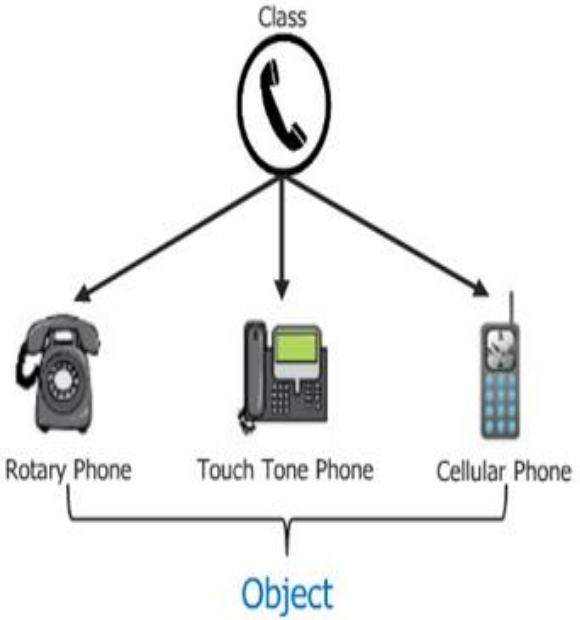
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Classes & Objects

→ Class is a blueprint used to create objects having same property or attribute as its class

→ An Object is an instance of class which contains variables and methods





Classes & Objects

- Class is a template definition of methods and variables in a particular kind of **object.**
- A class describes the abstract characteristics of a **real life** thing
- There can be **instances** of classes ,an instance is an object of a class created at runtime.
- The set of values of the attributes of a particular object is called its **states.**
- The set of methods of particular object is called its **behaviors.**

Classes & Objects

- Compared with other programming languages, Python's class mechanism adds classes with a minimum of new syntax and semantics.
- It is a mixture of the class mechanisms found in C++ and Modula-3.
- All Classes in python are derived from **object** class
- A class can consists of class and instance variables, methods,constructors etc.

Syntax:-

```
class <class-name>:
        "docstring"
         <class suite>
```

Defining Empty Class:

```
class Student(object):
        "Empty Student Class"
         pass
```

Creating Object:

```
obj=Student( )
print( obj )
```

Public, Protected and Private Attributes

→ Private attributes can only be accessed inside of the class definition

→ Protected (restricted) attributes should only be used under certain conditions

→ Public attributes can and should be freely used

Naming	Type	Meaning
name	Public	These attributes can be freely used inside or outside of a class definition
_name	Protected	Protected attributes should not be used outside of the class definition, unless inside of a subclass definition
__name	Private	This kind of attribute is inaccessible and invisible. It's neither possible to read nor to write those attributes, except inside of the class definition itself

Classes & Objects in Python

#Defining Constructor:

```
def __init__(self):
        print("constructor called")
```

#Defining Methods:

```
def show(self):
        print("Hello from Show")
```

#Defining Static Methods:

```
@staticmethod
def display():
   print(" This is static in python ")
```

#Destructor

```
def __del__(self):
   class_name = self.__class__.__name__
   print(class_name, "destroyed")
```

Inheritance in Python

Inheritance

- Inheritance is the mechanism of acquiring some attributes of any existing class type into a new class type.
- One of the key concepts of OOP's.
- Establishes a hierarchical relationship among classes.
- Establishes a superclass/subclass relationship.
- Establishes **"is a"** relationships.

Benefits:

- ✓ Reusability of code.
- ✓ Put code in one class, use it in all the subclasses.
- ✓ Write general purpose code designed for a supertype that works for all subtypes.
- ✓ A superclass defines a general set of functionality, whereas subclasses define functionalities specific to them.

Inheritance in Python

Syntax:

```
class DerivedClassName(BaseClassName):
                    <statement-1>
                     . . .
                    <statement-N>
```

- The name BaseClassName must be defined in a scope containing the derived class definition.
- In place of a base class name, other arbitrary expressions are also allowed. This can be useful, for example, when the base class is defined in another module.

Example:

```
class DerivedClassName(modulename.BaseClassName):
                        pass
```

Abstract classes in Python

- For making a class abstract first you have to import **abc** package for **Abstract base Class** in Python.

Example:

```
from abc import ABC, ABCMeta, abstractmethod
class  MyBase(ABC):
        __metaclass__ = ABCMeta
        @abstractmethod
        def show(self):
              pass

class Derive(MyBase):
   def show(self):
      print("Show from derive….")

derive=Derive()
derive.show()
#base=MyBase()     #error can’t instantiate
```

Operator Overloading in Python

```
class Point:
        def __init__(self, x = 0, y = 0):
                self.x = x
                self.y = y
         def __str__(self):
                  return "({0},{1})".format(self.x,self.y)
        def __add__(self,other):
                x = self.x + other.x
                y = self.y + other.y
                return Point(x,y)
p1=Point(2,3)
p2=Point(2,3)
print(p1+p2)
#  (4,6)
```

THANK YOU!!