Scripting Language

UNIT-V

K.RaviKanth, Asst.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

- Object Oriented Features
- Classes & Objects
- Inheritance
- Building a Class, Adding another Class
- Using Inherited Methods

python
Programming Language

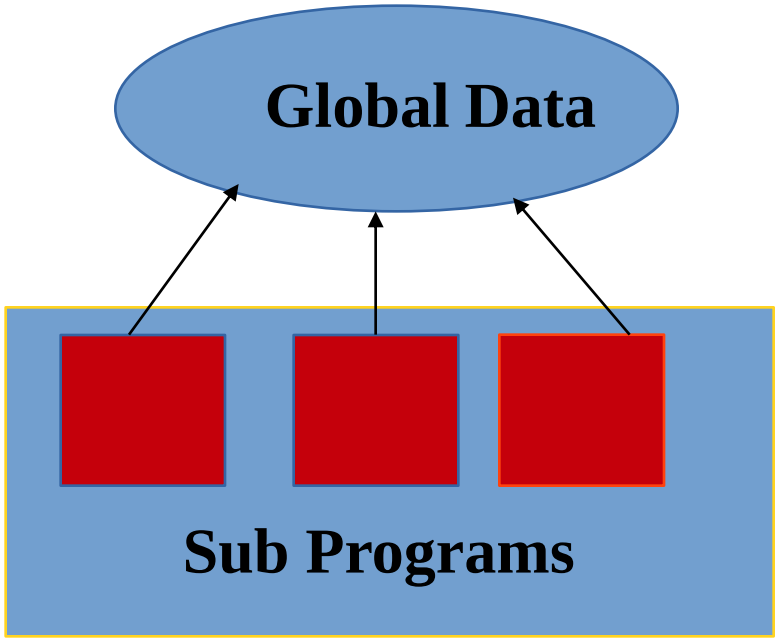
FIVE main kinds of Programming styles with the different types of Abstraction they employ:

Programming Style	Abstraction Employed
Monolithic	Emphasizes on finding a solution
Procedure Oriented	Lays stress on Algorithms (Computation)
Structured	Focuses on Modules (Functions or Sub-Programs)
Object Oriented	**Classes & Objects (Wide range of Architecture framework together)**
Logic Oriented	Goals (Often expressed in Predicate Calculus)
Rule Oriented	If-Then-Else Rules (Design of Knowledge Base)
Constraint Oriented	Utilizes Invariant Relationship (Complex Systems)

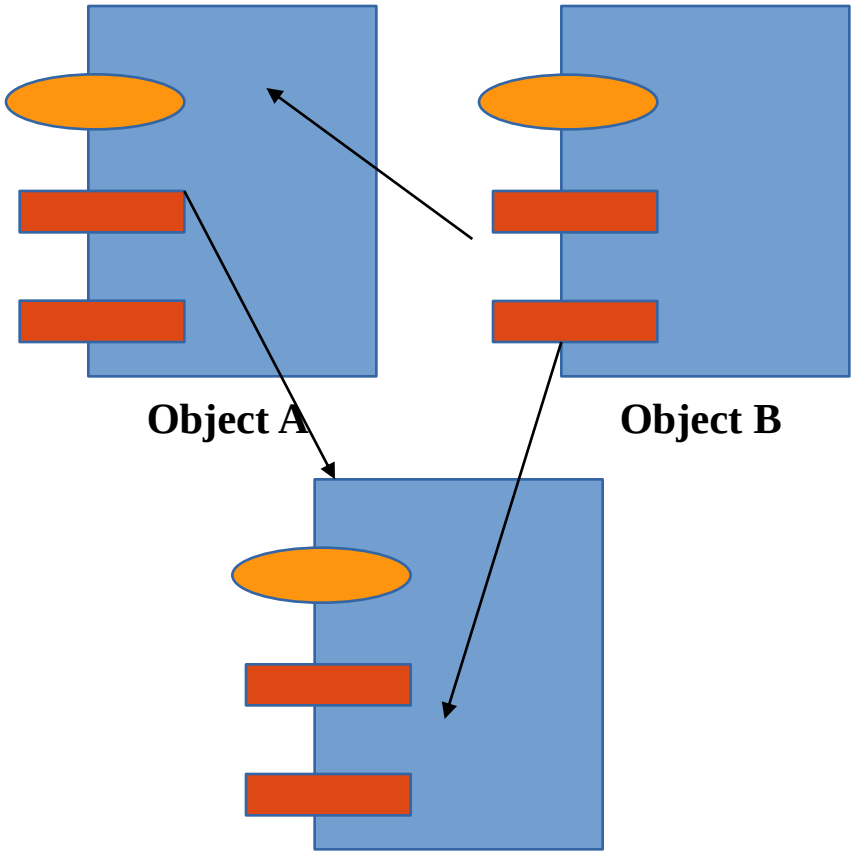
OOP is much more similar to the way the real world works; it is analogous to the human brain. Each program is made up of many entities called objects.

Pictorial Representation

Structured Programming



Object Oriented Programming



Need for Object Oriented Approach

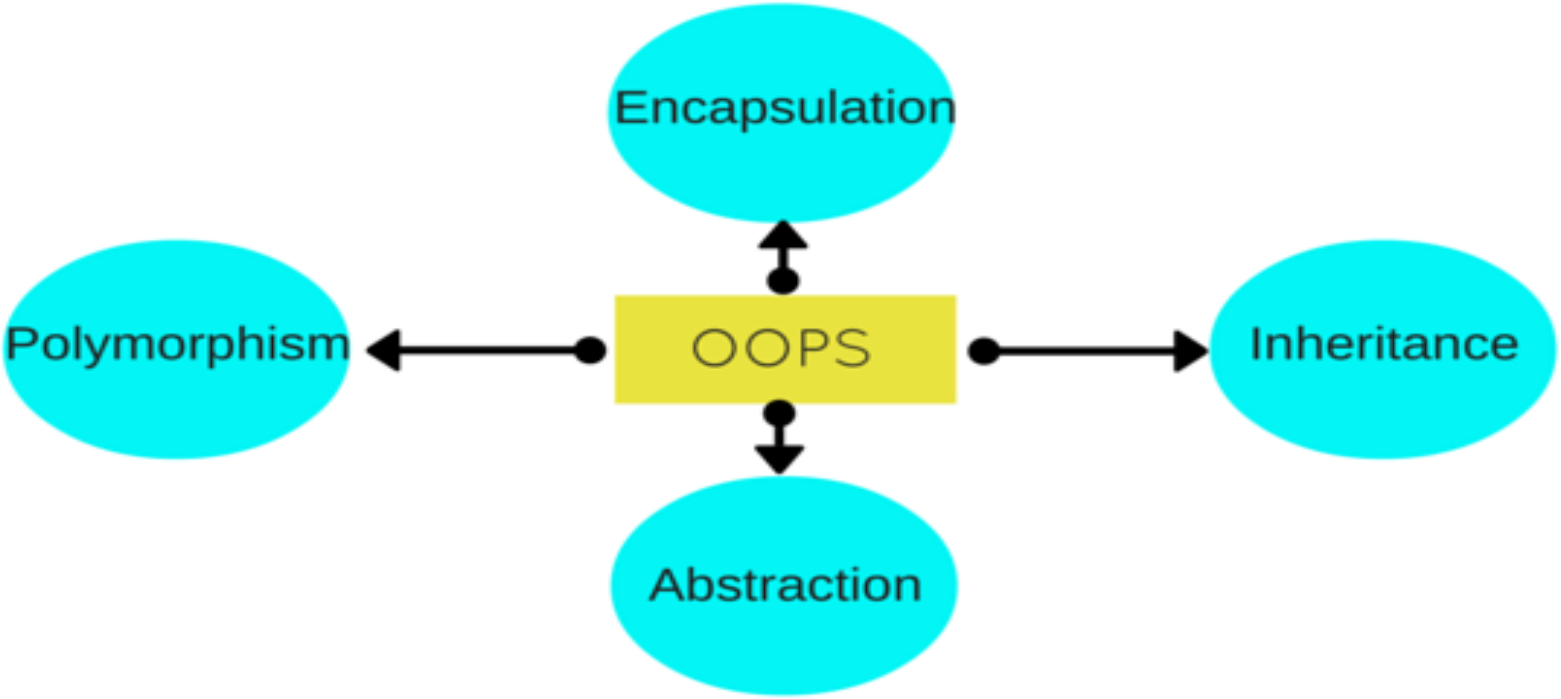
- To face the Challenges in Developing a business application
  - Integration of Modules/Applications
  - Extensibility of existing code
  - High level of flexibility
  - Illusion of simplicity
- If these challenges are not addressed it may lead to **SOFTWARE CRISIS**
- The striking Features needed in the business application to meet these challenges
  - Modularity
  - Data hiding
  - Extendibility
  - Reusability
  - Interoperability
  - Security
- These Challenges can be addressed using **OOA**

Major Features of Object Oriented Programming :

- The object oriented language must support mechanisms to define, create, store, manipulate objects and allow communication between objects are:
  - Classes
  - Objects
  - Methods
  - Encapsulation
  - Data Abstraction
  - Inheritance
  - Polymorphism
  - Reusability
  - Message Passing
  - Constructor
  - Garbage Collection

Basic concepts of Object Oriented Programming

- Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.



OOPs Terminologies

- **Class** – A description of what data is accessible through a particular kind of object, and how that data may be accessed

  OR

  Classes define the Properties and Behaviour of objects. Therefore a class is a collection of objects

  Class is not REAL. Class cannot be executed. Class is a Skeleton of a Program

  A class can be executed by alloting some area called as HEAP.

  Ex: Application Form is a class

- **Object** – Anything that provides a way to locate, access, modify and secure data

  OR

  An object is an instance of a class which can be uniquely identified by its name. Every objet has a state which is given by its attributes at a particular time

  Object is similar to variable, where a variable contains a Small Area , whereas an Object contains a Large Area

- **Note: While a class is a logical structure, an object is a physical**

- **Method** – The means by which an object's data is accessed, modified or processed or A method is a function associated with a class. It defines the operation that the object can execute when it receives a message
- **Abstraction** – Focusing on "what' should be hidden
- **Encapsulation** - Separate what from the how
- **Inheritance** – The way in which existing classes of object can be upgraded to provide additional data or methods
- **Polymorphism** – The way that distinct objects can respond differently to the same message, depending on the class they belong to
- **Message Passing-** Two objects can communicate with each other through message passing mechanisms. An object asks another object to invoke one of its method by sending it a message
- **Constructor –** Constructor is a method calling to allocate Heap area
- **Reusability –** Reusability means developing codes that can be reused either in the same program or in different programs. Reusability in python is achieved by Inheritance and Polymorphism
- **self -** self represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class in python
- **_ _init_ _ ( )-** "__init__( )" is a reserved method in python classes. It is known as a constructor in object oriented concepts. This method called when an object is created from the class and it allow the class to initialize the attributes of a class
- **_ _del_ _( )** – method is automatically called when an object is going out of scope. This is the time when an object is no longer be used and its occupied resources are returned back to the system. Which is similar to destructors in C++ and Java

Benefits or Merits of OOP Language

1. It Leads to development of Smaller but Stable subsystems
2. The subsystems are flexible to change
3. Reduces the risk factor in building large systems as they are build incrementally from subsystems which are stable
4. Through inheritance, we can **eliminate redundant code** and extend the use of existing classes.
5. The principle of data hiding helps the programmer to build **secure** programs.
6. It is easy to **partition** the work in a project based on objects.
7. Object oriented system easily **upgraded** from small to large systems.
8. Software complexity can be easily managed
9. OOPS support code reusability to a great extent
10. OOPS are Realistic, where we can think of Real Time Objects

Hence Object Orientation is suitable for development extremely complex business systems

Applications of OOP

- Object Oriented Technology has changed the way of thinking , analyzing,planning and implementing the software.
- Applications developed using OOT are not only efficient but also easy to upgrade.
- **Some of these areas includes the following:**
- Designing User interfaces such as work screens,menus,windows son on
- Real-time systems
- Simulation and modelling
- Compiler design
- Client Server system
- Object-oriented databases & Object Oriented distributed database
- Artificial intelligence
- Neural networks and parallel programming
- Decision control systems and Office automation systems
- Computer-aided design (CAD)
- Computer-aided manufacturing (CAM) systems
- Computer Animations
- Developing Computer Games
- Hypertext and Hypermedia
- Network for programming routers, firewalls and other devices
- All Business Oriented Applications son on…

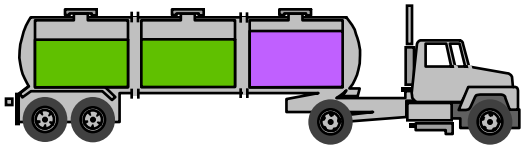
What is an Object?

Informally, an object represents an entity, either physical, conceptual, or software

An object is a concept, abstraction, or thing with sharp boundaries and meaning for an application

An object is something that has : State, Behavior, Identity

Physical entity

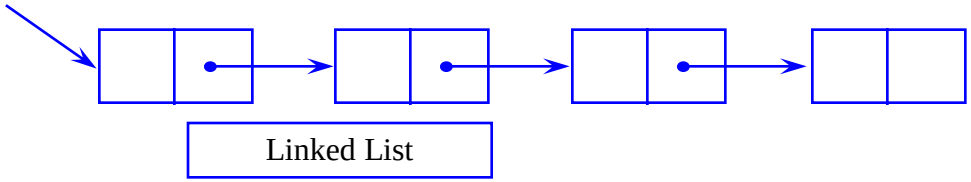


Truck

Conceptual entity

Chemical Process

Software entity



Representing Objects

An object is represented as rectangles with underlined names

:Professor

Class Name Only

ProfessorClark

Object Name Only

Professor Clark

ProfessorClark : Professor

Class and Object Name

Classes & Objects (1 of 2)

A **class** is collection of objects of similar type or it is a template.

Ex: fruit mango;

class object

Objects are instances of the type class.

Let us look at some customers

Cameron & Alphonso are two instances /examples /objects of Easy Shop customers

Attributes of Customer

Customer Id:	Customer Id:
1001	1002
Customer Name:	Customer Name:
Cameron	Alphonso
Telephone Number:	Telephone Number:
9901911445	9496244655
Address:	Address:
No.31, Silver Shine, Bangalore, India	No.255, Brigade, Bangalore, India

Values of attributes of Customer

What is a Class?

A class is a description of a group of objects with common properties (attributes), behaviour (operations), relationships, and semantics

- An object is an instance of a class

A class is an abstraction in that it:

- Emphasizes relevant characteristics
- Suppresses other characteristics

Sample Class

Class

Course

Properties

Name
Location
Days offered
Credit hours
Start time
End time

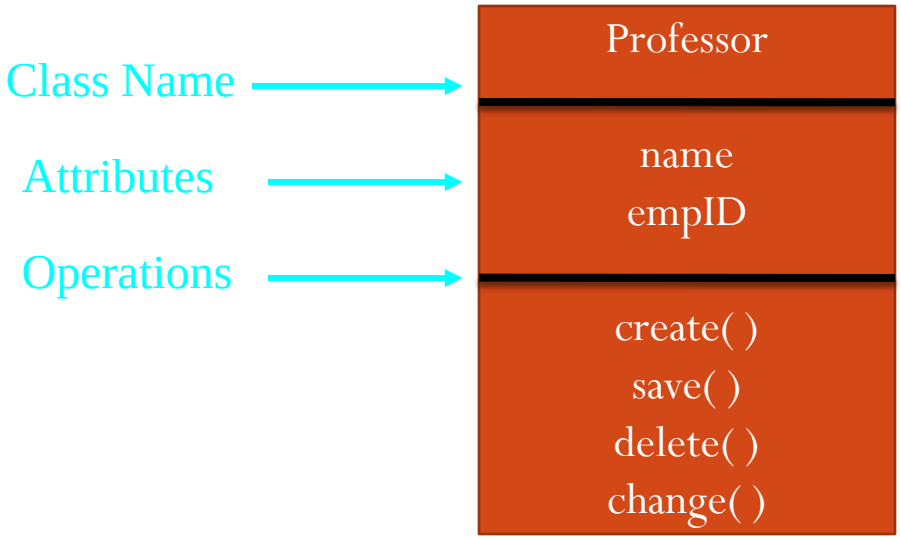
Behavior

Add a student
Delete a student
Get course roster
Determine if it is full

Class Compartments

A class is comprised of three sections

- The first section contains the class name
- The second section shows the structure (attributes)
- The third section shows the behaviour (operations)

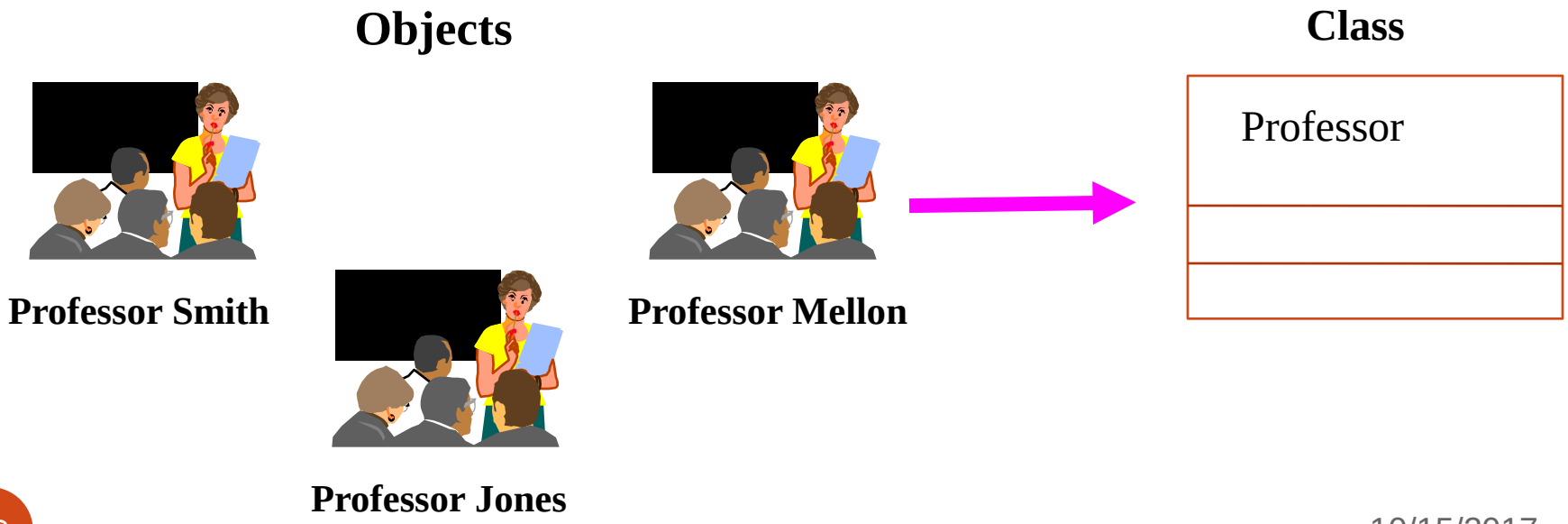


The Relationship Between Classes and Objects

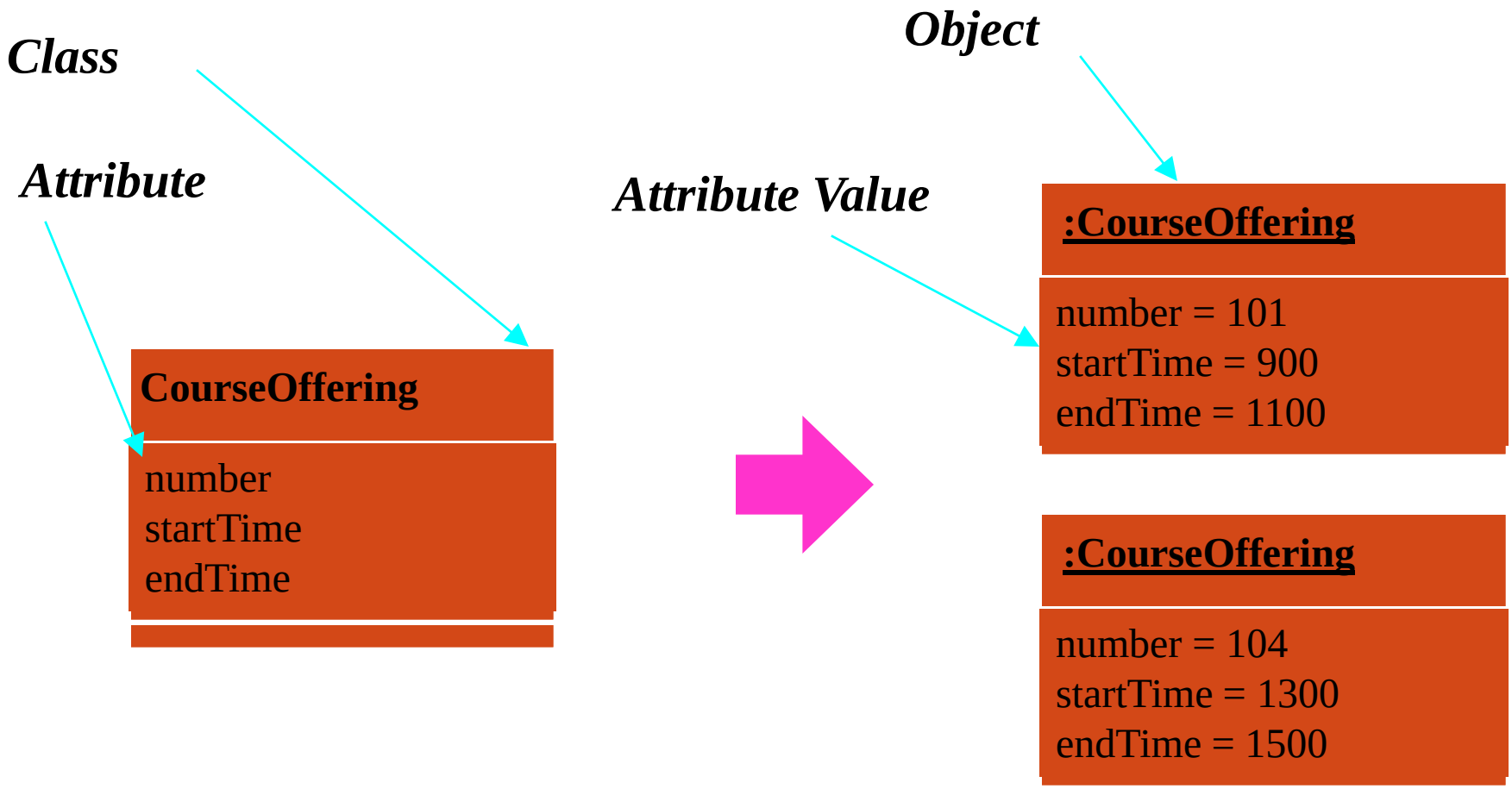
A class is an abstract definition of an object

- It defines the structure and behaviour of each object in the class
- It serves as a template for creating objects

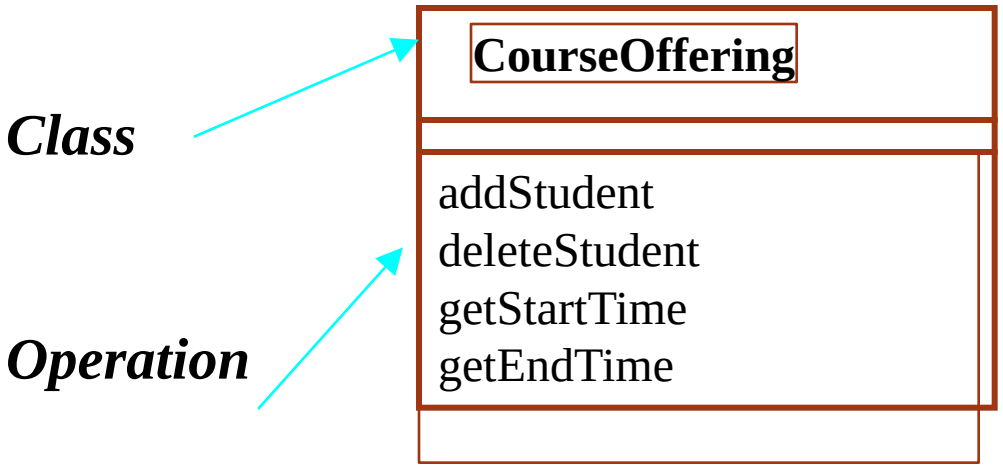
Objects are grouped into classes



What is an Attribute?



What is an Operation?



Classes & Objects (2 of 2)

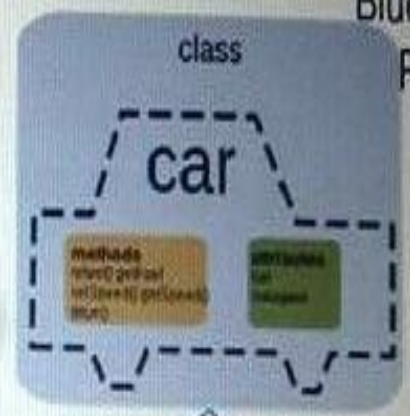
- A class is a prototype / design that describes the common attributes (properties) and activities (behaviors) of objects

Attributes

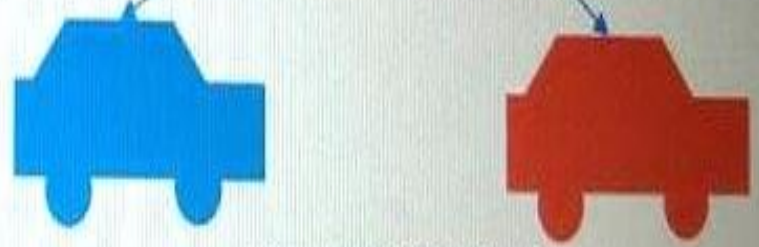
- Example : Customer Id, Name, Telephone number and Address

Behavior/ Activity

- Activities(behavior) exhibited by the class to external world
- Example: Purchasing items from the retail shop



Blueprint/Prototype
Passive Entity



Real time Objects
Active Entity

Object	Data or Attributes	Functions or Methods
Person	Name,Age,Sex,Color	Speak(),Walk(), Listern(),Write()
Vehicle	Name, Company, Model, Capacity,Color	Start(),Stop(),accelerate()
Polygon	Vertices, Border, Color	Draw(),Erase()
Account	Type, Number, Balance	Deposit(),Withdraw(),Enquire()
City	Name, Populatio, Area , Literacy rate	Analyse( ),Data( ),Display( )
Computer	Brand, Resolution, Price	Processing(),Display(),Printing()

Data Abstraction

- Data Abstraction refers to the process by which data and functions are defined in such a way that only essential details are revealed and the implementation details are hidden
- The main focus of the data abstraction is to separate the interface and the implementation of a program
- **Example:**
  - As User of Television set, we can switch it on or off, change the channel, set the volume and add external devices such as speakers and CD or DVD players without knowing the details about how its functionality has been implemented.
  - Therefore the internal implementation is completely hidden from the external world
- Similarly, in OOPS, classes provide PUBLIC methods to the outside world to provide the functionality of the object or to manipulate the object data and the implementation is hidden from the outside world

Abstraction – Guided Activity

- Users of the retail application – Billing staff, Admin, Retail
- Each user needs to know some details and need not know other

Who are the users of the retail application?

What are the things each user must know to perform their activities ?

Billing staff (Billing of customers)

Admin (Registration of customers)

Retail Outlet Manager (Registration of users)

ABSTRACTION : Process of identifying the essential details to be known and ignoring the non-essential details from the perspective of the user of the system

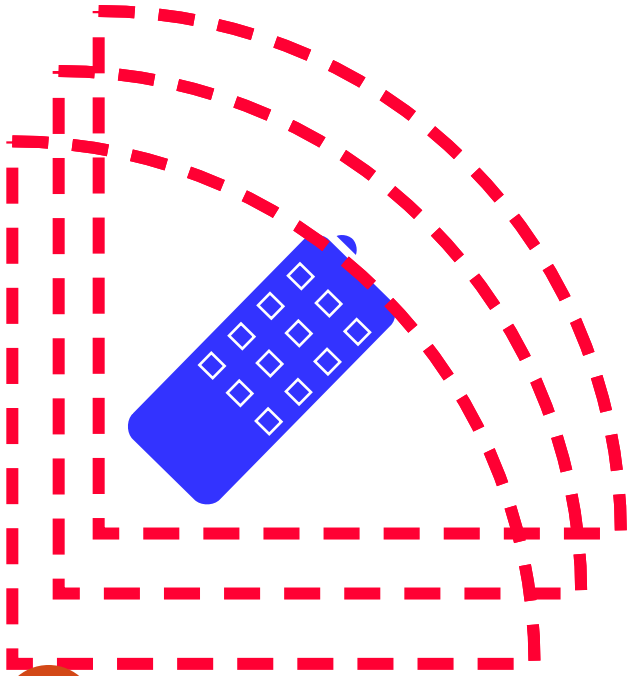
Data Encapsulation

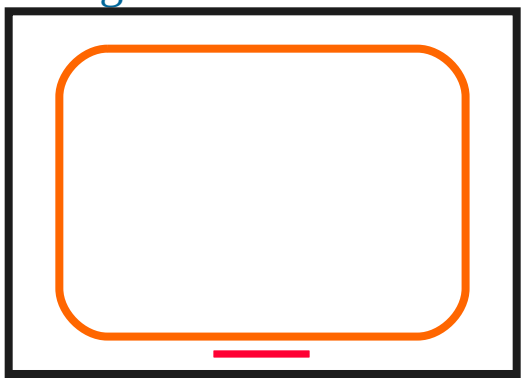
- Data encapsulation is also called as data hiding, is the technique of packing data and its functions into a single component (class) to hide implementation details of a class from the end user point of view
- Users are allowed to execute only a restricted set of operations (class methods) on the data members of that class.
- Therefore encapsulation organizes the data and methods into a structure that prevents data access by any function(or method) that is not specific to class
- **Encapsulation defines three access levels for data variables and member functions of the class:**
  - Any data or function with access level **public** can be accessed by any function belonging to any class. This is the lowest level of data protection
  - Any data or function with access level **protected** can be accessed only by that class or by any class that is inherited from it
  - Ay data or function with access level **private** can be accessed only by the class in which it is declared. This is the highest level of data protection

What is Encapsulation?

- Hide implementation from clients -Clients depend on interface
-

The wrapping up of data and functions into a single unit(called class) is known as **encapsulation**. Data encapsulation is the most striking features of a class.





How does an object encapsulate?
What does it encapsulate?

Improves Resiliency

Encapsulation – Guided Activity

How is a swipe machine used for payment of bill in a retail store?

- Swipe machine in a retail store
  - Used by billing staff to key the amount
  - Used by admin to record payment

ENCAPSULATION : A mechanism of hiding the internal details and allowing a simple interface which ensures that the object can be used without having to know how it works

Polymorphism

- Polymorphism is one of the essential concepts of OOP, refers to having several different forms
- Polymorphism is a concept that enables the programmers to assign a different meaning or usage to a method in different contexts.
- In Python, the word ‘polymorphism’ is often used with inheritance
- Polymorphism exists when a number of subclasses is defined which have methods of same name
- Polymorphism can also be applied to operators [Usage of ‘+’ operator on to number type and string type

Polymorphism – Guided Activity

Polymorphism:

It allows the single method to perform different actions based on the parameters.

- Payment of bill - Two modes
  - Cash (Calculation includes VAT)

What do you observe in this retail store scenario?

Total Amount = Purchase amount + VAT

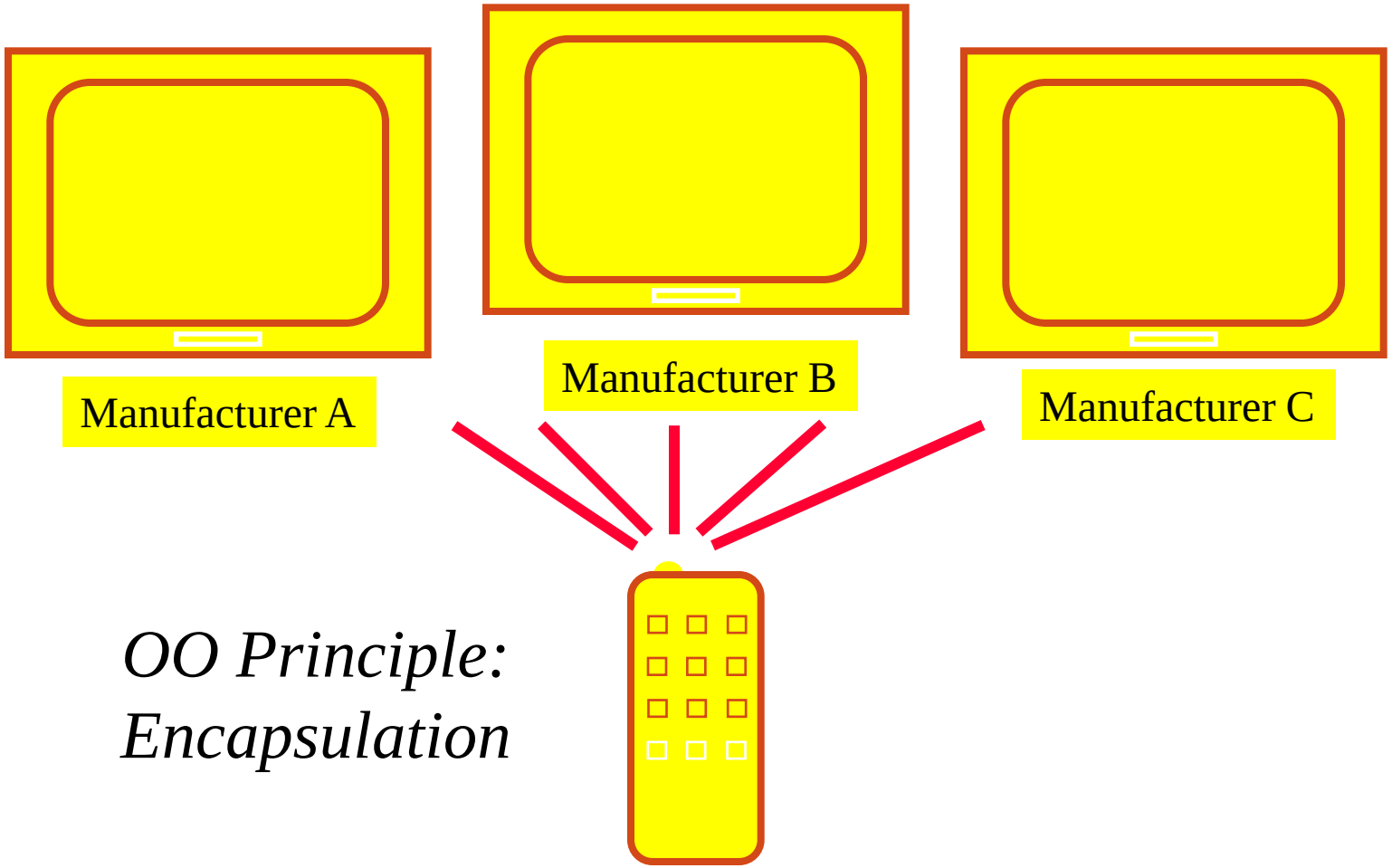
  - Credit card(Calculation includes processing charge and VAT)

Total Amount = Purchase amount + VAT + Processing charge

POLYMORPHISM: Refers to the ability of an object/operation to behave differently in different situations

What is Polymorphism?

The ability to hide many different implementations behind a single interface



OO Principle: Encapsulation

Inheritance

- Inheritance is a concept of OOP in which a new class is created form an existing class.
- The new class known as **Sub class** or **Derived class** or **Child class**, contains the attributes and methods of the **Super class** or **Base class** or **Parent class** [the existing class from which the new class is created
- The relationship followed from subclass to superclass is refers as **'is-a'** relation
- The subclass not only has all the states and behaviors associated with the super class but has other specialized features(additional data or methods)as well
- The main advantage of inheritance is the ability of **reuse** the code

Inheritance – Guided Activity

Inheritance is the process by which objects of one class acquire the properties of another class.
The concept of inheritance provides the **reusability.**

- Customers are of two kinds
  - Regular
  - Privileged

What are the two different types of customers you can see in the retail application?

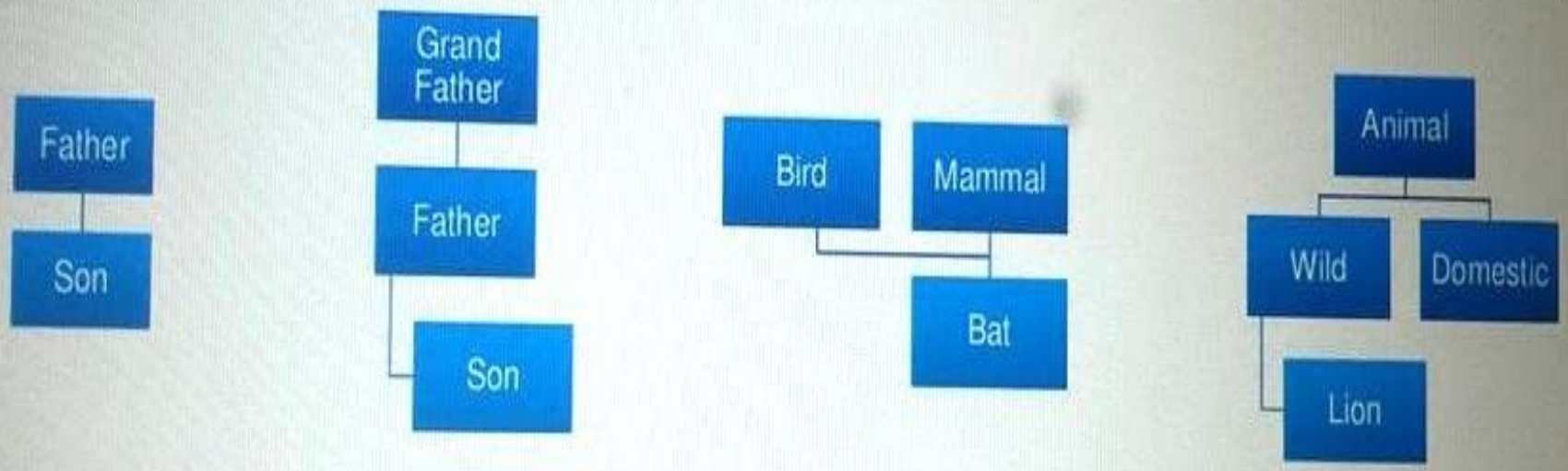
All customers have Customer Id, Name, Telephone Number and Address

The regular customer in addition is given discounts

The privileged customer gets a membership card based on which gifts are given

INHERITANCE : Is a mechanism which allows to define generalized characteristics and behavior and also create specialized ones. The specialized ones automatically tend to inherit all the properties of the generic ones

Types of Inheritance



Single Inheritance | Multi-level Inheritance | Multiple Inheritance | Hierarchy Inheritance

We focus more on Single, Multi-level and Hierarchy Inheritance for this course

- **Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Functions:** are for calculating things. They receive inputs and produce an output and/or have effects.
- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.
- **Function overloading:** The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.
- **Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.
- **Instance:** An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- **Instantiation:** The creation of an instance of a class.
- **Method:** A special kind of function that is defined in a class definition.

- **__init__ Method:** Inside the class body, we define this method – this is our object's method. . When we call the class object, a new instance of the class is created, and the __init__ method on this new object is immediately executed with all the parameters that we passed to the class object. The purpose of this method is thus to set up a new object using data that we have provided.

- **self:** in every method definition we have self as the first parameter, and we use this variable inside the method bodies – but we don't appear to pass this parameter in. This is because whenever we call a method on an object, *the object itself* is automatically passed in as the first parameter. This gives us a way to access the object's properties from inside the object's methods.

Working with Classes and Objects

- Define a class in Python
- In Python, a class is defined by using a keyword class like a function definition begins with the keyword def.
- **Syntax of a class definition:**

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Note: A class creates a new local namespace to define its all attribute. These attributes may be data or functions.

- # WAP to define a class

```
class Myclass:
    "This is Python Class"
    a=100
    def func(self):
        print("Hello welcome to my class")
```

Result: >>>Myclass.a will print value 100

Note: Here class creates a new local namespace to define Variables & Methods

Note: we can also use double underscore (__) as a special attribute for example in the above class "This is a Python class" is a doc attribute which is also termed as [doc string of a class]

```
>>> MyClass.__doc__    # To call the doc string
>>>Myclass.func    # To call the function
```

Which will print the address of the function of Myclass

Myclass.func Result: <function Myclass.func at 0x01CF5300>

- **Creating an object in Python:**
- We can create new object instances of the classes. The procedure to create an object is similar to a function call.
- Let's take an example to create a new instance object "ob". We can access attributes of objects by using the object name prefix.
  - Ex: # WAP to define a class

```
class Myclass:
    "This is Python Class"
     a=100
        def func(self):
            print("Hello welcome to my class")
ob=Myclass() # Creating of Object
```

- **Results:**
- >>> Myclass.func

  <function Myclass.func at 0x022101E0>
- >>> ob.func

<bound method Myclass.func of <__main__.Myclass object at 0x003AFC10>>

- >>> ob.func()

  Hello welcome to my class
- >>> ob.a

  100
- Note: Here, attributes may be data or method. Method of an object is corresponding functions of that class. For example: MyClass.func is a function object and ob.func is a method object.

- Once a class is defined, the next job is to create an object (or instance) of that class. The object can then access class variables and class methods using the dot operator(.)
- Syntax:

  object_name=class_name

  Creating an object or instance of a class is known as class instantiation
- Syntax:

  object_name.class_member_name
- **Ex:**
- **# Program to access class variable using class object**

```
class ABC:
    var=10      # class variable
obj=ABC()
print(obj.var)  # class variable is accessed using class object
```

- **# write a program illustrating example of class and object**

```
class Student:
    def __init__(self, name,id):
        self.name=name
        self.id=id
    def disp(self):
        print(self.name,self.id)
    s1=Student("Ravi",101)
    s2=Student("Arun",102)
    s1.disp()
    s2.disp()
```

- **Example**

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('Hello, my name is', self.name)
 p = Person('Swaroop')
p.say_hi()
# The previous 2 lines can also be written as
# Person('Swaroop').say_hi()
```

- **Output**

```
Hello, my name is Swaroop
```

- **Ex:**

```
class Student:
  def __init__(self, rollno, name):
    self.rollno = rollno
    self.name = name
  def displayStudent(self):
    print ("rollno : ", self.rollno,  ", name: ", self.name)
emp1 = Student(121, "Ajeet")
emp2 = Student(122, "Sonoo")
emp1.displayStudent()
emp2.displayStudent()
```

Result: rollno : 121 , name: Ajeet

rollno : 122 , name: Sonoo

Constructors

- Constructors are used for allocating memory space for variables. It is special function which gets activated automatically when object of that class is created.
- **Creating a constructor:**
- A constructor is a class function that begins with double underscore (_). The name of the constructor is always the same __init__().
- While creating an object, a constructor can accept arguments if necessary. When you create a class without a constructor, Python automatically creates a default constructor that doesn't do anything.
- Every class must have a constructor, even if it simply relies on the default constructor.

- --Instantiating Objects with '__init__':
- __init__is the default constructor
- __init__serves as constructor for the class. Usually does some intialization work
- An __init__ method can take any number of arguements
- However, the first arguement self in the definition of __init__ is special

Self

- The first argument of every method is a reference to the current instance of the class
- This particular variable refers to the object *itself*, and by convention, it is given the name self
- By convention, we name this argument self.
- In __init__, self refers to the objects currently being created; so , in other class methods, it refers to the instance whose method was called.
- Similar to the keyword 'this' in Java or C++
- But python uses self more often than java uses this.
- You do not give a value for this parameter (self) when you call the method, python will provide it.
- Although you must specify self explicitly when defining the method, you don't include it when calling the method.
- Python passes it for you automatically.

- **Example 2:-**

class Song(object):#We create a new class using the class statement and the name of the class. Here Object is called class variable

def __init__(self, lyrics):#__init__ method is useful to do any initialization you want to do with your object.lyrics parameter is called as instance variable or

self.ly = lyrics # there is something called "ly" that is part of the object called "self" so 'ly' is called as local variable and the other one (lyrics) is a object variable.

```
def sing_me_a_song(self):
    for line in self.ly:
        print (line)
happy_bday = Song(["Happy birthday to you",#instance object
                   "I don't want to get ueed",
                   "So I'll stop right there"])
bulls_on_parade = Song(["They rally around that family",
                        "With pockets full of shells"])
happy_bday.sing_me_a_song()#Accessing Attributes
bulls_on_parade.sing_me_a_song()#Accessing Attributes
```

'''

Creating Instance Objects

To create instances of a class, you call the class using class name and pass in
whatever arguments its __init__ method accepts.

Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable
would be accessed using class name
'''

Example 3:-

```
class BankAccount(object):
    def __init__(self, initial_balance=0):
        self.balance = initial_balance
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
money_add=int(input("Enter amount in Rs. :"))
my_account = BankAccount(money_add)
money_draw=int(input("Enter withdraw amount in Rs. :"))
my_account.withdraw(money_draw)
print (my_account.balance)
```

Destroying Objects (Garbage Collection)

- Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically reclaims blocks of memory that no longer are in use is termed Garbage Collection.
- Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes.
- An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with del, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero, Python collects it automatically.

- **Example**

```
class Point:
 def __init__( self, x=0, y=0):
     self.x = x
     self.y = y
 def __del__(self):
     class_name = self.__class__.__name__
     print class_name, "destroyed"
pt1 = Point()
 print id(pt1)
print id(1)
print id(2)
print id(3)
print id(1)
del pt1
```

- **Output:**

3072927948

167049392

167049380

167049368

167049392

Point destroyed

Inheritance

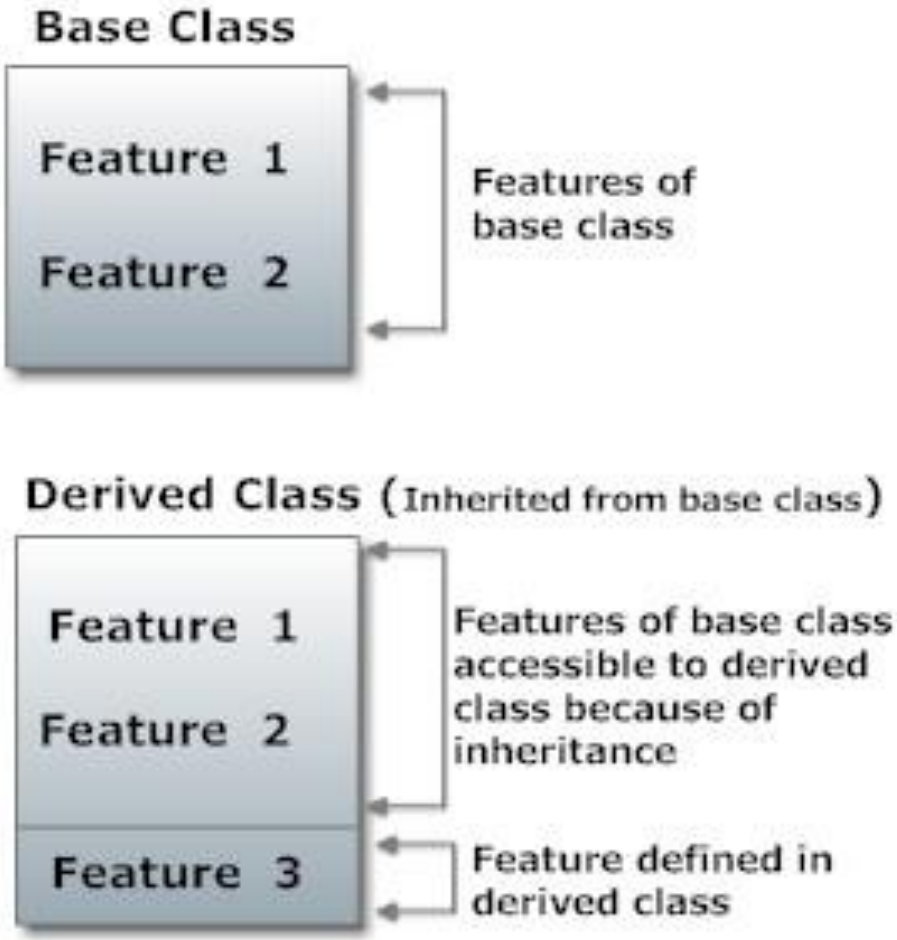
- Inheritance is a powerful feature in object oriented programming. It refers to defining a new class with little or no modification to an existing class.
- The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class. Derived class inherits features from the base class, adding new features to it. This results into re-usability of code.

Pictorial Representation

- **Syntax**

class DerivedClass(BaseClass):

body_of_derived_class



- **Example**
- The SchoolMember class in this situation is known as the **base class** or the **superclass**. The Teacher and Student classes are called the **derived classes** or **subclasses**.
- We will now see this example as a program (save as oop_subclass.py):

```
class SchoolMember:            #Represents any school member.
        def __init__(self, name, age):
                self.name = name
                self.age = age
                print('(Initialized SchoolMember: {})'.format(self.name))
        def tell(self):            #Tell my details.'''
                print('Name:"{}" Age:"{}"'.format(self.name, self.age))
class Teacher(SchoolMember):    #Represents a teacher.'''
        def __init__(self, name, age, salary):
                SchoolMember.__init__(self, name, age)
                self.salary = salary
                print('(Initialized Teacher: {})'.format(self.name))
        def tell(self):
                SchoolMember.tell(self)
                print('Salary: "{:d}"'.format(self.salary))
```

```
class Student(SchoolMember):     #Represents a student.'''
        def __init__(self, name, age, marks):
                SchoolMember.__init__(self, name, age)
                self.marks = marks
                print('(Initialized Student: {})'.format(self.name))
        def tell(self):
                SchoolMember.tell(self)
                print('Marks: "{:d}"'.format(self.marks))
t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 25, 75)
print()     # prints a blank line
members = [t, s]
for member in members:       # Works for both Teachers and Students
        member.tell()
```

Output:

(Initialized SchoolMember: Mrs. Shrividya)
(Initialized Teacher: Mrs. Shrividya)
(Initialized SchoolMember: Swaroop)
(Initialized Student: Swaroop)

Name:"Mrs. Shrividya" Age:"40" Salary: "30000"
Name:"Swaroop" Age:"25" Marks: "75"

Method Overriding

- In the above example, notice that __init__() method was defined in both classes, SchoolMember as well Teacher.
- When this happens, the method in the derived class overrides that in the base class. This is to say, __init__() in Teacher gets preference over the same in SchoolMember.
- Generally when overriding a base method, we tend to extend the definition rather than simply replace it. The same is being done by calling the method in base class from the one in derived class (calling SchoolMember.__init__() from __init__() in Teacher).
- A better option would be to use the built-in function super(). So, super().__init() is equivalent to SchoolMember.__init__() and is preferred.

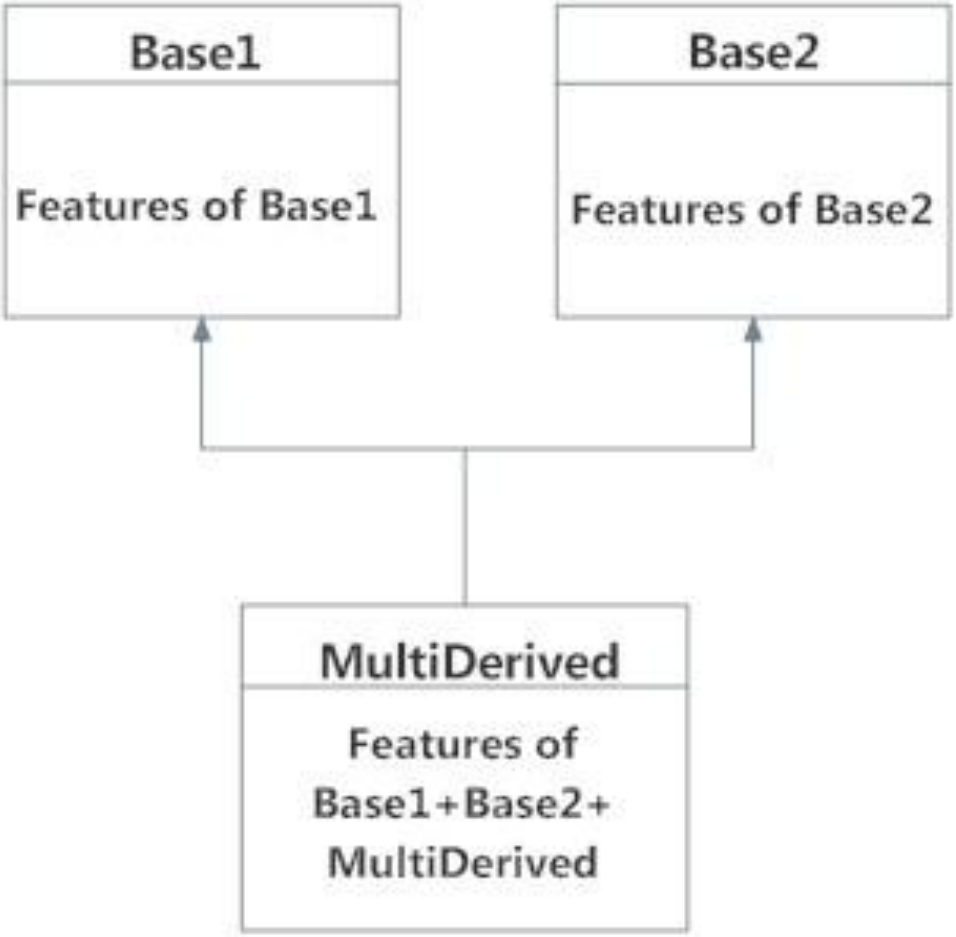
Multiple Inheritance

- A class can be derived from more than one base classes. The syntax for multiple inheritance is similar to single inheritance.

Example

```
class Base1:
    pass
class Base2:
    pass
class MultiDerived(Base1, Base2):
    pass
```

The class MultiDerived inherits from both Base1 and Base2.

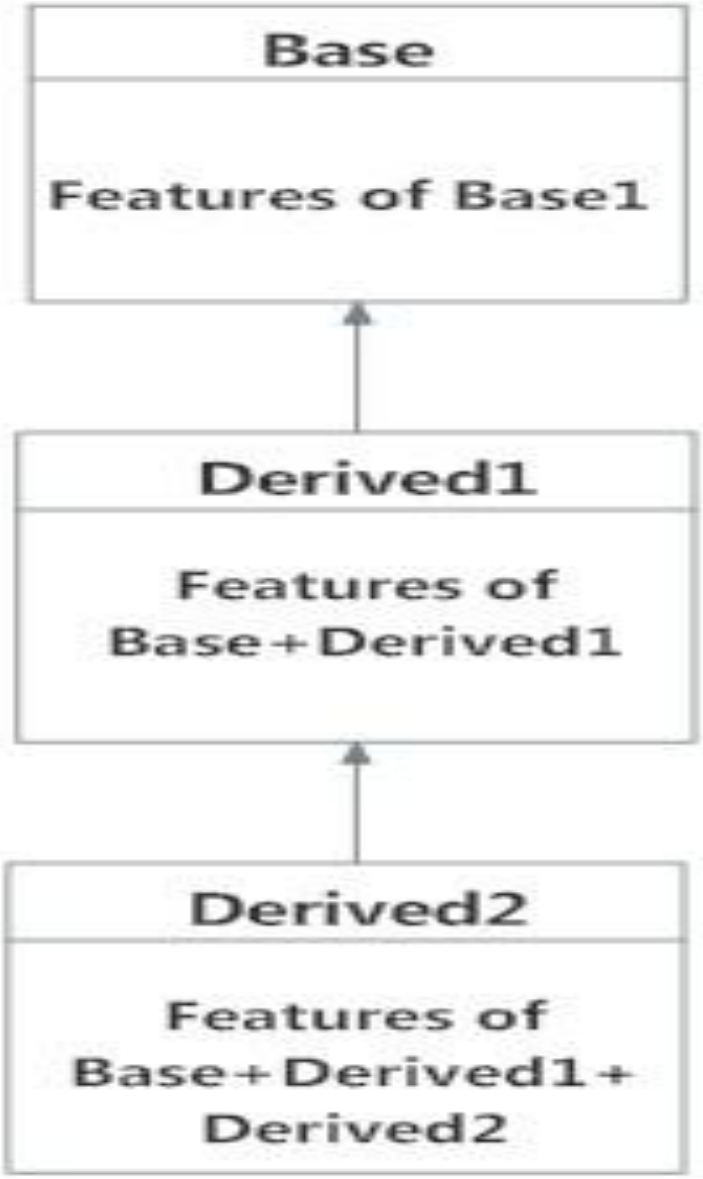

Base1
Features of Base1
Base2
Features of Base2
MultiDerived
Features of
Base1+Base2+
MultiDerived

- On the other hand, we can inherit form a derived class. This is also called multilevel inheritance. Multilevel inheritance can be of any depth in Python. An example with corresponding visualization is given below.

```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```


Base
Features of Base1
Derived1
Features of
Base+Derived1
Derived2
Features of
Base+Derived1+
Derived2

Find the output:

```
class Employee:
    Dept="CSE"
    def __init__(self,name):
        self.name=name
    #def disp(self):
      #  print(self.name)
E1=Employee("RAVI")
print(E1.Dept)
print(E1.name)
E2=Employee("SURESH")
print(E2.Dept)
print(E2.name)
```

Find the output:

```
class car:
    def __init__(self,company,model):
        self.company=company
        self.model=model
Polo=car("Volkesvagon","Polo VLX")
print("model=",Polo.model,"company=",Polo.company)
```


K.RaviKanth, Asst.Prof., Dept. of CSE, IIIT-RGUKT- Basara, T.S, IND.

Find the output:

```
class Car:
    company="BENZ"
    def __init__(self,model,year=2017):
        self.model=model
        self.year=year
    def display(self):
        print("Company-%s,Model-%s,Year -%d"%(self.company,self.model,self.year))
Dzire=Car("Swift DzX")
Dzire.display()
```

Find the output:

```
class Person:
    def __init__(self,name):
        self.__name=name
    def __display(self):
        print("Good Morning", self.__name)
    def greet(self):
        self.__display()
obj=Person("RGUKT")
obj.greet()
```

Assignment Questions

- Write a program that uses class to store the name and marks of students. Use list to store the marks in three subjects
- Write a program with class Employee that keeps a track of the number of employees in an organization and also stores their name, designation and salary details
- Write a program that has a class student that stores roll number,name and marks(in three subjects) of the students. Display the information (roll number,name and total marks) stored about the students
- Write a program that has a class Store which keeps a record of code and price of each product. Display a menu of all the products to the user and prompt him to enter the quantity of each item required.Generate a bill and display the total amount
- Write a program to deposit or withdraw money from a bank account
- Write a menu driven program that keeps record of books and journals available in a library [Take fields as Title of the Book,Author of the book and Price of the book
- Write a program that uses datetime modules within a class. Enter manufacturing date and expiry date of the product. The Program must display the years,months and days that are left over for Product Expiry

- Write a progam that has classes such as Student,Course and Department. Enroll a student in a course of a particular department using Inheritance
- Write a program with class Bill. The users have the option to pay the bill either by cheque or by cash. Use the inheritance to model this situation
- Write a program that has a class Person. Inherit a class Faculty from Person which also has a class Publications

- **NOTE: SUBMIT THE ASSIGNMENT BEFORE MT-3**

Case Study

&

Academic Project

IIIT BASARA

Infosys | Campus Connect

For Details Contact Me @ :

9247448766

ravikanth27787@gmail.com

python
Programming Language