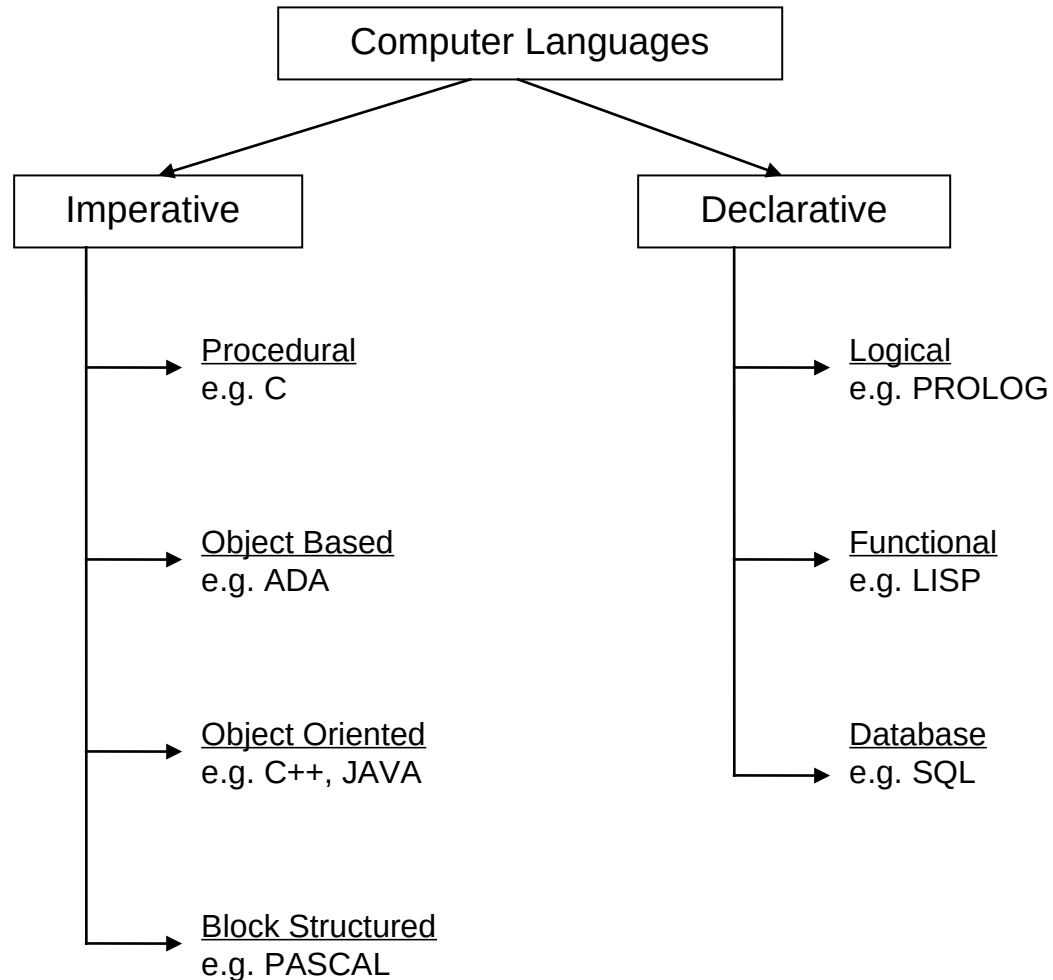


Object Oriented Programming in Python

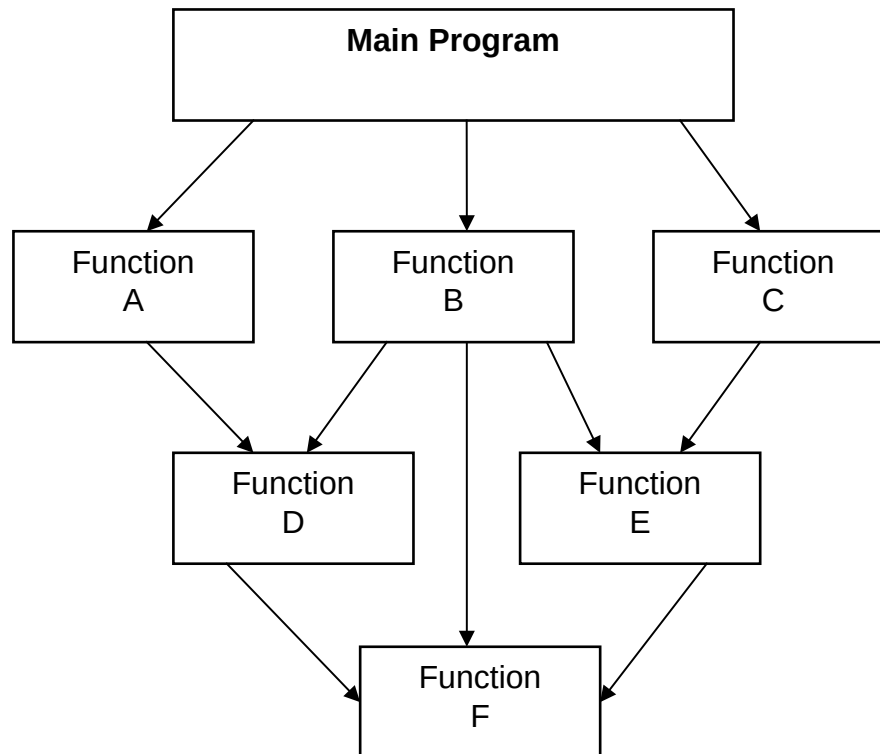
Tushar B. Kute,
<http://tusharkute.com>



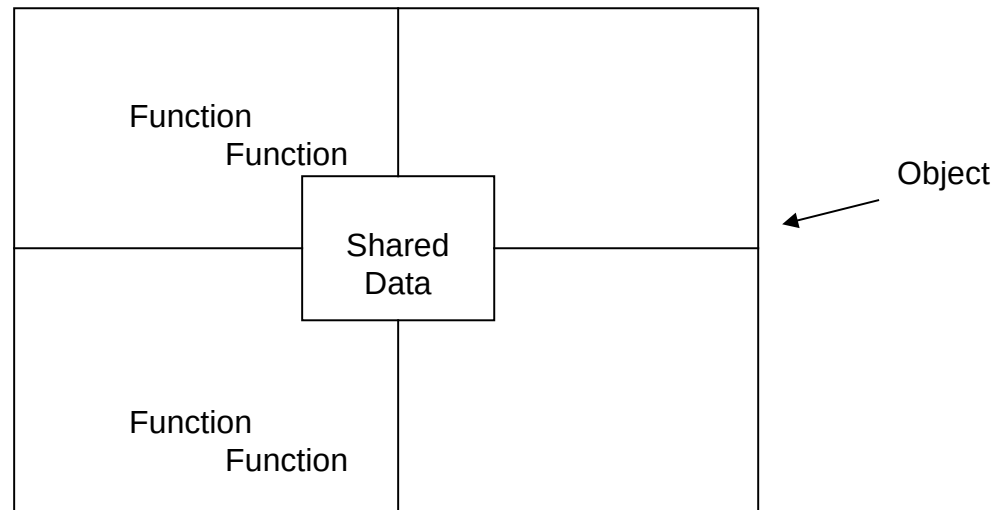
Programming Languages



Procedure Oriented Programming



Object Oriented Programming



What is Object Oriented Programming?

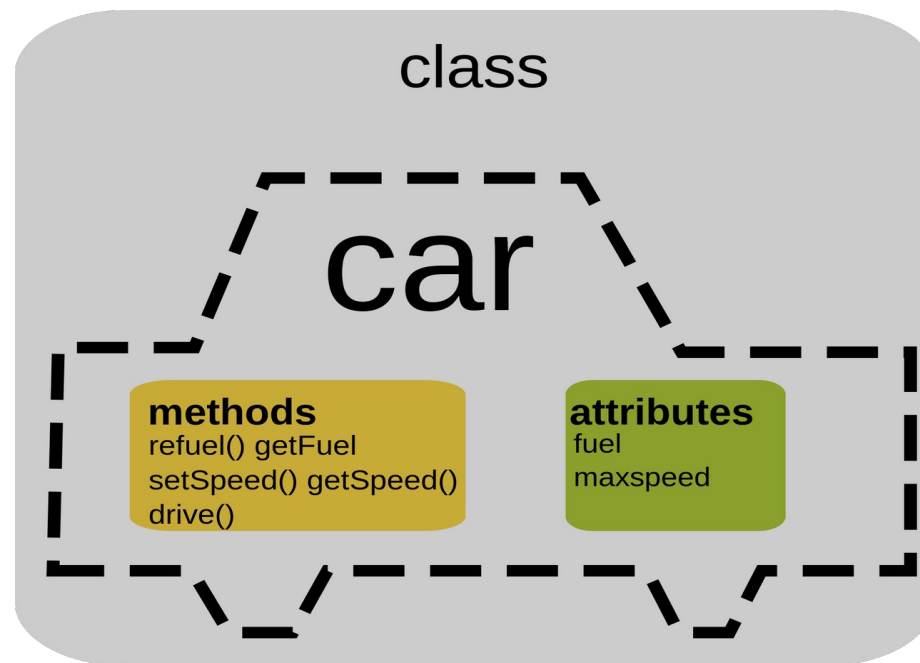
- Object-oriented Programming, or OOP for short, is a programming paradigm which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.
- For instance, an object could represent a person with a name property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending.

What is Object Oriented Programming?

- Put another way, object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc.
- OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.

Class

Classes are used to create new user-defined data structures that contain arbitrary information about something. In the case of an car, we could create an Car() class to track properties about the Car like the fuel and maxspeed.

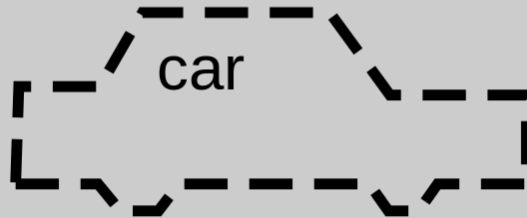


Objects

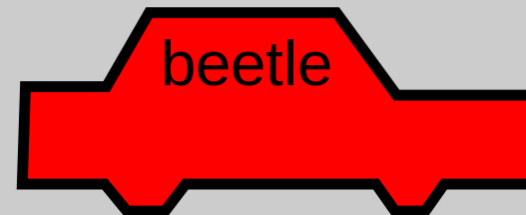
- While the class is the blueprint, an instance is a copy of the class with actual values, literally an object belonging to a specific class.
- Put another way, a class is like a form or questionnaire.
- It defines the needed information. After you fill out the form, your specific copy is an instance of the class; it contains actual information relevant to you.

Objects

class



objects



OOP Features

- Reduction in the complexity
- Importance of data
- Creation of new data structure
- Data Hiding
- Characterization of the objects
- Communication among objects
- Extensibility
- Bottom-up programming approach

OOP Concepts

- Class
- Object
- Data Hiding / abstraction / encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding

Example:

```
# A simple example class
```

```
class Test:
```

```
    # A sample method
```

```
    def fun(self):
```

```
        print("Hello")
```

```
# Driver code
```

```
obj = Test()
```

```
# Object Creation
```

```
obj.fun()
```

```
# Method Call
```

What is self?

- Class methods must have an extra first parameter in method definition. We do not give a value for this parameter when we call the method, Python provides it.
- If we have a method which takes no arguments, then we still have to have one argument – the **self**. See fun() in above simple example.
- This is similar to **this** pointer in C++ and **this** reference in Java.
- When we call a method of this object as myobject.method(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2) – this is all the special self is about.

The `__init__` method

- The `__init__` method is similar to constructors in C++ and Java.
- It is executed as soon as an object of a class is instantiated.
- The method is useful to do any initialization you want to do with your object.

Example

A Sample class with init method

```
class Person:
```

```
    # init method or constructor
```

```
    def __init__(self, name):  
        self.name = name
```

```
    # Sample Method
```

```
    def say_hi(self):  
        print('Hello, my name is', self.name)
```

```
p = Person('Rashmi')  
p.say_hi()
```

Person

p
name
say_hi()

Class and instance variables

- In Python, instance variables are variables whose value is assigned inside a constructor or method with **self**.
- Class variables are variables whose value is assigned **in class**. These are similar to static variables in C++ and Java.

Example

```
class Student:
    # Class Variable
    stream = 'cse' ←

    # The init method or constructor
    def __init__(self, roll):
        # Instance Variable
        self.roll = roll ←

# Objects of CSStudent class
a = Student(101)
b = Student(102)

print(a.stream) # prints "cse"
print(b.stream) # prints "cse"
print(a.roll)   # prints 101

# Class variables can be accessed using class
# name also
print(Student.stream) # prints "cse"
```

Instance variables in normal methods

```
class Student:
    stream = 'cse'
    def __init__(self, roll):
        self.roll = roll

    # Adds an instance variable
    def setAddress(self, address):
        self.address = address

    # Retrieves instance variable
    def getAddress(self):
        return self.address

# Driver Code
a = Student(501)
a.setAddress("Pune, Maharashtra")
print(a.getAddress())
```



Empty class

- We can create an empty class using pass statement in Python.

```
# An empty class  
class Test:  
    pass  
  
t = Test()
```

Data Hiding

- In simple words, data hiding is an object-oriented programming technique of hiding internal object details i.e. data members.
- Data hiding guarantees restricted data access to class members & maintain object integrity.
- Encapsulation, abstraction & data hiding is closely related to each other.
- In Python, we use double underscore (or `__`) before the attributes name and those attributes will not be directly visible outside.

Example

```
class MyClass:
    # Hidden member of MyClass
    __hiddenVariable = 0

    # A member method that changes
    # __hiddenVariable
    def add(self, increment):
        self.__hiddenVariable += increment
        print (self.__hiddenVariable)

# Driver code
myObject = MyClass()
myObject.add(2)
myObject.add(5)

# This line causes error
print (myObject.__hiddenVariable)
```

What is private?

- Private methods are accessible outside their class, just not easily accessible.
- Nothing in Python is truly private; internally, the names of private methods and attributes are mangled and unmangled on the fly to make them seem inaccessible by their given names.
- We can access the value of hidden attribute by a tricky syntax.

Accessing private variables

```
# A Python program to demonstrate that hidden  
# members can be accessed outside a class  
class MyClass:
```

```
    # Hidden member of MyClass  
    __hiddenVariable = 10
```

```
# Driver code  
myObject = MyClass()  
print(myObject._MyClass__hiddenVariable)
```

Printing objects

- Printing objects gives us information about objects we are working with.
- In C++, we can do this by adding a friend ostream& operator << (ostream&, const Foobar&) method for the class.
- In Java, we use toString() method.
- In python this can be achieved by using `__repr__` or `__str__` methods.

Example:

```
class Test:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __repr__(self):
        return "Test a:%s b:%s" % (self.a, self.b)

    def __str__(self):
        return "Values: a is %s & b is %s" %(self.a,self.b)

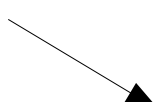
# Driver Code
t = Test(786, 419)
print(t) # This calls __str__()
print([t]) # This calls __repr__()
```

Without `__str__`

- If no `__repr__` method is defined then the default is used.

```
class Test:
    def __init__(self, a, b):
        self.a = a
        self.b = b
```

```
# Driver Code
t = Test(365, 102)
print(t)
```



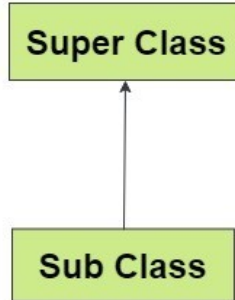
```
<__main__.Test object at 0x7f87eecf0c88>
```

Inheritance

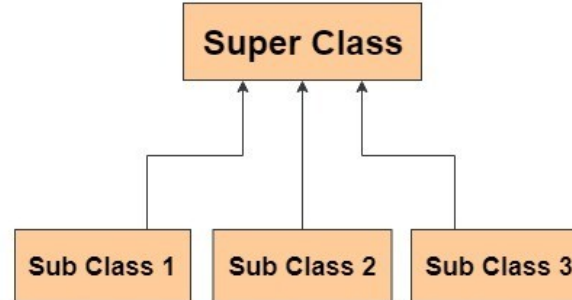
- One of the major advantages of Object Oriented Programming is re-use.
- Inheritance is one of the mechanisms to achieve the same.
- In inheritance, a class (usually called superclass) is inherited by another class (usually called subclass).
- The subclass adds some attributes to superclass.

Inheritance

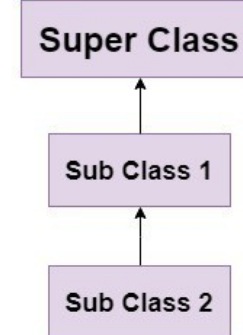
Single Inheritance



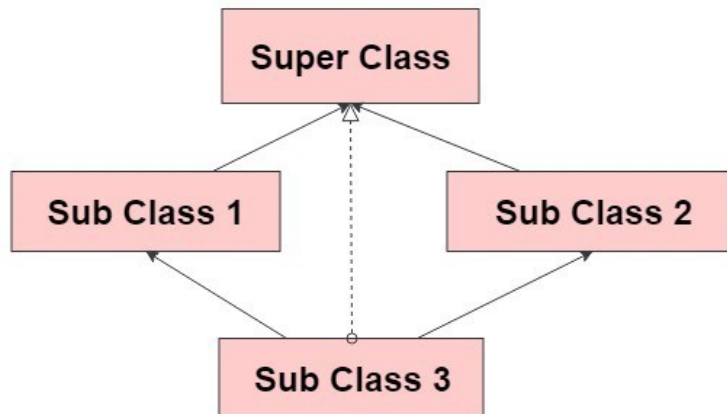
Hierarchical Inheritance



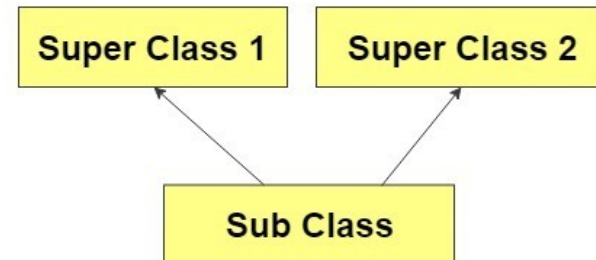
MultiLevel Inheritance



Hybrid Inheritance



Multiple Inheritance



Simple Inheritance

```
class Person(object):
    def __init__(self, name):
        self.name = name
    def getName(self):
        return self.name
    def isEmployee(self):
        return False

# Inherited or Sub class
class Employee(Person): ←
    def isEmployee(self):
        return True

# Driver code
emp = Person("Tushar") # SuperClass
print(emp.getName(), emp.isEmployee())

emp = Employee("Rashmi") #SubClass
print(emp.getName(), emp.isEmployee())
```

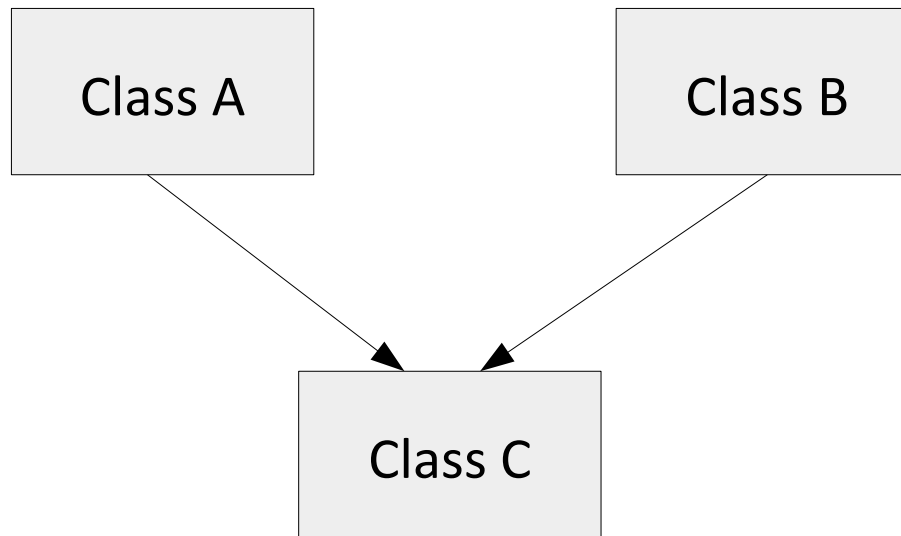
Check the subclass

```
class Base(object):  
    pass # Empty Class  
  
class Derived(Base):  
    pass # Empty Class  
  
print(issubclass(Derived, Base))  
print(issubclass(Base, Derived))  
  
d = Derived()  
b = Base()  
  
# b is not an instance of Derived  
print(isinstance(b, Derived))  
  
# But d is an instance of Base  
print(isinstance(d, Base))
```



Multiple Inheritance

- Multiple Inheritance is a type of inheritance where a class can inherit from more than one classes.



Example

```
class Base1(object):
    def __init__(self):
        self.str1 = "Rashmi"
        print("Base1")

class Base2(object):
    def __init__(self):
        self.str2 = "Tushar"
        print("Base2")

class Derived(Base1, Base2):
    def __init__(self):
        Base1.__init__(self)
        Base2.__init__(self)
        print("Derived")
    def printStrs(self):
        print(self.str1, self.str2)

ob = Derived()
ob.printStrs()
```



Accessing the super class

```
class Base(object):  
    def __init__(self, x):  
        self.x = x  
  
class Derived(Base):  
    def __init__(self, x, y):  
        Base.x = x ←  
        self.y = y  
  
    def printXY(self):  
        print(Base.x, self.y)
```

Driver Code

```
d = Derived(10, 20)  
d.printXY()
```

Accessing the super class

```
class Base(object):  
    def __init__(self, x):  
        self.x = x  
  
class Derived(Base):  
    def __init__(self, x, y):  
        super().__init__(x) ←  
        self.y = y  
  
    def printXY(self):  
        print(self.x, self.y)
```

```
# Driver Code  
d = Derived(10, 20)  
d.printXY()
```

Overloading

- Method overloading refers to defining same method name with multiple number of times with different parameters.
- Python does not support the method overloading.
- But by some alternative method we can create single method to perform different task, which is **NOT** actually the method overloading.

Example

```
def add(instanceOf,*args):  
    if instanceOf=='int':  
        result=0  
    if instanceOf=='str':  
        result=''  
    for i in args:  
        result+=i  
    return result  
  
print(add('int',4,2,3))  
print(add('str','I','Love','Python'))
```

Destructor

- Destructors are called when an object gets destroyed or out of scope.
- In Python, destructors are not needed as much needed in C++ because Python has a garbage collector that handles memory management automatically.
- The `__del__()` method is known as a destructor method in Python.
- It is called when all references to the object have been deleted i.e when an object is garbage collected.

Example

```
class Employee:

    # Initializing
    def __init__(self):
        print( 'Employee created.' )

    # Deleting (Calling destructor)
    def __del__(self):
        print( 'Employee deleted.' )

obj = Employee()
del obj
```

Operator Overloading

- Operator Overloading means giving extended meaning beyond their predefined operational meaning.
- For example operator + is used to add two integers as well as join two strings and merge two lists.
- It is achievable because '+' operator is overloaded by int class and str class.
- You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

Exercises

```
class A:
    def __init__(self, a):
        self.a = a

    # adding two objects
    def __add__(self, o):
        return self.a + o.a

ob1 = A(1)
ob2 = A(2)
ob3 = A("Hello")
ob4 = A("World")

print(ob1 + ob2)
print(ob3 + ob4)
```


Exercises

- Write a Python class which has two methods `get_String` and `print_String`. `get_String` accept a string from the user and `print_String` print the string in upper case.
- Write a Python class named `Rectangle` constructed by a length and width and a method which will compute the area of a rectangle.
- Write a Python class named `Circle` constructed by a radius and two methods which will compute the area and the perimeter of a circle.

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License



@mitu_skillologies



/MITuSkillologies



@mitu_group



/company/mitu-
skillologies



MITUSkillologies

Web Resources

<https://mitu.co.in>

<http://tusharkute.com>

contact@mitu.co.in

tushar@tusharkute.com