

# OOPS IN PYTHON

Inheritance enable us to define a class that takes all the functionality from parent class and allows us to add more. In this article, you will learn to use inheritance in Python.

---

### Base Class

Feature 1  
Feature 2

### Derived Class

Feature 1  
Feature 2  
Feature 3

# Interitance

- Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.
- The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.

- The Syntax is:

## Syntax

Derived classes are declared much like their parent class; however, a list of base classes to inherit from is given after the class name –

```
class SubClassName (ParentClass1[, ParentClass2, ...]):  
    'Optional class documentation string'  
    class_suite
```

```
class Parent:          # define parent class
    parentAttr = 100
    def __init__(self):
        print "Calling parent constructor"

    def parentMethod(self):
        print 'Calling parent method'

    def setAttr(self, attr):
        Parent.parentAttr = attr

    def getAttr(self):
        print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
    def __init__(self):
        print "Calling child constructor"

    def childMethod(self):
        print 'Calling child method'

c = Child()           # instance of child
c.childMethod()        # child calls its method
c.parentMethod()       # calls parent's method
c.setAttr(200)         # again call parent's method
c.getAttr()            # again call parent's method
```

Similar way, you can drive a class from multiple parent classes as follows –

```
class A:          # define your class A
.....

class B:          # define your class B
.....

class C(A, B):    # subclass of A and B
.....
```

You can use `issubclass()` or `isinstance()` functions to check a relationships of two classes and instances.

- The **`issubclass(sub, sup)`** boolean function returns true if the given subclass **`sub`** is indeed a subclass of the superclass **`sup`**.
- The **`isinstance(obj, Class)`** boolean function returns true if *obj* is an instance of class *Class* or is an instance of a subclass of *Class*

# Overriding Methods

- You can always override your parent class methods. One reason for overriding parent's methods is because you may want special or different functionality in your subclass.

# Example

```
class Parent:          # define parent class
    def myMethod(self):
        print 'Calling parent method'

class Child(Parent):   # define child class
    def myMethod(self):
        print 'Calling child method'

c = Child()            # instance of child
c.myMethod()           # child calls overridden method
```



```
class A(object):  
    def foo(self):  
        print 'A'
```

```
class B(A):  
    def foo(self):  
        print 'B'  
        super(B, self).foo()
```

```
class C(A):  
    def foo(self):  
        print 'C'  
        super(C, self).foo()
```

```
class D(B,C):  
    def foo(self):  
        print 'D'  
        super(D, self).foo()
```

```
d = D()  
d.foo()
```

# Base Overloading Methods

Following table lists some generic functionality that you can override in your own classes –

SN	Method, Description & Sample Call
1	<b><code>__init__ ( self [,args...] )</code></b> Constructor (with any optional arguments) Sample Call : <code>obj = className(args)</code>
2	<b><code>__del__( self )</code></b> Destructor, deletes an object Sample Call : <code>del obj</code>
3	<b><code>__repr__( self )</code></b> Evaluatable string representation Sample Call : <code>repr(obj)</code>
4	<b><code>__str__( self )</code></b> Printable string representation Sample Call : <code>str(obj)</code>
5	<b><code>__cmp__ ( self, x )</code></b> Object comparison Sample Call : <code>cmp(obj, x)</code>

# Overloading Operators



- Suppose you have created a Vector class to represent two-dimensional vectors, what happens when you use the plus operator to add them? Most likely Python will yell at you.
- You could, however, define the `__add__` method in your class to perform vector addition and then the plus operator would behave as per expectation

# Example

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

    def __str__(self):
        return 'Vector (%d, %d)' % (self.a, self.b)

    def __add__(self, other):
        return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

When the above code is executed, it produces the following result –

```
Vector(7,8)
```

# Data Hiding



- An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then are not be directly visible to outsiders.

Try it

```
class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```

When the above code is executed, it produces the following result –

```
1
2
Traceback (most recent call last):
  File "test.py", line 12, in <module>
    print counter.__secretCount
AttributeError: JustCounter instance has no attribute '__secretCount'
```

Python protects those members by internally changing the name to include the class name. You can access such attributes as *object.\_className\_\_attrName*. If you would replace your last line as following, then it works for you –

```
.....  
print counter._JustCounter__secretCount
```

When the above code is executed, it produces the following result –

```
1  
2  
2
```

# Questions1(Account Class)

- Design the class name **account** that contains:
    - A private **id** for account
    - A private **balance** for account
    - A private **intrate** for annual interest rate
    - A **constructor** that creates an account with specified id, initial balance and interest rate.
    - A method name **getmonthlyinterestrate()** that return monthly interest rate.
    - A method name **checkbalance()** that return balance amount from the account.
    - A method name **withdraw()** that withdraws a specified amount from the account.
    - A method name **deposit()** that deposits a specified amount to the account.
- == Create 10 accounts with initial balance 100 and rate of interest 4%.
- ==The system prompts the user to enter account ID, if it is correct open a main menu having chaises
1. Check balance, 2. withdraw, 3. deposit and 4. exit.