Inheritence ¶

- One of the most important goals of the object-oriented approach to programming is the creation of stable, reliable, reusable code. If you had to create a new class for every kind of object you wanted to model, you would hardly have any reusable code.
- In Python and any other language that supports OOP, one class can inherit from another class. This means you can base a new class on an existing class; the new class inherits all of the attributes and behavior of the class it is based on.
- A new class can override any undesirable attributes or behavior of the class it inherits from, and it can add any new attributes or behavior that are appropriate. The original class is called the **parent** class, and the new class is a **child** of the parent class. The parent class is also called a **superclass**, and the child class is also called a **subclass**.
- The child class inherits all attributes and behavior from the parent class, but any attributes that are defined in the child class are not available to the parent class.
- This also means a child class can override behavior of the parent class. If a child class defines a method that also appears in the parent class, objects of the child class will use the new method rather than the parent class method.

Python Inheritance Syntax :

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Example :

In [1]:

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]

    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(sel

    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
```

In [2]:

```
class Triangle(Polygon):
    def __init__(self):
        Polygon.__init__(self,3)

    def findArea(self):
        a, b, c = self.sides
        # calculate the semi-perimeter
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('The area of the triangle is %0.2f' %area)
```

In [3]:

```
t = Triangle()
t.inputSides()
```

```
Enter side 1 : 5
Enter side 2 : 5
Enter side 3 : 4
```

In [4]:

```
t.dispSides()
```

```
Side 1 is 5.0
Side 2 is 5.0
Side 3 is 4.0
```

In [5]:

```
t.findArea()
```

```
The area of the triangle is 9.17
```

Checking Inheritence

Two built-in functions isinstance() and issubclass() are used to check inheritances. Function isinstance() returns True if the object is an instance of the class or other classes derived from it. Each and every class in Python inherits from the base class object.

In [6]:

```
isinstance(t,Triangle)
```

Out[6]:

```
True
```

In [7]:

```
isinstance(t,Polygon)
```

Out[7]:

```
True
```

In [8]:

```
isinstance(t,int)
```

Out[8]:

```
False
```

In [9]:

```
isinstance(t,object)
```

Out[9]:

```
True
```

Similarly, **issubclass()** is used to check for class inheritance.

In [10]:

```
issubclass(Polygon,Triangle)

```

Out[10]:

```
False
```

In [11]:

```
 issubclass(Triangle,Polygon)
```

Out[11]:

```
True
```

In [12]:

```
issubclass(bool,int)
```

Out[12]:

```
True
```

In [13]:

```
type(t)
```

Out[13]:

```
__main__.Triangle
```

In [14]:

```
type(t) == '__main__.Triangle'
```

Out[14]:

```
False
```