Object Oriented Programing

Axpino Technologies

Python Programing

OOP

Class Definition

```
class a:
        def add(self):
                print(“This is add function")

obj = a()
a.add(obj)
obj.add()
```

OOP

Multiple Object of a class

```
class a:
        def add(self):
                print(“This is add function")

obj = a()
obj2=a()
obj.add()
obj2.add()
```

OOP

__init__ Method

```
class a:
        def __init__(self)
                print(“This is init function")

obj = a()
```

OOP

Passing Arguments to a Method

```
class person:
        def a(self,name):
                print("Hi",name)

obj = person()
obj.a("Harminder")
```

OOP

Accessing Variable

```
class person:
        x=1

obj = person()
print(obj.x)
```

Binding variable to object

OOP

```
class a:
        def b(self,k=5,n=4):
                self.k=k
                self.n=n
        def c(self):
                print(self.k,self.n)


obj = a()
obj.b(44,67)
obj.c()

obj2 = a()
obj2.b()
obj2.c()
```

Binding variable to object using __init__

OOP

```
class a:
        def __init__(self,k=5,n=4):
                self.k=k
                self.n=n
        def c(self):
                print(self.k,self.n)

obj = a(44,67)
obj.c()

obj2 = a()
obj2.c()
```

Instance Variable

OOP

```
class a:
        def __init__(self):
                self.b=5

c1=a()
c2=a()
print(c1.b)
print(c2.b)
c1.b=10
print(c1.b)
print(c2.b)
```

Class Variable

OOP

```
class a:
        x=4
        def __init__(self):
                self.b=5

c1=a()
c2=a()
print(c1.x)
print(c2.x)
a.x=15
print(c1.x)
print(c2.x)

c1.x=55
print(c1.x)
print(c2.x)
```

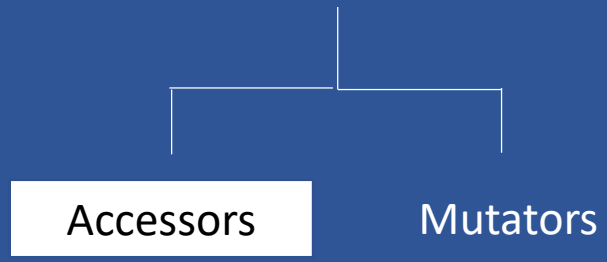
Types of Methods
(Instance Methods)

OOP

```
class student:
        def __init__(self,m1,m2,m3):
                self.m1=m1
                self.m2=m2
                self.m3=m3

        def avg(self):
                return (self.m1+self.m2+self.m3)/3

s1 = student(23,56,44)
s2 = student(90,89,45)
print(s1.avg())
print(s2.avg())
```

Types of Methods

(Instance Methods)

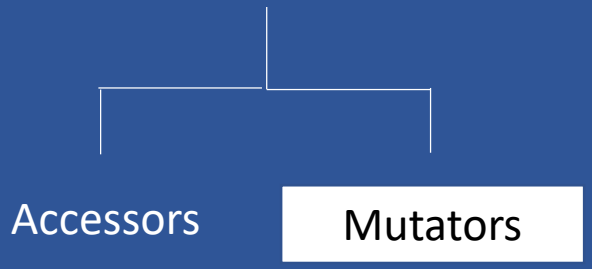



```
class student:
        def __init__(self):
                self.a="Harminder"

        def get_a(self):
                print(self.a)

s1 = student()
s1.get_a()
```

OOP

Types of Methods

(Instance Methods)




```
class student:
        def __init__(self):
                self.a="Harminder"

        def set_a(self):
                self.a="surender"
                return self.a

s1 = student()
print(s1.a)
print(s1.set_a())
```

OOP

Types of Methods
(Class Methods)

OOP

```
class student:
        school="vsics"

        @classmethod
        def get_school(cls):
                print(cls.school)

s1=student()
student.get_school()
```

Types of Methods
(Static Methods)

OOP

```
class student:
        @staticmethod
        def a():
                print("hi")

s1=student()
s1.a()
```

Inner Class

OOP

```
class student:
        def a(self):
                print("hi")
                self.obj = self.b()
                self.obj.hello()

        class b:
                def hello(self):
                        print("hello")

s1=student()
s1.a()
```

Inner Class

(using Object of inner class outside main class)

OOP

```
class student:
        def a(self):
                print("hi")
                self.obj = self.b()

        class b:
                def hello(self):
                        print("hello")

s1=student()
s1.a()
s1.obj.hello()
```

Inner Class

(Defining Object of inner class outside main class)

OOP

```
class student:
        def a(self):
                print("hi")

        class b:
                def hello(self):
                        print("hello")

s1=student()
obj=s1.b()
obj.hello()
```

Inheritance

OOP

```
class a:
        def feature1(self):
                print("Feature 1 is working")

        def feature2(self):
                print("Feature 2 is working")

class b(a):
        def feature3(self):
                print("Feature 3 is working")

        def feature4(self):
                print("Feature 4 is working")

obj1 = b()
obj1.feature1()
```

Multi Level Inheritance

OOP

```
class a:
        def feature1(self):
                print("Feature 1 is working")

        def feature2(self):
                print("Feature 2 is working")

class b(a):
        def feature3(self):
                print("Feature 3 is working")

        def feature4(self):
                print("Feature 4 is working")

class c(b):
        def feature5(self):
                print("Feature 5 is working")

obj1 = c()
obj1.feature1()
```

Multiple Inheritance

OOP

```
class a:
        def feature1(self):
                print("Feature 1 is working")

        def feature2(self):
                print("Feature 2 is working")

class b:
        def feature3(self):
                print("Feature 3 is working")

        def feature4(self):
                print("Feature 4 is working")

class c(a,b):
        def feature5(self):
                print("Feature 5 is working")

obj1 = c()
obj1.feature1()
```

Constructor Behavior in Single/Multi level inheritance

OOP

```
class a:
        def __init__(self):
                print("Init of a")
        def feature1(self):
                print("This is feature 1")
        def feature2(self):
                print("This is Feature 2")

class b(a):
        def __init__(self):
                super().__init__()
                print("Init of b")
        def feature3(self):
                print("This is feature 3")
        def feature4(self):
                print("This is Feature 4")

k = b()
```

Constructor Behavior in Multiple Inheritance Method Resolution Order (MRO)

OOP

```
class a:
        def __init__(self):
                super().__init__()
                print("Init of a")
        def feature1(self):
                print("This is feature 1")
        def feature2(self):
                print("This is Feature 2")

class b:
        def __init__(self):
                super().__init__()
                print("Init of b")
        def feature3(self):
                print("This is feature 3")
        def feature4(self):
                print("This is Feature 4")

class c(a,b):
        def __init__(self):
                super().__init__()
                print("Init of c")
        def feat(self):
                print("This is feat")

k = c()
```