Object Oriented Programming in Python

Table of Contents

Class members(public, private), class variables, instance members

class myclass:

```
    __value = 10
    knd = 'canine'
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.__value = str("New Val") + str(name)

    def get_values(self):
        ret_str = str(self.name) + str(" : ") + str(self.age)
        print (self.__value)
        print (self.knd)
        return ret_str

myc1 = myclass("one",1)
print (myc1.get_values())
print (myc1.knd)
```

Multiple Inheritance:

```
  class myclass():
    def __init__(self,name,age):
        self.name = name
        self.age  = age

    def get_values(self):
       return (self.name, self.age)

  class myanotherclass():
    def __init__(self,height,weight):
        self.height = height
        self.weight = weight
    def get_values(self):
        return (self.height, self.weight)

  class my_derived_class(myclass):
     def __init__(self,name,age,sex):
```

```
        myclass.__init__(self,name,age)
        self.sex = sex

    def get_values(self):
        return (self.name, self.age, self.sex)

class my_new_dclass(myclass, myanotherclass):
    def __init__(self):
        myclass.__init__(self,name="newname",age=25)
        myanotherclass.__init__(self,height=175,weight=75)

    def print_values(self):
         print (self.get_values())

def main():
    myc1 = myclass("Suresh", 22)
    myc2  = myclass("Rajesh", 25)

    mydc1 = my_derived_class("Ramesh", 26, "M")
    mydc2 = my_derived_class("Kumar", 19, "M")
    print (mydc2.get_values())

    myndc1 = my_new_dclass()
    myndc1.print_values()
main()
```

Function Overriding

Functions with same name from base and derived classes

```
import types

class base:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name
```

```
class der1(base):
    def __init__(self,name,age):
        base.__init__(self, name)
        self.age = age

    def get_name(self):
        return "name: %s, age %d"%(self.name,int(self.age))


class der2(base):
    def __init__(self,name,age,sex):
        base.__init__(self, name)
        self.age = age
        self.sex = sex

    def get_name(self):
        return "name: %s, age %d, sex : %s"%(self.name,int(self.age), self.sex)



def main():
    b1 = base("This is base class")
    print (b1.get_name())

    d1 = der1("This is derived1 class",10)
    print (d1.get_name())



    d2 = der2("This is derived2 class",20,"M")
    print (d2.get_name())

if __name__ == "__main__":
```

```
main()
```

Function overloading with default arguments.

Functions with same name with different set of arguments

```
import types

class base:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name


class der1(base):
    def __init__(self,name,age):
        base.__init__(self, name)
        self.age = age

    def get_name(self):
        return "name: %s, age %d"%(self.name,int(self.age))


class der2(base):
    def __init__(self,name,age,sex):
        base.__init__(self, name)
        self.age = age
        self.sex = sex

    def get_name(self):
        return "name: %s, age %d, sex : %s"%(self.name,int(self.age), self.sex)

    def getname(self, key = "KEY", value= "VALUE"):
        return "key = %s, value = %s, name: %s, age %d, sex : %s"%(key,value,self.name,int(self.age),
self.sex)
```

```
def main():
    b1 = base("This is base class")
    print (b1.get_name())

    d1 = der1("This is derived1 class",10)
    print (d1.get_name())

    d2 = der2("This is derived2 class",20,"M")
    print (d2.get_name())
    #function overloading with default arguments
    print (d2.getname())
    print (d2.getname("KEY1"))
    print (d2.getname("KEY2","VALUE2"))

if __name__ == "__main__":
   main()
```