Python Fundamentals day 53

Today's Agenda

- super() in inheritance

super( ) in Inheritance

The **super** keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor. The most common use of the **super** keyword is to eliminate the confusion between super-classes and sub-classes that have methods with the same name.

Let us look at an example and see how to use super()

```
class Customer:

    def __init__(self,name,ph,email):
        self.name=name
        self.ph=ph
        self.email=email

class PlatinumCustomer(Customer):
    def __init__(self,name,ph,email,plat_id):
        self.name=name
        self.ph=ph
        self.email=email
        self.plat_id=plat_id
    def display(self):
        print(self.__dict__)

def main():
    p=PlatinumCustomer('Rohit',9900887766,'rohit@gmail.com',10)
    p.display()

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

Output:

```
In [1]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
{'name': 'Rohit', 'ph': 9900887766, 'email':
'rohit@gmail.com', 'plat_id': 10}
```

In the above example we can reduce the size of __init__() of child class i.e. PlatinumCustomer() by using super(). Let us see below how to do it

```
class Customer:

    def __init__(self,name,ph,email):
        self.name=name
        self.ph=ph
        self.email=email

class PlatinumCustomer(Customer):
    def __init__(self,name,ph,email,plat_id):
        super().__init__(name,ph,email)
        self.plat_id=plat_id
    def display(self):
        print(self.__dict__)

def main():
    p=PlatinumCustomer('Rohit',9900887766,'rohit@gmail.com',10)
    p.display()

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

Output:

```
In [2]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
{'name': 'Rohit', 'ph': 9900887766, 'email':
'rohit@gmail.com', 'plat_id': 10}
```

We can also call any other method of parent class using super(). Let us take another example to know how

```
class Customer:

    def __init__(self,name,addr,ph_no):
        self.name=name
        self.addr=addr
        self.ph_no=ph_no

    def place_order(self,dish):
        cost=0
        del_charge=50
        if dish=='pizza':
            cost= 500+ del_charge
        else:
            cost=250+del_charge
        return cost

class PlatinumCustomer(Customer):
    def __init__(self,name,addr,ph_no,plat_id):
        super().__init__(name,addr,ph_no)
        self.plat_id=plat_id
    def place_order(self,dish):
        del_charge=50
        return (super().place_order(dish)-del_charge)*0.95

def main():
    p=PlatinumCustomer('Rohit','ITC',9900887766,12)
    print(p.place_order('pizza'))

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

Output:

```
In [5]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
475.0
```

Next let us see how to use super() in multilevel inheritance

```
class A:
    def fun(self):
        print('A')

class B(A):
    def fun(self):
        print('B')

class C(B):
    def fun(self):
        super().fun() #super(C,self).fun()
        print('C')
c=C()
c.fun()
```

Output:

```
In [6]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
B
C
```

Great!! But can we access class A methods as well? Certainly we can

```
class A:
    def fun(self):
        print('A')

class B(A):
    def fun(self):
        print('B')

class C(B):
    def fun(self):
        super(B,self).fun()
        print('C')
c=C()
c.fun()
```

Output:

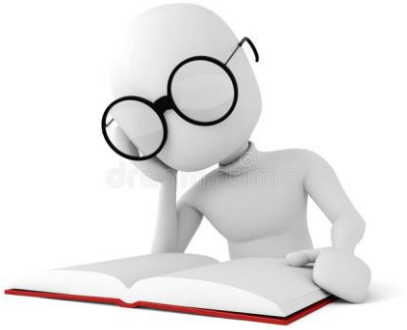
```
In [7]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
A
C
```

Now let us go ahead to multiple inheritance

```
class A:
    def fun(self):
        print('A')

class B:
    def fun(self):
        print('B')

class C(A,B):
    def test(self):
        super().fun()
        print('C')
c=C()
#help(C)
c.test()
```



Output:

```
In [10]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
A
C
```

What if we want to check whether the object or variable is an instance of specified class type or datatype and check if a class is subclass of another class. How do we do it?

```
class A:
    def fun(self):
        print('A')

class B:
    def fun(self):
        print('B')

class C(A,B):
    def test(self):
        super().fun()
        print('C')
c=C()
#help(C)
#c.test()
print(isinstance(c,int))
print(issubclass(C,object))
```

Output:

```
In [12]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
False
True
```