Programming with Python & Java (CS 29008)

Lab 4

Object Oriented Programming in Python
Inheritance

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

School of Electronics Engineering
KIIT Deemed to be University
Bhubeneswar, Odisha

Contents

Lab 3: Classes, objects, methods & variables

1 Introduction

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a new class (subclass) to inherit properties and methods from an existing class (superclass). In Python, like in many other object-oriented languages, inheritance enables code reuse and facilitates building hierarchies of classes. Example of inheritance in Python:

1. **Defining a Base Class (Superclass):** A base class, also known as a superclass or parent class, serves as the blueprint for other classes to inherit from. It encapsulates common attributes and behaviors that subclasses can inherit.

```
    class Animal:
    def __init__(self, species):
        self.species = species

    def speak(self):
        pass  # Abstract method

    def info(self):
        print(f"I am a {self.species}")
```

2. **Creating Subclasses:** Subclasses are new classes derived from the base class. They inherit attributes and methods from the superclass and can also have their own unique attributes and methods.

```
class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"
```

3. **Accessing Superclass Methods:** Subclasses inherit all methods and attributes of the superclass. They can access these methods using dot notation.

```
dog = Dog("Canine")
dog.info()  # Output: I am a Canine
print(dog.speak())  # Output: Woof!
```

2 Types of Inheritance

2.1 Single Inheritance

The concept of inheriting the properties from one class to another class is known as single inheritance.

```
class P:
    def m1(self):
        print("Parent Method")
class C(P):
    def m2(self):
        print("Child Method")

c=C()
c.m1()
c.m2()
```

2.2 Multi Level Inheritance

The concept of inheriting the properties from multiple classes to single class with the concept of one after another is known as multilevel inheritance.

```
class P:
    def m1(self):
        print("Parent Method")
class C(P):
    def m2(self):
        print("Child Method")
class CC(C):
    def m3(self):
        print("Sub Child Method")

>> c=CC()
>> c.m1()
>> c.m2()
>> c.m3()
```

2.3 Hierarchical Inheritance

The concept of inheriting properties from one class into multiple classes which are present at same level is known as Hierarchical Inheritance

```
class P:
    def m1(self):
        print("Parent Method")
```

```
class C1(P):
    def m2(self):
        print("Child1 Method")
class C2(P):
    def m3(self):
        print("Child2 Method")

>> c1=C1()
>> c1.m1()
>> c1.m2()
>> c2=C2()
>> c2.m1()
>> c2.m3()
```

2.4 Multiple Inheritance:

The concept of inheriting the properties from multiple classes into a single class at a time, is known as multiple inheritance.

```
class P1:
    def m1(self):
        print("Parent1 Method")

class P2:
    def m2(self):
        print("Parent2 Method")

class C(P1,P2):
    def m3(self):
        print("Child2 Method")
>> c=C()
>> c.m1()
>> c.m2()
>> c.m3()
```

If the same method is inherited from both parent classes, then Python will always consider the order of Parent classes in the declaration of the child class.
class C(P1, P2): P1 method will be considered
class C(P2, P1): P2 method will be considered

```
class P1:
   def m1(self):
     print("Parent1 Method")
class P2:
   def m1(self):
     print("Parent2 Method")
```

```
class C(P1,P2):
   def m2(self):
     print("Child Method")

>> c=C()
>> c.m1()
>> c.m2()
```

2.5 Hybrid Inheritance

Combination of Single, Multi level, multiple and Hierarchical inheritance is known as Hybrid Inheritance.

2.6 Cyclic Inheritance

The concept of inheriting properties from one class to another class in cyclic way, is called Cyclic inheritance. Python won't support for Cyclic Inheritance of course it is really not required. Eg - 1: class A(A):pass NameError: name 'A' is not defined

```
class A(B):
    pass
class B(A):
    pass
```

3 super() Method

super() is a built-in method which is useful to call the super class constructors,variables and methods from the child class.

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def display(self):
        print('Name:',self.name)
        print('Age:',self.age)

class Student(Person):
    def __init__(self,name,age,rollno,marks):
        super().__init__(name,age)
        self.rollno=rollno
        self.marks=marks
    def display(self):
        super().display()
        print('Roll No:',self.rollno)
        print('Marks:',self.marks)

>> s1=Student('John',22,101,90)
>> s1.display()
```

In the above program we are using super() method to call parent class constructor and display() method.

```
#Demo Program-2 for super():
class P:
    a=10
    def __init__(self):
        self.b=10
    def m1(self):
        print('Parent instance method')

    @classmethod
    def m2(cls):
        print('Parent class method')

    @staticmethod
    def m3():
        print('Parent static method')

class C(P):
    a=888
    def __init__(self):
        self.b=999
        super().__init__()
        print(super().a)
        super().m1()
        super().m2()
        super().m3()

>> c=C()
```

In the above example we are using super() to call various members of Parent class.

3.0.1 To Call Method of a Particular Super Class

We can use the following approaches:

```
>> super(D, self).m1()
#It will call m1() method of super class of D.
>> A.m1(self)
# It will call A class m1() method
```

```
class A:
    def m1(self):
        print('A class Method')
class B(A):
    def m1(self):
        print('B class Method')

class C(B):
    def m1(self):
        print('C class Method')
class D(C):
    def m1(self):
        print('D class Method')
class E(D):
    def m1(self):
        A.m1(self)

>> e=E()
>> e.m1()
```

Various Important Points about super():

1. Case-1: From child class we are not allowed to access parent class instance variables by using super(), Compulsory we should use self only. But we can access parent class static variables by using super().

```
class P:
    a=10
    def __init__(self):
        self.b=20

class C(P):
    def m1(self):
        print(super().a)#valid
        print(self.b)#valid
        print(super().b)#invalid
>> c=C()
>> c.m1()
```

2. Case-2: From child class constructor and instance method, we can access parent class instance method, static method and class method by using super().

```
class P:
    def __init__(self):
        print('Parent Constructor')
    def m1(self):
        print('Parent instance method')
    @classmethod
    def m2(cls):
        print('Parent class method')
    @staticmethod
    def m3():
        print('Parent static method')

class C(P):
    def __init__(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

    def m1(self):
        super().__init__()
        super().m1()
        super().m2()
        super().m3()

>> c=C()
>> c.m1()
```

3. Case-3: From child class, class method we cannot access parent class instance methods and constructors by using super() directly(but indirectly possible). But we can access parent class static and class methods.

```
class P:
    def __init__(self):
        print('Parent Constructor')
    def m1(self):
        print('Parent instance method')
    @classmethod
    def m2(cls):
        print('Parent class method')
    @staticmethod
    def m3():
        print('Parent static method')

class C(P):
    @classmethod
    def m1(cls):
        #super().__init__()--->invalid
        #super().m1()--->invalid
        super().m2()
        super().m3()

>> C.m1()
```

```
'''From Class Method of Child Class, how to call Parent Class
                                   Instance Methods and
                                   Constructors'''
class A:
    def __init__(self):
        print('Parent constructor')

    def m1(self):
        print('Parent instance method')

class B(A):
    @classmethod
    def m2(cls):
        super(B,cls).__init__(cls)
        super(B,cls).m1(cls)

>> B.m2()
```

4. Case-4: In child class static method we are not allowed to use super() generally (But in special way we can use).

```
class P:
    def __init__(self):
        print('Parent Constructor')
    def m1(self):
        print('Parent instance method')
    @classmethod
    def m2(cls):
        print('Parent class method')
    @staticmethod
    def m3():
        print('Parent static method')

class C(P):
    @staticmethod
    def m1():
        super().m1()-->invalid
        super().m2()--->invalid
        super().m3()--->invalid

>> C.m1()
```

```
''How to Call Parent Class Static Method from Child Class Static
Method by using super()'''
class A:
    @staticmethod
    def m1():
        print('Parent static method')
```

```
class B(A):
    @staticmethod
    def m2():
        super(B,B).m1()

>> B.m2()
```

Lab 4 Exercises

The objectives of this lab

- To apply OOPs concepts to implement different types of Inheritance concepts to facilitate code re-usability and modularity
- To Understand inheritance to create scalable and adaptable systems with enhancement in flexibility and extensibility in code design.

Lab 4 Assignments

1. Define a base class Shape with attributes color and filled, and a method get_info() that prints the color and whether the shape is filled or not. Create a subclass Rectangle that inherits from Shape and adds attributes length and width. Implement a method calculate_area() in Rectangle to compute the area of the rectangle.
2. Create a base class Vehicle with attributes make, model, and year, and a method display_info() to print these attributes. Create a subclass Car that inherits from Vehicle and adds an attribute mileage. Implement a method display_info() in Car to include the mileage along with the other attributes.
3. Define a base class Animal with a method sound() that prints the sound the animal makes. Create subclasses Dog and Cat that inherit from Animal and override the sound() method to print the specific sound each animal makes.
4. Create a base class Person with attributes name and age, and a method introduce() to print the name and age of the person. Create a subclass Student that inherits from Person and adds an attribute student_id. Implement a method introduce() in Student to include the student ID along with the name and age.

5. Define a base class BankAccount with attributes account_number, balance, and methods deposit() and withdraw() to add or subtract funds from the account. Create a subclass SavingsAccount that inherits from BankAccount and adds an attribute interest_rate. Implement a method add_interest() in SavingsAccount to calculate and add interest to the account balance.