PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Inheritance

Prof. Sindhu R Pai
PCPS Theory Anchor - 2024
Department of Computer Science and Engineering

Introduction

- Acquiring or obtaining the features of one type in another type.
- Allows programmers to define a new class which inherits almost all the properties(data members and methods) of existing class.
- Two ways of relationships**: Is – a relationship** and **Has-a relationship**
- Is – a relationship is also known as **parent-child relationship**
- Has – a relationship is nothing but **containership or composition or collaboration**

Inheritance

Is – a relationship: Indicates that one class gets most or all of its features from a parent class.

When this kind of specialization occurs, there are three ways in which parent and child can interact.

1. Action on child imply an action on the parent

2. Action on the child override the action on the parent

3. Action on the child alter the action on the parent

Inheritance

1. Action on child imply an action on the parent

   Example

```
class A:
        def disp(self):
                print("in disp A")

class B(A):
        pass

a1=A()
a1.disp()
b1=B()
b1.disp()
```

Output:

```
in disp A
in disp A
```

2. Action on the child override the action on the parent

Example

```
class A:
	def disp(self):
		print("in disp A")

class B(A):
	def disp(self):
		print("in disp B")
a1=A()
a1.disp()
b1=B()
b1.disp()
```

Output:

```
in disp A
in disp B
```

3. Action on the child alter the action on the parent

Example

```
class A:
        def disp(self):
                print("in disp A")
class B(A):
        def disp(self):
                A.disp(self)
                print("in disp B")

a1=A()
a1.disp()
b1=B()
b1.disp()
```

Output:

```
in disp A
in disp A
in disp B
```

Types of Is-a relationships:

1. Single level inheritance: Sub classes inherit the features of one super class.
2. Multi Level inheritance: A class is inherited from another class which is in turn inherited from another class and so on.
3. Multiple inheritance: A class can have more than one super class and inherit the features from all parent classes.
4. Hierarchical inheritance: One class serves as super class for more than one sub classes
5. Hybrid inheritance: A mix of two or more above types of inheritance. Also known as **Diamond shaped inheritance**

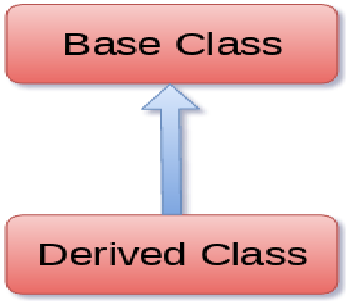
Inheritance

Benefits of inheritance:

- It allows to inherit the properties of a base class, to another class (derived) representing the real-world relationship.
- It provides the **reusability** of a code.
- Allows us to add more features to a class without modifying it.
- Transitive in nature, which means that if class B inherits from class A, then all the subclasses of B would automatically inherit from class A.
- Less development and maintenance expenses

Single Level Inheritance

```
 class BaseClass1
             #Body of base class

class DerivedClass(BaseClass1):
            #body of derived - class
```



Inheritance

Example 1: Program to create a parent class and child class objects

```
class Person:
 #Constructor
  def __init__(self, name, id_no):
    self.name = name
    self.id_no = id_no

  def Display(self):
    print(self.name, self.id_no)

#creating an object of a person
p = Person("Akash", 1001)
p.Display()
```

```
class stud(Person):
     def Print(self):
        print("stud class called")

student = stud("Madan", 103)

# Calling child class function
student.Print()

# calling parent class function
student.Display()
```

Inheritance

Example 2: Program to demonstrate the parent constructors

```
  class Person:
   def __init__(self, name, idnumber):
          self.name = name
          self.idnumber = idnumber
   def display(self):
           print(self.name)
          print(self.idnumber)
class Employee(Person):
   def __init__(self, name, idnumber, salary, desgn):
       self.salary = salary
       self.desgn = desgn
        Person.__init__(self, name, idnumber)   #observe carefully
emp = Employee('Riya', 802, 50000, "Admin")
 emp.display()
```

Example 3: Demo of the error if __init__() of the parent is not invoked

```
class A:
   def __init__(self, n='Rahul'):
      self.name = n
class B(A):
   def __init__(self, roll):
      self.roll = roll

b1 = B(23)
print(b1.name)
```

Output:

```
Traceback (most recent call last):
  File
"C:\Users\ADMIN\Desktop\inheritance.py",
line 101, in <module>     print(b1.name)
AttributeError: 'B' object has no attribute
'name'
```

Super() Function

- It is a built-in function that provides a way to access methods and properties from a parent class within a subclass.

- There might be situations where the overridden method as well as the functionality of the parent method is required. That's where super() becomes helpful.

Inheritance

Example 4:Assume the parent class has thousands of instance variables

```
class sample:
        def __init__(self,m,n,o):
                self.a=m
                self.b=n
                self.c=o
 class sample_child(sample):
        def __init__(self,m,n,o,q):
                #super().__init__(m,n,o)
                Sample.__init__(self,m,n,o)
                self.e=q
        def display(self):
                print(self.a,"--",self.b,"--",self.c,"--",self.d,"--",self.e)
s1=sample_child(1,2,3,4,90)
s1.display()
```

Example 5: Using super() a subclass can override methods or attributes from its superclass

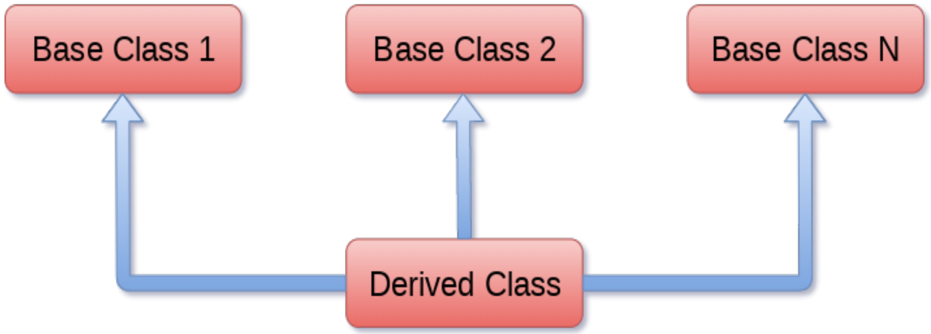
```
class ParentClass:
    def __init__(self):
        self.parent_attribute = "Parent Attribute"

    def parent_method(self):
        print("Parent Method")

class ChildClass(ParentClass):
    def __init__(self):
        super().__init__()                    # Calling the parent class constructor
        self.child_attribute ="Child Attribute"
```

Inheritance

```
def child_method(self):
    super().parent_method()
    print("Child Method")

# Creating an instance of the ChildClass

child_obj = ChildClass()

# Accessing attributes and calling methods
print(child_obj.child_attribute)
print(child_obj.parent_attribute)
child_obj.child_method()
```

Multiple inheritance

It provides the flexibility to inherit attributes and methods from more than one class



Inheritance

Example 6

```
class A:
        def disp(self):
                print("in disp A")
class B:
        def disp(self):
                print("in disp B")

class C(A,B):            #reverse the order of A and B and observe the output
        def disp(self):
                super().disp()
                print("in disp C")
c1=C()
c1.disp()
```

Note:

- When there is implicit action on class C, then the class hierarchy of A is considered.
- super() refers to only the first Parent mentioned in the subtype creation

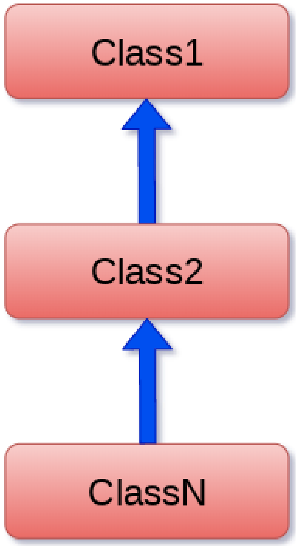
Inheritance

Multi-Level inheritance

It refers to a type of inheritance where a subclass inherits from another subclass, forming a hierarchical chain of classes.

Syntax:

```
class class1:
            <class-suite>
class class2(class1):
             <class suite>
class class3(class2):
             <class suite>
```



Example 7: Use of super() in multi level inheritance

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

    def info(self):
        return f"This is a {self.name}."

class Polygon(Shape):
    def __init__(self, name, sides):
        super().__init__(name)
        self.sides = sides
```

Overrides info of "Shape"

```
def info(self):
        return f"A {self.name} is a polygon with
{self.sides} sides."

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

class Quadrilateral(Polygon):
    def __init__(self, name):
        super().__init__(name, 4)
```

Inheritance

```
# Creating instances and accessing methods

triangle = Triangle("Triangle")
print(triangle.info())

quadrilateral = Quadrilateral("Quadrilateral")
print(quadrilateral.info())
```

Output

```
A Triangle is a polygon with 3 sides.
A Quadrilateral is a polygon with 4 sides.
```

Inheritance

issubclass() and isinstance() methods

issubclass(sub, sup)

Used to check the relationships between the specified classes.
Returns True if the first class is the subclass of the second and False otherwise.

isinstance(obj,class)

Used to check the relationship between the objects and classes.
Returns True if the object is the instance of the specified class.

Inheritance

Example 8: (use of issubclass() and isinstance())

```
class add:
   def Summation(self,a,b):
      return a+b

class mult:
   def Multiplication(self,a,b):
      return a*b

class Derived(add,mult):
   def Divide(self,a,b):
      return a//b
```

Inheritance

```
d = Derived()

print(issubclass(Derived,mult))
print(issubclass(add,mult))

print(isinstance(d,Derived))

print("Summation of  a and b:  ",d.Summation(12,20))
print("Product of a and b:   ",d.Multiplication(9,8))
print("Quotient:“  ,d.Divide(20,10))
```

```
Output:
True
False
True
Summation of  a and b:
32
Product of a and b:  72
Quotient: 2
```

Inheritance

Composition: When one object contains another object as a part or member.
Ex: Library has books

```
class Author:
   def __init__(self, name):
      self.name = name
class Book:
   def __init__(self, title, author):
      self.title = title
      self.author = author
author1 = Author("J.K. Rowling")
book1 = Book("Harry Potter and the Sorcerer's Stone", author1)
# Accessing Book and Author attributes
print(f"The book '{book1.title}' was written by {book1.author.name}")
```

Output:

```
 'Harry Potter and the Sorcerer's Stone' was written
by J.K. Rowling
```

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CDSAML & CCBD, PESU
Prof. Sindhu R Pai – sindhurpai@pes.edu
Prof. C N Rajeswari