PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. C N Rajeswari, Prof. Sindhu R Pai
Department of Computer Science
and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Unit - 4: Functional and Object Oriented programming

Iterators

Prof. C.N.Rajeswari , Prof. Sindhu R Pai

Department of Computer Science and Engineering

Iterators

PES UNIVERSITY

Iterators:

An iterator is an object that allows iteration through a sequence of elements, one at a time. It implements two main methods: **__iter__()** and **__next__()**.

__iter__():: returns the iterator object itself and is called when the iterator is initialized.

__next__():: returns the next item in the sequence.

When there are no more elements to return, it raises the **StopIteration** exception.

Iterators

Iterators lazy object or Eager object?

Lazy Evaluation: Iterators follow the principle of lazy evaluation, meaning they generate values on-demand rather than computing all values at once.

This can be memory-efficient when dealing with large datasets as it only retrieves elements as needed.

Iterators

PES UNIVERSITY

Custom Iterable Objects

The container class (like list) should support a function

1. __iter__(callable as iter(container-object)) which returns an object of a class called an iterator.

2. __next__(callable as next(iterator_object))

These two functions are interfaces which can be implemented by traversing through a container

Ex. we may visit only elements in odd position or elements satisfying a boolean condition – like elements greater than 100

Iterators

Example (Creates an object of MyContainer whose attribute mylist refers to the list a.)

```
class MyContainer:
        def __init__(self, mylist):
                self.mylist = mylist
        def __iter__(self):
                self.i=0
                return self
        def __next__ (self):
                self.i += 1
                if self.i <= len(self.mylist):
                        return self.mylist[self.i - 1]
                else:
                                raise StopIteration
```

Iterators

PES UNIVERSITY

```
a = ['apple', 'banana', 'orange', 'dates',  'cherry']

c = MyContainer(a)

for w in c :
        print(w)

c = MyContainer(a)
```

Here, observe that it creates an object of MyContainer whose attribute mylist refers to the list a

Iterators

PES UNIVERSITY

Working:

The for statement calls iter(c) which is changed to MyContainer.__iter__(c)

This __iter__ function adds a position attribute i to the object and then returns the MyContainer object itself as the iterator object.

The for statement keeps calling next on this iterable object.

The __next__ function has the logic to return the next element from the list and update the position and also raise the exception stop iteration when the end of the list is reached.

Iterators

PES UNIVERSITY

Example 2

```
class SquareNum:
    def __init__(self, n):
        self.n = n
        self.current = 0

    def __iter__(self):
        return self
    def __next__(self):
        if self.current >= self.n:
            raise StopIteration
        square = self.current ** 2
        self.current += 1
        return square
```

Iterators

```
squares = SquareNum(5)                # Using the iterable class

for num in squares:
    print(num)
```

In this ex, SquareNum is a class that generates a sequence of squares of numbers from 0 to n-1

It has __iter__() and __next__() methods implemented, making it iterable.

When instance of this class in a for loop, it iterates through the sequence, printing the squares of the numbers from 0 to 4

THANK YOU

Team Python - 2022

Department of Computer Science and Engineering

Contact Email ID's:

sindhurpai@pes.edu

cnrajeswari@pes.edu

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. C N Rajeswari, Prof. Sindhu R Pai
Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Unit - 4: Functional and Object Oriented programming

Inheritance

Prof. C.N.Rajeswari , Prof. Sindhu R Pai

Department of Computer Science and Engineering

Inheritance

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
```

Inheritance

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
```

Inheritance

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
```

Inheritance

PES UNIVERSITY

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

PES UNIVERSITY

Benefits of inheritance:

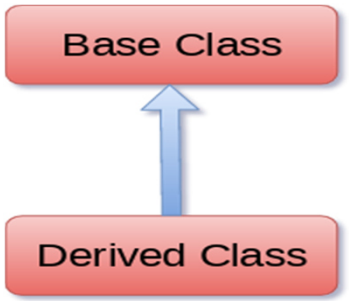
- It allows to inherit the properties of a base class, to another class (derived) representing the real-world relationship.
- It provides the **reusability** of a code.
- Allows us to add more features to a class without modifying it.
- Transitive in nature, which means that if class B inherits from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

Inheritance

PES UNIVERSITY

Single Level Inheritance

```
class BaseClass1
        #Body of base class

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



Inheritance

PES UNIVERSITY

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

PES UNIVERSITY

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()
```

Inheritance

PES UNIVERSITY

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'
```

Inheritance

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

PES UNIVERSITY

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()
```

Inheritance

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"
```

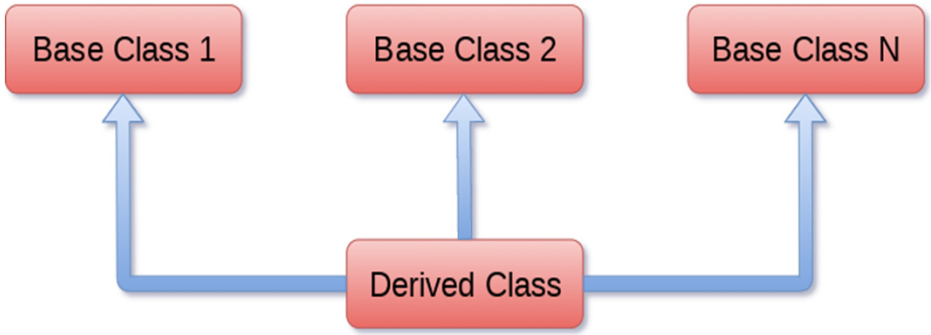
Inheritance

```
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()
```

Inheritance

Multiple inheritance

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



Inheritance

PES UNIVERSITY

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

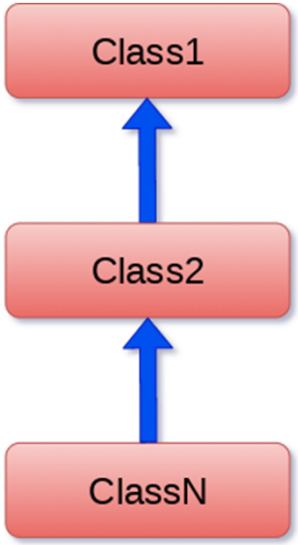
Inheritance

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

    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

PES UNIVERSITY

```
# 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.
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. C N Rajeswari, Prof. Sindhu R Pai
Department of Computer Science
and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Unit - 4: Functional and Object Oriented programming

Polymorphism

Prof. C.N.Rajeswari , Prof. Sindhu R Pai

Department of Computer Science and Engineering

PES UNIVERSITY

Polymorphism

- Polymorphism refers to having multiple forms.
- refers to the use of the same function name, but with different signatures
- allows objects of different classes to be treated as objects of a common superclass.
- This concept enables a single interface to be used for entities of different types.

Polymorphism

Runtime Polymorphism

- Python supports runtime polymorphism by using techniques like method overloading and method overriding
- Runtime polymorphism is the ability of an object to behave differently based on its actual type during program execution
- It is also known as dynamic polymorphism
- It enables the same method name to behave differently based on the specific class instance at runtime.

Polymorphism

Key Aspects of Runtime Polymorphism:

1.Inheritance:

- Runtime polymorphism is closely associated with inheritance.
- Subclasses inherit methods from their superclass, and they can provide their own implementation for these methods.

2.Method Overriding:

- Subclasses override methods from their superclass to provide their own specialized implementation.
- The method signature remains the same in both the superclass and subclass.

Polymorphism

PES UNIVERSITY

3.Dynamic Binding:

- During runtime, the appropriate method to execute is dynamically determined
- It is based on the actual type of the object invoking the method

4.Common Interface:

- Different subclasses sharing a common superclass interface
- Exhibits different behaviors based on their specific implementations.

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. C N Rajeswari, Prof. Sindhu R Pai
Department of Computer Science
and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Unit - 4: Functional and Object Oriented programming

Zip function and practice programs

Prof. C.N.Rajeswari , Prof. Sindhu R Pai

Department of Computer Science and Engineering

Zip() function

- Python zip() method takes iterable containers and returns a single iterator object, having mapped values from all the containers.
- It is used to map the similar index of multiple containers so that they can be used just using a single entity.
- The function zip is used to associate the corresponding elements of two or more iterables into a single lazy iterable of tuples.
- It does not have any callback function.

Syntax : zip(*iterators)

Zip() function

PES UNIVERSITY

Working

- The zip() function is used to combine two or more lists (or any other iterables) into a single iterable.
- Elements from corresponding positions are paired together.
- The resulting iterable contains tuples, where the first element from each list is paired together, the second element from each list is paired together, and so on.

Zip() function

Example 1 Consider the code below which pairs the two lists:

```
m=[1,2,3]
n=[4,5,6]
l_new=[]
for i in range(len(m)):
        l_new.append((m[i],n[i]))
print(l_new)
```

The above can be done very easily and with less code using zip. We can observe the same output.

```
print(list(zip(m,n)))
```

Output

```
[(1, 4), (2, 5), (3, 6)]
[(1, 4), (2, 5), (3, 6)]
```

Zip() function

Example 2: Combining two iterables (tuples) into a single iterable

```
name = [ "Sudha", "Suma", "Sara", "Asha" ]
roll_no = [ 404, 112, 393, 223 ]

# using zip() to map values
mapped = zip(name, roll_no)

print(set(mapped))
```

Output:

```
{('Sudha', 404), ('Suma', 112), ('Sara', 393), ('Asha', 223)}
```

Zip() function

Example 3: Combining two iterables (lists) into a single list

```
a = [1, 2, 3, 4, 5]

b = list(map(lambda x : x * x * x, a))

print(a)

print(b)

print(list(zip(a, b)))
```

Output:

```
[1, 2, 3, 4, 5]
[1, 8, 27, 64, 125]
[(1, 1), (2, 8), (3, 27), (4, 64), (5, 125)]
```

Zip() Function

Example 4: Zipping list and tuple with unequal size

```
#Define lists for 'persons',and a tuple for 'ages'
persons = ["Baskar", "Monica", "Riya", "Madhav", "John",
"Prashanth"]
ages = (35, 26, 28, 14)

#lists along with the 'ages' tuple
zipped_result = zip(persons,ages)

print("Zipped result as a list:")
for i in list(zipped_result):
  print(i)
```

Zip() Function

Example 5: Zipping list with unequal size.(Two ways)

```
lis1 = [1,2,3]
lis2 = [4,5,6,7]
print(list(zip(lis1,lis2)))
```

The same can be acheived using the directory comprehension as:

```
print({k:v for k,v in (zip(lis1,lis2))})
```

Output

```
[(1, 4), (2, 5), (3, 6)]
{1: 4, 2: 5, 3: 6}
```

Practice programs

1.Given list of circle areas all in five decimal places, round each element in the list up to its position decimal places, meaning round up the first element in the list to one decimal place, the second element to two decimal places, the third element to three decimal places and so on.

2.Given
list= [1,2,3,4,5]
Chars= [‘a’, ’b’, ’c’, ’d’, ’e’]
Output the following.

('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]

3.The following are the scores of chemistry exam . Filter out those who passed with scores >75. Use an appropriate function

Practice programs

4.You are given a list of fruits names. Using lambda and map function print the list of fruit names starting with ‘A’
Lst=[‘Orange’, ’Apple’ , ’ Mango’, ‘ Apricot’]
Output=[‘Apple’, ‘Apricot’]

5.Using reduce function find the sum of the digits of the digits of the given number

For ex: n = '1729’
output= summation of 1,7,2,9=19

6. Using the min and reduce function find the smallest number in a given list of 10 numbers.

THANK YOU

Team Python - 2022

Department of Computer Science and Engineering

Contact Email ID's:

sindhurpai@pes.edu

cnrajeswari@pes.edu

Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #103

Problem Solving on classes, objects, Polymorphism and Inheritance

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M(Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Practice Programs

Try solving these problem statements. Solutions for a few are available in this link:

https://drive.google.com/file/d/1DbYfPusibwR-W-Nhi2bcGbzQIb0VrLY_/view?usp=drive_link

1. Implement the Shape Hierarchy by creating a "Shape" type as the parent type. Add new types ' Rectangle', 'Circle' and 'Triangle' inheriting from 'shape'. Take the **radius of the circle**, length and width of rectangle, base, height, side1, side2 and side3 for a triangle from the user. Add methods in the sub types to calculate area and perimeter. Create instances of the respective types and test all these calculations.

2. Create a base class 'Vehicle' with attributes - price, model and year. Derive named classes such as Car and Motorcycle from Vehicle. Each subclass should have additional attributes specific to Car and Motorcycle respectively. Create two instances of Car and two instances of Motorcycle and print the instances directly by calling a print function.

3. Create a Bank_Account type. Add balance as its instance variable. Add separate instance methods to deposit the amount and withdraw the amount from the account separately. Add appropriate functionality in withdraw method to check for minimum balance before withdrawing the amount and display proper message. Create two instances of Bank_Account. Initial balance is always 0 for every instance. Test all these functionality in the driver/test code.

4. Create a new type called Home with attributes - num_rooms and num_stories. Create an object of Home by passing these values. Print the object of Home type.
This must print "The house has ___ rooms and ____ stories"

5. Create a type called Complex with real and imaginary parts. Create two objects of this type. Perform addition of these two using +, subtraction using -, multiplication using * and true division using /. Write a complete code to fit into the given driver/test code below.

```
c1 = Complex(1, 4)
c2 = Complex(6, 1)
print(c1)
print(c2)
print("----------------------")
print(c1 + c2)
print(c1 - c2)
print(c1 * c2)
print(c1 / c2)
```

6. Create an Employee type with details like – id, name, age. Create a few instances of Employee. Print the number of objects created. Delete any two objects and again print the count of objects.

7. Create a class for books with attributes like title, author, and genre. Implement a simple library catalog system to add and display book information.

8. Develop a system with classes for employees and departments. Include functionalities like displaying employee details and assigning employees to departments.

9. Create a class for temperature conversion with methods to convert Celsius to Fahrenheit and vice versa. Test the class with sample temperature values.

10. Design classes for basic animals (cat, dog) with attributes like name and sound. Simulate their behavior by displaying the name and the sound they make.

For interested students only:

11. Implement a simple ToDo list application with classes for tasks. Include functionalities like adding tasks, marking them as complete, and displaying the list.

12. Create a library management system with classes for books, patrons, and transactions. Implement functions to handle book checkout, return, and overdue fines.

13. Build a multimedia player application with classes representing different types of media (audio, video, images). Implement features like play, pause, stop, and volume control.

14. Develop a reservation system for a hotel with classes for rooms, guests, and reservations. Implement methods to check room availability, make reservations, and calculate total costs.

15. Design a simple online shopping system with classes for products, customers, and shopping carts. Include features like adding items to the cart, checkout, and order history.

16. Build a music player program where 'Audiofile' is the base class. Implement sub classes 'MP3File' and 'WAVFile' inheriting from 'AudioFile'. Both classes should have a 'play' method but provide different functionalities for playing MP3 and WAV files.

17. Create a base class 'Vehicle' with a method 'calculate_speed'. Derive subclasses 'Car' and 'Bike' from 'Vehicle'. Override the 'calculate_speed' method in each subclass to calculate their specific speed based on different parameters

-END-