BCA SEM-VI

Subject: Python Programming (BCA-601)

Created by: Smit Trivedi (Asst.Prof BCA/PGDCA College Idar)

UNIT 3

Topic:

List

Tuples

File Handling

Classes

Inheritance and Polymorphism

List (What is List)?

List is a Sequence of items separated by commas and items are enclosed in square braces [ ]. In list items may be different data type & it is mutable..

Exploring List: A list in Python is used to store the sequence of various types of data. Python lists are mutable type its mean we can modify its element after it created..

A list can be define as below.

```
L1 = [“SMIT", 105, "USA"]
L2 = [1, 2, 3, 4, 5, 6]
print(type(L1))
print(type(L2))
```

- List Class: All the list objects are the objects of the list class in Python. Use the list() constructor to convert from other sequence types such as tuple, set, dictionary, string to list.

```
nums=[1,2,3,4]
print(type(nums))
mylist=list('Hello')
print(mylist)
nums=list({1:'one',2:'two'})
print(nums)
nums=list((10, 20, 30))
print(nums)
nums=list({100, 200, 300
print(nums)
```

Loop List

Looping through list items in Python refers to iterating over each element within a list.

We do so to perform the desired operations on each item. These operations include list modification, conditional operations, string manipulation, data analysis, etc.

Python provides various methods for looping through list items, with the most common being the **for loop.**

We can also use the **while loop** to iterate through list items, although it requires additional handling of the loop control variable explicitly i.e. an index.

- **Loop Through List Items with For Loop**

  A for loop in Python is used to iterate over a sequence (like a list, tuple, dictionary, string, or range) or any other iterable object.

  It allows you to execute a block of code repeatedly for each item in the sequence.

Syntax:

```
for item in list:
    # Code block to execute
```

Task 1: lst = [25, 12, 10, -21, 10, 100]

Task 2: Fruits = [’orange’, ’apple’, ’banana’, ’kiwi’, ’mango’]

Loop Through List Items with While Loop

- A while loop in Python is used to repeatedly execute a block of code as long as a specified condition evaluates to "True".
- We can loop through list items using while loop by initializing an index variable, then iterating through the list using the index variable and incrementing it until reaching the end of the list.
- **Syntax**

```
while condition:
    # Code block to execute
```

An index variable is used within a loop to keep track of the current position or index in a sequence, such as a list or array. It is generally initialized before the loop and updated within the loop to iterate over the sequence.

- Example

  we iterate through each item in the list "my_list" using a while loop.

  We use an index variable "index" to access each item sequentially, incrementing it after each iteration to move to the next item −

  Task:

  my_list = [1,2,3,4,5]

  index = 0

Loop Through List Items with Index

- An index is a numeric value representing the position of an element within a sequence, such as a list, starting from 0 for the first element.
- **Example**
- lst = [25, 12, 10, -21, 10, 100]
- indices = range(len(lst))
- for i in indices:
- print ("lst[{}]: ".format(i), lst[i])

List Comprehension

- A list comprehension in Python is a concise way to create lists by applying an expression to each element of an iterable. These expressions can be arithmetic operations, function calls, conditional expressions etc.

Syntax

- [expression for item in iterable]
- Example:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = [num ** 2 for num in numbers]
print (squared_numbers)
```

Join Two List

- **There are several ways to join, or concatenate, two or more lists in Python.**
- **This can be achieved using various methods, such as concatenation, list comprehension, or using built-in functions like extend() or + operator.**
- **One of the easiest ways are by using the + operator.**

Example:

```
# Two lists to be joined
  L1 = [10,20,30,40]
  L2 = ['one', 'two', 'three', 'four']
# Joining the lists
 joined_list = L1 + L2
# Printing the joined
 list print("Joined List:", joined_list)
```

Using append() Function

```
  list1 = ['Fruit', 'Number', 'Animal']
  list2 = ['Apple', 5, 'Dog']
for element in list2:
    list1.append(element)
    print("Joined List:", list1)
```

Using Extend() Function

```
# List to be extended
# List to be added
list1.extend(list2)
print("Extended List:", list1)
```

Updating the elements of the list

- You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to ele.ments in a list with the **append() method.**

Tuples

Tuple is a Sequence of items separated by commas and items are enclosed in round braces ().

In list items may be different data type & it is immutable.

Major Difference:

List	Tuple
Mutable	immutable
⇩	⇩
Can be Change	Can't Change

Create Tuple:

Syntax : Variable _ name & Tuple_ name = (elements……)

a = (1,2,3,4,5)

a = ( ) -> Empty Tuple

a = (1) -> a is not a tuple

a= (1,) -> tuple with single elements

a= 1,2,3 print (a)

- **Create a basic Tuple**

  Let us first create a basic Tuple with integer elements and then move towards Tuples within a Tuple..

```
# Creating a Tuple
mytuple = (20, 40, 60, 80, 100)
# Displaying the Tuple
print("Tuple = ",mytuple)
# Length of the Tuple
print("Tuple Length= ",len(mytuple))
```

Accessing Tuple

-7 -6 -5 -4 -3 -2 -1

a= (10,20,30,40,50,60,70)

0 1 2 3 4 5 6

Giving index Range of items [:]

Index value start from 0

That Example:

a = [0:4]

a = [:5]

a = [3:]

- **Python Access Tuple using a Positive Index**
- Using square brackets we can get the values from tuples in Python.
- t = (10,20,30,40,50,60,70)

Access Tuple using Negative Index

t = (10,20,30,40,50,60,70)

Basic operations on Tuples

- Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.
- In fact, tuples respond to all of the general sequence operations we used on strings in the prior.

Python Expression	OutPut	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

- Length of Tuple

  To determine how many items a tuple has, use the len() function:

  Example

  Print the number of items in the tuple:

  thistuple = ("apple", "banana", "cherry“, “Mango”)
  print(len(thistuple))

  OUTPUT : 4

- Concatenation Two Tuples:
- To join two or more tuples you can use the + operator:
- **Example**

```
tuple1 = ("a", "b" , "c“)
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

OUTPUT

- **Repetition of Tuple**
- If you want to multiply the content of a tuple a given number of times, you can use the * operator:

Example

Multiply the fruits tuple by 2:

```
fruits = ("apple", "banana", “Mango")
mytuple = fruits * 2
print(mytuple)
```

- Iteration of Tuple

  for x in (1, 2, 3):

  print (x)

  Output

File Handling

What is File?

Python files are like containers where you can store your Python code.

They end with a **. py** extension. Think of them as folders for organizing your code.

What is File Handling Operation?

A file is a named location used for storing data.

For example, main.py is a file that is always used to store Python code.

Python provides various Functions to perform different file operations, a process known as File Handling.

Various Types of File Handling.

(1) Create File (2) Reading File (3) Writing File (4) Deleting

Mode	Description
r rb r+	Read-only mode. Read-only in binary mode. Read and write mode.
rb+ w	Read and write in binary mode. Write mode.
wb w+	Write in binary mode. Write and read mode
wb+ a ab	Write and read in binary mode. Append mode. Append in binary mode.
a+ ab+	Append and read mode. Append and read in binary mode.
x xb x+	Exclusive creation mode. Exclusive creation in binary mode. Exclusive creation with read and write mode.
xb+	Exclusive creation with read and write in binary mode.

How to Create File

- Creating a text file with the command function "w“

  f = open("myfile. txt", "w") #This "w" command can also be used create a new file but unlike the the "x" command the "w" command will overwrite any existing file found with the same file name.

example:

```
f = open(“C:\\users\\Lenovo\\Desktop\\python\A.txt”,”x”)
Print(“File  Created Successfully”)
```

Read File

- This mode opens the text files for reading only. The start of the file is where the handle is located.

```
f=open("C:\\Users\\lenovo\\OneDrive\\Desktop\\Python program\A.txt","r")
print(f.read(30))
```

The readline() method:

This function reads a line from a file and returns it as a string. It reads at most n bytes for the specified n. But even if n is greater than the length of the line, it does not read more than one line.

```
print(f.readline())
```

Writing files

- This mode opens the file for writing only. The data in existing files are modified and overwritten. The start of the file is where the handle is located.

```
f=open("C:\\Users\\lenovo\\OneDrive\\Desktop\\Python program\A.txt","w")
f.write("Learn coding\n Smit | Mit")
print("Data write Successfully...!")
```

Deleting File

- Step 1 - Importing a Library. import os. We have only imported os which is needed.
- Step 2 - Creating a File. We created a file by function . ...
- Step 3 - Opening and Deleting File. We have open the file by using the.

Example: **import os**

```
if os.path.exists(("C:\\Users\\lenovo\\OneDrive\\Desktop\\Python program\A.txt"):

os. Remove(("C:\\Users\\lenovo\\OneDrive\\Desktop\\Python program\A.txt")

else:
  print("File not available…!")

  print("File Deleted Successfully….!")
```

File Copying

```
Try:
 With open("C:\\users\\lenovo\\Desktop\\Python\\A.txt")as f2:
 With open("C:\\users\\lenovo\\Desktop\\Python\\B.txt","w")as f3:
  for i in f2:
   f3. write(i)
    except:
       print("file not available …… plz create first!")
   else:
     f2.close()
         print("file closed…!")
```

Classes

Python is an **object-oriented programming language.**

which means that it is based on principle of OOP concept.

The entities used within a Python program is an object of one or another class.

Like that: numbers, strings, lists, dictionaries, and other similar entities of a program are objects of the corresponding built-in class.

What is a Class in Python?

A class is a code template for creating objects. Objects have member variables and have behavior associated with them. In python a class is created by the **keyword class.**

Syntax: Class Class_name:

 # Variables

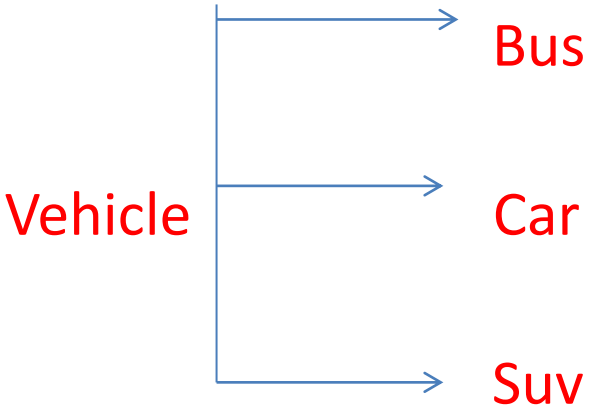
 # Methods

Object

An **object** is refered to as an instance of a given Python class. Each object has its own attributes and methods, which are defined by its class**.**

When a class is created, it only describes the structure of obejcts. The memory is allocated when an object is instantiated from a class.

Syntax: Obj_Name = Class_Name



In the above figure, **Vehicle** is the class name and **Car**, **Bus** and **SUV** are its objects.

Empty Class Define:

```
Class A:
   Pass
#obj = A()
```

```
Class A:
  def_init_(self):   [Python Default constructor]
  age = 20
   print(age)
#obj = A()
```

Self parameter

Self parameter in python refers to the current instance of a class.

It is used to access and modify The class's attributes and methods.

How to use Self:

self must be the first parameter in any function in the class.

self is used within a class definition to refer to the class's own instance variables and methods.

self is automatically passed as the first argument when a method of an object created from a class is called.

Syntax: class Person:

```
def __init__(self, name):
self.name = name person = Person("Stt")
print(person.name)
```

```
# class
class Maruti:
   # init method
   def __init__(self, company, model):
      # self
      self.company = company
      self.model = model
# creating instances for the class Maruti
Maruti_one = Maruti('Altok10', '2024')
Maruti_two = Maruti('Swift', '2025')
# printing the properties of the instances
print(f“ Maruti One: {Maruti_one.company}")
print(f“ Maruti Two: {Maruti_two.company}")
```

Modifying the property of a class

- To modify the value of a class attribute, we simply need to assign a new value to it using the class name followed by dot notation and attribute name.
- Example:

  In the below example, we are initializing a class variable called **empCount** in Employee class.

  For each object declared, the __init__() method is automatically called. This method initializes the instance variables as well as increments the empCount by 1.

```
class Employee:
   # class attribute
      empCount = 0
      def __init__(self, name, age):
       self.__name = name
       self.__age = age
     # modifying class attribute
      Employee.empCount += 1
      print ("Name:", self.__name, ", Age: ", self.__age)
     # accessing class attribute
     print ("Employee Count:", Employee.empCount)

   e1 = Employee("", )
print()
e2 = Employee("", )
```

Inheritance

What is Inheritance

inheritance is a mechanism that allows a class to inherit attributes and methods from another class.

It's a key concept in object-oriented programming.

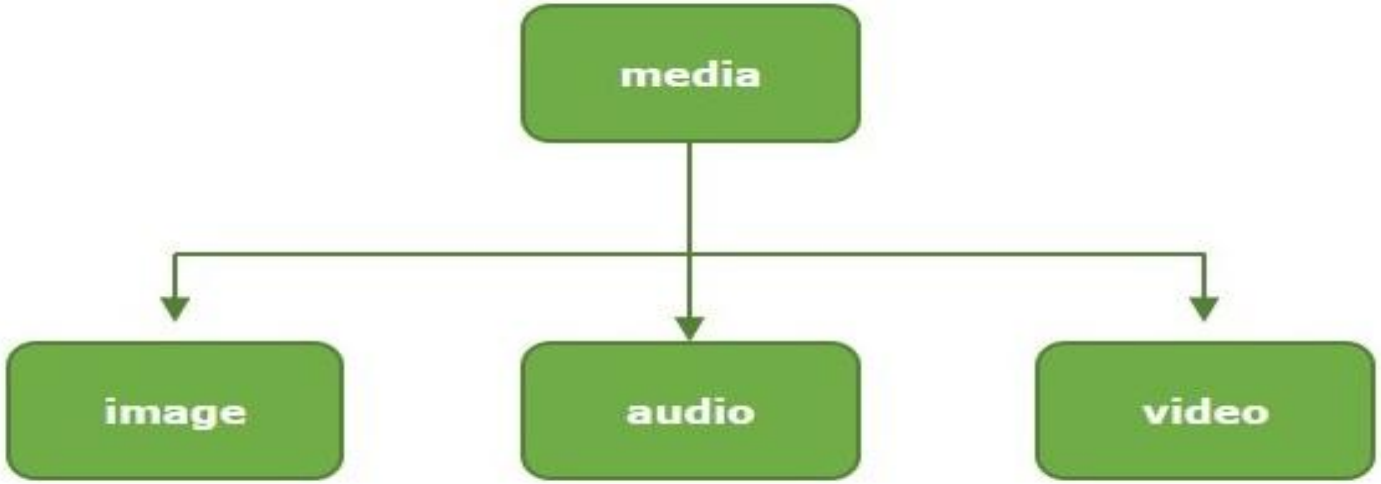
How does inheritance work?

The class that inherits is called the derived class or child class.

The class that is inherited from is called the base class or parent class.

The base class provides common attributes and methods

The derived class inherits and extends the functionality of the base class.



The class that inherits another class is called a **child class** and the class that gets inherited is called a **base class or parent class**.

For example, Car IS a vehicle, Bus IS a vehicle, Bike IS also a vehicle. Here, Vehicle is the parent class, whereas car, bus and bike are the child classes.

Creating a Parent Class: The class whose attributes and methods are inherited is called as parent class. It is defined Just like other classes using the **class is keyword.**

Syntax: class ParentClassName:

{class body}

- Creating a Child Class:

  Classes that inherit from base classes are declared similarly to their parent class, however, we need to provide the name of parent classes within the parentheses.

  Syntax: class SubClassName (ParentClass1[, ParentClass2, ...]):

  {sub class body}

Types of Inheritance:

1) Single Inheritance

2) Multiple Inheritance

3) Multilevel Inheritance

4) Hierarchical Inheritance

5) Hybrid Inheritance

Encapsulation

Encapsulation is the process of bundling attributes and methods within a single unit. It is one of the main pillars on which the object-oriented Programming paradigm is based.

Python Provides access to all the variable and methods globally.

By using Encapsulation, we can restrict the variables and methods access globally by making it Private or Protected.

Note: Single Underscore (Protected)

Double Underscore (Private)

Polymorphism

- The term **polymorphism** refers to a function or method taking different forms in different contexts. Since Python is a dynamically typed language, polymorphism in Python is very easily implemented.
- If a method in a parent class is overridden with different business logic in its different child classes, the base class method is a polymorphic method.

There are Three ways to implement polymorphism in Python.

(1 Duck Typing

(2 Operator Overloading

(3 Method Overriding