UNIT IV - Python functions, Modules & packages.

1. Define function.

→ In Python, a function is a block of code that performs a specific task and can be called multiple times throughout your program.
functions are defined using the 'def' keyword followed by the function name, parantheesis, and a colon.

2. List any 4 built-in function.

→ 1) int(a, base)
2) float()
3) ord()
4) hex()
5) oct()

3. What is user defined function & why it is needed.

→ User defined function -
- When we create our own functions to perform a particular task are called as ~~fo~~ user defined functions.

4. List different user ~~desio~~ defined arguments.

→ 1) required arguments
2) keyword arguments
3) default arguments
4) variable-length arguments

5. Explain all function arguments -

→ 1) ~~These arguments~~ Required arguments.

- These arguments are passed to a function in positional order.
- Here, the number of arguments in the function call should match exactly with the functional definition.

· Example -

2) Keyword arguments -

- By using these arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows us to skip arguments or place them out of order because the python interpreter is able to use the keywords provided to match the values with parameteres.

3) Default arguments -

- A default argument is an argument that assumes a default value if a value is not provided in function call for that argument.

4) Variable - Length arguments -

- In some cases we may need to process a function for more arguments than specified while defining the function.
- These arguments are called as variable - length arguments ~~and~~ and are not named in function definition.

6. Explain local scope -

→ Local scope -

- local scope variables can only be accessed within its block

```
a = 10
def function():
    print("Hello")
b = 20
function()
print(a)
print(b)
```

7. Explain global scope -

→ Global scope -

- The variables that are declared in the global scope can be accessed from anywhere in the program.
- Global variables can be used inside any functions.
- We can also change the global variable value.

```
print Msg = "This is delared globally"
    def function():
        print(Msg)
    function()
```

9. Define Module -

→ A module a file that contains a Python code that we can use in our program.

10. Write syntax of importing module.

→ import module-name

11. Define Package.

→ • A Python module may contain several classes, functions, variable, etc. whereas a Python package can contains several module.

• In simpler terms a package is folder that contains various modules as files.

• A package is directory containing modules or subpackages.

12. What is role of __int__.py file in package?

→

UNIT-V Object ORIENTED PROGRAMMING

1. Define class in Python.

→ · A class is a block of statement that combine data & operations, which are performed on the data, into a group as a single unit and act as a blueprint for the creation of objects.

2. Define object in python.

→ · A unique instance of a data structure that is defined by it's class.
· An object compires both data members and methods.
· class itself does nothing but real functionality is achieved through their objects.

3. Explain 'self' method with an example.

→ Self method:

· Class methods must have an extra first parameter in the method definition.
· We do not give a value for this parameter when we call the method, python provides it.
· If we have a method that takes no arguments, then we still have to have one argument.
· Example -

```
class car:
    def get(self, color, style):
        self.color = color
        self.style = style
    def put(self):
        print(self.color)
        print(self.style)
c = car()
c.get('Brio', 'Red')
c.put()
```

```
op|-  Brio
      Red
```

5. Explain '__init__' method with example.

→ In Python, the __init__() method is called the constructor and is always called when an object is created.

Syntax - def __init__(self):
    # body

```
class student:
    def __init__(self,rollno,name,age):
        self.rollno = rollno
        self.name = name
        self.age = age
        print("student object is created")
p1 = student(11,"Ajay",20)
print("Roll No =", p1.Rollno)
print("Name =", p1.name)
print("age =", p1.age)
```

5. Define Inheritance.

→ • The mechanism of designing and constructing classes from other classes is called inheritance.
• Inheritance is the capability of one class to derive or inherit the properties from some another class.
• There are 5 types of inheritance -
1) Single inheritance
2) Multiple inheritance
3) Multilevel inheritance
4) Hierarchical inheritance
5) Hybrid inheritance.

6. Explain properties of inheritance.

→ Properties of inheritance -
1) It allows to derive a new class from old class.
~~2) The old class is called as derived class (subclass or childclass).~~
2) The old class is called as base class or parent class.
3) The new class is called as derived class / subclass or child class.
4) Inheritance allows subclasses to inherit all the variables and methods to their parent class.
5) Reusability of code is best advantage of inheritance.

7. Explain single inheritance with an example -

→ • It includes only one parent class and one child class.

• Syntax -

```
class SubClassName (ParentClass1 [, Parent class2, ...,])
    "Optional class documentation string"
class-suite
```

Create two classes named Student() and Test() and student information and marks.

```
class Animal:
    def eat (self):
        print ("I can eat")

class Dog (Animal):
    def bark (self):
        print ("I can bark")

dog1 = Dog()
do1.bark()
dog1.eat()
```

8. Multiple inheritance -

→ · It includes more than one parent class and the child class inherits all properties of both parent class.

· Syntax -

```
class Base1:
    Body of the class
class Base2:
    Body of the class
Class Derived (Base1, Base2):
    Body of the class
```

```
class class1:
    def m(self):
        print ("In class1")

class class2 (class1):
    def m (self):
        print ("In class2)
```

g. Multilevel inheritance -

→ · In this inheritance a child class can act as parent class for another child class.

```
· Class A :
    # properties of class A
  Class B(A):
    # class B inheriting property of class A
    # more properties of class B
  class C(B):
    # class C inheriting property of class B
    # thus, class C also inherits properties of class A
    # more properties of class C

class C1:
    def display1(self):
        print("class C1")
class C2(C1):
    def display2(self):
        print("class C2")
class C3(C2):
    def display3(self)
        print("class C3")

S1 = C3()
S1.display3()
S1.display2()
S1.display1()
```

10. Explain hierarchical inheritance -

→ • It includes multiple inheritance from same parent class.

• example -

```
class Parent():
    def fun1 (self):
        print("fun1 from parent class")

class child1 (Parent):
    def fun2 (self):
        print ("fun2 from child1 class")

class child2 (Parent):
    def fun3 (self):
        print ("fun3 from child3 class")

obj1 = Child1()
obj2 = child2()
print ("from child1:")
obj1. fun1()
obj1. fun2()
print ("\n from child2:")
obj2. fun1()
obj2. fn 3()
```

11. Explain hybrid inheritance with an example -

→ • If multiple inheritance occurs in a program then it is called hybrid inheritance.

• example -

```
class Parent1():
    def fun1(self):
        print("fun1 from parent")

class Parent2():
    def fun2(self):
        print("fun2 from parent")

class Child1(Parent1):
    def fun3(self):
        print("fun 3 from child1")

class Child2(Parent1, Parent2):
    def fun4(self):
        print("fun 4 from child 2")

obj1 = Child1()
obj2 = Child2()
print("from child1: Single Inheritance")
obj1.fun1()
obj1.fun3()
print("from child 2: Multiple Inheritance")
obj2.fun1()
obj2.fun2()
obj2.fun4()
```

12. Explain Method Overloading with an example.

→ • Methods in Python can be called with zero, one or more parameters.

• This process of calling the same method in different ways is called method overloading.

• Two methods cannot have the same name in Python; hence method overloading is a feature that allows the same operator to have different meanings.

```
• Class Student:
      def __init__(self, a, b):
          self.a = a
          self.b = b
      def sum(self, a, b, c):
          s = a + b + c
          return s
  s1 = Student(10,20)
  print("sum is:", s1.sum(10,20,30))
```

13. Method overriding -

→ • Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is ~~ready~~ already provided by one of its super-class(s) or parent class(s).

• When a method in a subclass has the same name, same parameters or signature and same return type (or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

```
class Parent:
    def fun1 (self):
        print ("I parent class with fun1 method")

class Child (Parent):
    def fun1 (self):
        print ("Child class with fun2 method")

obj = child()
obj.fun1()
print ("child class has overriden fun1 method of P.class")
```

14. Explain Data Hiding -

→ • Data Hiding is a concept which underlines the hiding of data or information from the user.
• Data hiding is also known as information hiding.
• It includes object details such as data members, internal work.
• Data hiding excludes full data entry to class members and defends object integrity by preventing unintended changes.
• Data hiding also minimize system complexity for increase robustness by limiting interdependencies betn slw requirements.

```
class Solution:
    __private Counter=0

    def sum(self):
        self.__private Counter += 1
        print (self.__private Counter)

count = Solution()
count.sum()
count.sum()

print (count._Solution__private Counter)
```

15. What are advantages and disadvantages of Data hiding.

→ Advantages -

1) It helps to prevent damage or misuse of volatile data by hiding it from the public.
2) The class objects are disconnected from the irrelevant data.
3) It isolates objects as the basic concept of oop.
4) It increases the security against hackers that are unable to access important data.

Disadvantages -

1) It enables programmers to write lengthy code to hide important data from common clients.
2) The linkage between the visible and invisible data makes the objects work faster, but data hiding prevents this linkage.

15. What are advantages and disadvantages of Data hiding.

→ Advantages -

1) It helps to prevent damage or misuse of volatile data by hiding it from the public.
2) The class objects are disconnected from the irrelevant data.
3) It isolates objects as the basic concept of oop.
4) It increases the security against hackers that are unable to access important data.

Disadvantages -

1) It enables programmers to write lengthy code to hide important data from common clients.
2) The linkage between the visible and invisible data makes the objects work faster, but data hiding prevents this linkage.

16. Explain Data Abstraction -

→ • The process by which data and functions are defined in such a way that only essential details can be seen and unnecessary implementations are hidden is called. Data Abstraction

• The main focus of data abstraction is to separate the interface and the implementation of the program.

• Syntax -

```
from abc import ABC
class ClassName (ABC):
```

```
from abc import ABC, abstractmethod
class car (ABC):
    def mileage (self):
        pass

class Tesla (car):
    def mileage (self):
        print ("mileage is 30 kmph")
class Suzuki (car):
    def mileage (self):
        print ("mileage is 25 kmph")
t = Tesla()
t.mileage()

s = suzuki()
s.mileage()
```

UNIT VI -

File Handling and Exception Handling

M T W T F S S
Page No.
Date:
YOUVA

1. Define file in python -

→ · It is a kind of permanent storage consuming less memory but satisfies the user basic needs.
· A file can be of any type -
- Audio
- video
- text
- image.

2. Explain open() function with all modes.

→ *open() function -
· Before performing any operation on the file like read or write, first we have to open that file.
· for this, we should use python's inbuilt function open().
· But at the time of opening, we have to specify the mode the mode, which represents the purpose of the opening file.

```
f = ope("demo.txt", "r")
content = f.read()
print(content)
f.close()
```

Modes -

1) r = Open file a read mode, if file not present then it will raise the error.

2) W = Open file in write mode, if file not present it will create a file and open it in write mode.

3) a = append new contents at the end of existing file, If file not present it will create file and open it in append mode.

4) t = text mode, it is default mode in which file opens.

5) b = Open file in a binary mode.

3. What is difference between write, append & x mode.

→ Write (r) : Opens a file for writing, creates the file if it does not exist.

append (a) : Opens a file for appending, creates the file if it does not exist.

Create (x) : Creates the specified file, returns an error if the file exists.

4. List different file method.

→ 1) Write() -

- The write() method writes any string to an open file.
- Python strings can have binary data and not just text.

2) open()
3) read()
4) rename()
5) remove()
6) close()

1) file.no()
2) file.seek()
3) file.tell()
4) file.truncate()
5) file.readlines()

5. Explain how to rename and delete file with an example.

→ rename() -

The rename() method takes two arguments, the current filename and the new filename.

```
import os
os.rename("TY", "TYCW")
```

remove() -

You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.

```
import os
os.remove("TYCOO")
```

6. Define Directory in python.

→ Directory -

- A directory or folder is a collection of files and subdirectories.
- Python has the os module that provides us with many useful methods to work with directories (and files as well)
- Directories are a way of storing, organizing, and separating the files on a computer.
- The directory that does not have a parent is called a root directory.

7. Explain how to create directory and check current working directory.

→ **Creating new directory:**

- os.mkdir(name) method to create a new directory.
- The desired name for the new directory is passed as the parameter.
- By default it creates the new directory in the current working directory
- If the new directory has to be created somewhere else then that path has to be specified and the path should contain forward slashes instead of backward ones.
- os.mkdir(Drive name/directory name)

```
import os
print ("OUTPUT")
os.mkdir ('ABC')
os.mkdir ('E:/ABC')
print ("Directory ABC created successfully in E drive")
```

* Current working directory -

- os:getcwd() can be used.
- It returns a string that represents the path of the current working directory.
- os.getcwdb() can also be used but it returns a byte string that represents the current working directory.
- Both methods do not require any parameters to be passed.

```
import os
print ("String format: ", os.getcwd())
print ("Byte String format: ", os.getcwdb())
```

8. Explain two types of error in Python..

→ 1) Syntax error -

- As the name suggests this error is caused by the wrong syntax.
- It leads to the termination of the program.

2) Exceptions error -

- Exceptions are raised when the program is syntactically correct, but the code resulted in an error.
- This error does not stop the execution of the program, however, it changes the normal flow of the program.

Syntax error :

```
amount = 10000
if (amount > 2ggg)
print ("You are eligible to get loan")
```

exception error -

```
marks = 10000
a = marks 10
print (a)
```

9. Explain 'try' and 'except' with an example.

→ Try and except

- Try and except statements are used to catch and handle exceptions in Python.
- Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside clause.

```
a = [1, 2, 3]
try:
    print("Second element = %d" %(a[1]))
    print ("fourth element = %d" %(a[3]))
except:
    print("An error occurred")
```

10. Explain catching specific exception with an example.

→ * Catching specific ~~spec~~ exception :

- A try statement can have more than one except clause, to specify handlers for different exceptions.
- Syntax -

```
try :
    # statement (s)
except IndexError :
    # statement (s)
except ValueError :
    # statement (s)
```

```
def fun(a):
    if a < 4:
        b = a/(a-3)
    print ("Value of b = ", b)
try :
    fun (3)
    fun (5)
except ZeroDivisionError :
    print ("ZeroDivisionError Occured and Handled")
except NameError :
    print ("NameError Occurred and Handled")
```

11. Explain 'raise' statement with an example.

→ 'raise' statement -

- The raise statement allows the programmer to force a specific exception to occur.
- The sole argument in raise indicates the exception to be raised.
- This must be either an exception instance or an exception class (a class that derives from Exception)

```
try:
    raise NameError ("Hi there")
except NameError:
    print ("An exception")
    raise
```