GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING

Unit-V

FILES, MODULES, PACKAGES

PART-B

Introduction to Files:

- Most of the programs till used are transient (i.e.) they run for a short period of time and produce some output but when they and their data disappear. If we sun the program again it shorts with a clean slate.
- So that the idea of "Persistent "programs arise they keep data in a permanent storage. If they end and restart again the data will be present in one of the Disk memory.
- Examples of persistent storage are,
    - Files - to maintain data by reading and writing.
    - Database - Organized for storing data.

Advantages of Files:

- Data are stored permanently in File.
- It is possible to update data.
- Data stored in a file can be used in other programs.
- Useful to store huge amount of data.

FILES:

- File is a collection of logically related data stored in a permanent storage such as Hard Disk.
- Files can be of two types.
    (i) Binary File - It can store data in the form of bytes. (i.e.) 0's and 1's.
        - It is used to store text, images, audio and video.
    (ii) Text File - It contains sequence of characters.

Text Files: [Explain how Operations can be performed on Files.]

- Text file contains sequence of characters stored in permanent storage like hard drive.
- It contains only text.
- Each line in text file is terminated with a special character called End of Line (EOL) character. It ends the current line.
- In python the new line character ("/n") is considered as EOL by default.

Operations on Text File:

The basic operations that can be performed on text files are,

(i) Open
(ii) Read
(iii) Write
(iv) Close

- To perform any operation on a file it must be opened first.

(ii) Opening a File:

- To open a new file or to open an existing file open () function is used.
- Syntax:

Fileobj = Open (File Name, File Mode)

Where,

File Name - Name of the file. Text file must have the extension " · txt "(i.e.)

One · txt
File Name Extension.

File Mode - the purpose of a File.

FIES MODES; / ACCESS MODE;

- It determines the mode in which a file is opened also it represent for what purpose a file is opened.
- Depends on this access mode only we can able to perform an operation.
- The default access mode for files in python is read (r) mode.
- The different types of access modes are as follows :

Si. No	File Mode	Description	File pointer position

1.	r	Read only. (Default Mode)	Beginning of the File.
2.	r+	Read and write	Beginning of the File.
3.	rb	Read only in Binary File.	Beginning of the File.
4.	rb +	Read and write in Binary File	Beginning of the File.
5.	w	Write only. Overwrites the file if exists. Otherwise create a new File.	Beginning of the File.
6.	w+	Read and Write	Beginning of the File
7.	wb	Write only in Binary	Beginning of the File
8.	wb +	Read and Write in Binary File.	Beginning of the File .
9.	a	Append only. If file not exist it creates new file.	End of the File.
10.	a+	Append and Read	End of the File.
11.	ab	Append only in Binary	End of the File.
12.	ab +	Append and read in. Binary File.	End of the File

Ex: f1. Open ("demo. txt ", "w ") ≠ Write Mode

F1. Open ("demo. txt ", "r ") ≠ Read Mode

(ii) Reading from a File:

- To read the content of a file it must be opened in read (r) mode.
- There are three functions available to read contents from a file.
  - read () - It reads 'n' bytes. In string format. If not specified it reads the entire file.
  - Read line () - It reads a line of string. End of line is identified by "/n" newline character.
  - read lines () - It reads all the lines in a file and return them as a list of

Strings.

- Syntax :
  File object · read (n) n → No. of Bytes to be read.
  File object · readline (n)
  File object · readlines (n)

Example:

Consider a text file named as "demo · txt "with the following contents.

"demo · txt "

Python programming Joy of python Basics of python

→ f1 · read (10) - python pro

→ f1 · readline () - python programming

→ f1 · readlines () - python programming Joy of python Basics of python.

(iii) Writing to a File:

- To write data in a file it must be opened in write (w) mode or append (a) mode.
- When we open a file in write mode it will overwrite into the file if it already exists. (i.e.) previous data will be erased.
- The method used to write data in a file is of two ways:
  (i) Write () - It write a single line of text.
  Syntax: fileobj · write (str)
  (ii) Write lines () - It writes multiple strings in the file.
  Syntax: fileobj · write lines (L) → can be a list of strings
- Example :
  l1 = ["Apple ", "Orange "]
  f1 · writelines (l1) → Apple Orange.
  f1 · write ("Banana ") → Apple Orange Banana

(iv) Closing a File:

- If all the operations are completed in a file it can be closed.
- Close () function is used to close a file and frees the memory space used by the file.
- Syntax : fileobj · close ()
- Example : f1 · close ()

- After a file is closed we cannot able to do any operation on that file. If we want to do so it must be opened again.

Example:

```
i1 = Open (“Demo · txt “, “w “ )
i1 · write (“Welcome python/n “ )
i1 · write (“Programming “ )
i1 · close ()
i1 · open (“Demo, txt “, “r “ )
Print (“File contains: “, i1 · read () )
i1 · close ()
```

Output:

```
File Contains:
Welcome python
Programming
```


FILE POSITIONS; / RANDOM ACCESS ON FILES:

- When a file is opened in read, write or append mode file pointers will be in beginning or End of the file.
- So if there is a need to perform operation in middle of a file random accessing is needed.
- To do random access on a file the file pointer must be moved to other positions except beginning and End. This can be achieved by two functions. Seek () and tell (0.

(i) Seek () - used to change the current file position.

Syntax: Seek (offset, from position)

Where, offset - No. of bytes to be mored.

from position - reference position from where the bytes are mored. It Can be,

0 → Beginning of File.
1 → Current position
2 → End of File.

Example: Seek (3, 0) - From the beginning of a File 3 bytes will get read.

Seek (5, 1) - From the current position 5 types will get read.

- If the 2nd argument of seek is 0 then 1st argument should not be Negative.
- If the 2nd argument of seek is 2 then 1st argument should be Negative.

Ex: Seek (-3, 2) - From the end of a File 3 bytes will get read.
Seek (5, 2) - It's Invalid because. From the End of a File again there will be no data to read.
Seek (-3, 0) - It's Invalid because from the beginning there is no data before to read.

(iii) Tell () - It returns the current position of a file.

- The current position is calculated from the beginning of the file.

Syntax:

file obj · tell ()

Ex:

f1 = open ("new · txt ", "w ")

f1 · write ("Hello Hai ")

Print (f1 · tell ()) → O/P: 9

Example Program:


```
f1 · Open ( " demo · txt " , " w " )    ≠ Open a new File " demo · txt
f1 · Write ("Hello World " )            ≠ Write data into a file.
f1 · close ()
f1 · Open ("demo · txt ", "r " )
Print ("File Contents: ", f1 · read (8) )    ≠ read 8 bytes
Print ("Current File position: ", f1 · tell () )
f1 · Seek (0, 0)          ≠ More pointer to beginning of File.
Print ("Now File Position is: ", f1 · tell () )
f1 · Seek (-6, 2)         ≠ From End of File the 5th byte gets pointed.
Print ("Content is ", f1 · read () )
f1 · Seek (0, 0)          ≠ File pointer moved to beginning.
f1 · Seek (5, 0)          ≠ from beginning move pointer after 5 bytes.
Print ("Content is ", f1 · read () )
f1 · close ()
```

Output:

File Contents: Hello WO

Current File Position: 8

Now File Position is: 0

Content is World

Content is Hello.

RENAMING AND DELETING FILES;

Python os module provide methods that helps to perform file – processing operations such as renaming and deleting files. To use this module it must be imported first. Then the needed functions can be called using it.

(i) rename ():

- It is used to renames an existing file with other name.
- It takes two arguments the current File name and the new File name.
- Syntax :

  OS · rename (Current FN, New FN)
- Example:

  Import os

  Os · rename (“One · txt “, “two · txt “)
- File named one · txt will be renamed as two · txt.

(ii) remove ():

- It is used to delete an existing file.
- It takes the file to be deleted as argument.
- Syntax:

  OS · remove (file name)
- Example:

  import os

  OS · remove (“two · txt “) ≠ File named two · txt will be deleted.

FILENAMES and PATHS;

- Files are organized into Directories also called as Folders. Every running program has a current directory which is the default directory for most operations.
- The Module named as “Os “(i.e.) Operating system provides functions for working with files and directories.
- If a file is opened in read mode python interpreter looks for its current directory.
- To access a file present in a directory, file path is required.
- File path is the location where a file is stored in the computer system.
- Two types of path names are there,
- They are,

(i) Relative Path - It represent the file along with its current working Directory.

(ii) Absolute path - It represents the file from its home directory till its current Working Directory.

- Example: Consider a file named " Demo · txt "

Is stored inside "/ home / Drill / Geetha "

Location. The user is working from Directory "Geetha "then,

Relative path - / Geetha / Demo · txt.

Absolute path - / home / Drill / Geetha / Demo · txt

Methods in os Module:

1. mkdir () - Used to create new directory.

   Syntax: OS · mkdir ("Name ")

2. Chdir () - Used to change current working directory.

   Syntax: OS · Chdir ("New DirName ")

3. getcwd () - Displays the current working directory.

   Syntax: OS · getcwd ()

4. rmdir () - It deletes a Directory.

   Syntax: OS · rmdir ("Dir. Name ")

5. abs path () - Displays the absolute path of a file.

   Syntax: OS · path · abs path ("Filename ")

6. exists () - Checks Whether a file or directory exists.

   Syntax: OS · path · exists ("Filename ")

7. isdir () - Checks the argument is a Directory or not.

   Syntax: OS · path · isdir ("DirName ")

8. isfile () - Checks the argument is a File.

   Syntax: OS · path · isfile ("Filename ")

9. listdir () - It list all the file and Directions present in a Directory.

   Syntax: OS · listdir ("Dir. Name ")

Example: Output:

```
import    os                                    / home / geetha
Cwd = os · getcwd () `                          True
Print (cwd)                                     Files are:
Print (os · path ·isdir ())                     One · txt Demo · py
Print ("Files are: ", os · listdir () )         new · py
```

FORMAT OPERATOR; [Explain about Format operators in python.]

- To write data into a file, write () method is used. The arguments of write () must be a string.
- If other values are written in a file it must be converted to strings.
- The easiest way to do that is with str () method.

  Example: f1 · Open ("Demo, txt ", "w ")
  f1 · write (5) ≠ Type Error: 5 is not
  f1 · write (str (5)) a string.

  → No Error · 5 is converted to string using str ().

- An alternative to this format operator can be used.
- The symbol used is " % ".
- The % operator is used to format a set of variables enclosed in a tuple along with a format string. Which contains one or more format sequences.
- Format sequence specify how the second operand is formatted.
- The result is a string.

  Ex: " % *s* " % 15 O/P: '15' as a string.
  ↓ ↓ ↓ String
  Format Format Second
  Sequence Operator Operand

Different format Strings;

$\%d$, $\%i$ - Integer value.

$\%f$ - Floating point value

$\%c$ - Character value

$\%s$ - String value

Example:

>>> "There are $\%d$ colours in rainbow " % 7

There are 7 colours in rainbow.

>>> "There are $\%s$ colours in rainbow " % " "seven "

There are seven colours in rainbow

>>> "value of PI is 3.140000

>>> "value of PI is %5 · 2f " % 3.14

Value of PI is 3.14

Where " % 5 · 2f " → 5 - Output field

2 - Precision Digits after the decimal point.

- If there are more than one format sequence in the string, the second argument must be a tuple.
- Each format sequence is matched with an element of the tuple in order. With the number of sequence and type of sequence.

Example:

1) >>> "Reg No %d with Name %s obtained

% · 1f $CGPA$ % (15," Anu " , 8.5)

Reg No 15 with Name Anu obtained 8.5 CGPA.

2) >>> " % d %d %d " % (10,15)

Type Error: Not enough arguments. [One argument missing]

3) >>> " % d " % " Hai "

Type Error: Incorrect Data type for argument.

COMMAND LINE ARGUMENTS:

- It is a way to get input for a python program.
- In this input can be directly sent as an argument.
- When the program is running under command prompt then inputs can be passed directly in the command.
- Python " sys " module provides access to any command line arguments through sys · argv
  → Syc · argv - List of command line arguments.
  → Len (sys · argv) - Number of Command line arguments.
  → Sys · argv [0] - represent program Name.

Example:

```
Import sys

n = len (sys · argv )

Print ("No. Of arguments: ", n)

arg = str (sys · argv)

Print ("Arguments are: ", arg )
```

Print ("Program Name: ", sys · argv [0])

Output:

>>> Python demo · py one two

No. of arguments: 3

- Arguments are: ["demo · py " · "One ", " two "]

Program Name: demo · py

ERRORS AND Exceptions;

ERRORS:

- Errors are normally referred as bugs in the program. They are almost always the fault of the programmer.
- The process of finding and eliminating errors is called Debugging.
- Two types of Errors:

  (i) Syntax Errors

  (ii) Run time Errors.

(i) Syntax Errors:

- The python Interpreter finds the syntax errors when it passes the source program.
- Once it find a syntax error python will exit the program without running anything.
- Commonly occurring syntax errors are,
  → Putting a keyword at wrong place
  → Misspelling the keyword.
  → Incorrect Indentation
  → Forgetting the symbols such as comma, brackets, Quotes.

(ii) Run time Errors:

- The Run time errors are not detected while passing the source program but will occur due to some logical mistake.
- The program may unexpectedly exit during runtime error.
- Examples of run time error are,
  → Division by zero
  → Use of an identifier which is not defined.
  → Trying to access a file which does not exists.

EXCEPTIONS IN PYTHON;

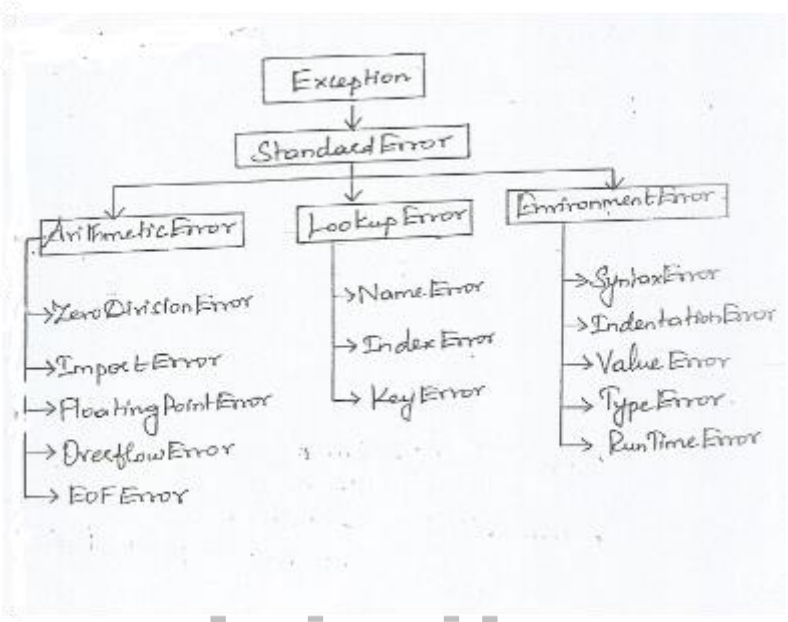
Definition:

An exception is an event, which occurs during the execution of a program that change the normal flow of the program. Also run – time error is known as Exception.

Example: Divide by zero, Index out of Bound.

Exception Hierarchy:

The following explains python's Built – in exception list.



Exception - Base class for all exceptions.

Standard Error - Base class for all Built – in exceptions.

Arithmetic Error - Base class for Numeric Exceptions.

- Zero Division Error - Raised when division or modulo by zero takes Place.
- Import Error - Raised when an import statement fails.
- Floating point Error - Raised when floating point calculation fails.
- Over flow Error - Raised when calculation exceeds maximum limit.
- EOF Error - Raised when End of File is reached.

Lookup Error - Base class for all Lookup Errors.

- Name Error - Raised when identifier not found.
- Index Error - Raised when on index is not found in a sequence.
- Key Error - Raised when a key is not found in Dictionary.

Environment Error - Base class for all exceptions that occurs outside the python Environment.

- Syntax Error - Raised when there is an error in python syntax.
- Indentation Error - Raised when indentation is not specified property.
- Value Error - Raised when invalid values are specified in Function.
- Type Error - Raised when an operation done is invalid for the Specified Data type.
- Runtime Error - Raised when a generated error does not fall into any category.

Exception Handling: [Appraise the use of try block and except block in python]

- When a python script raise an exception it must be handled immediately otherwise it terminates abnormally.
- Exception can be handled by using "try … except … else "block.

(i) try Block - It contains the code cause an exception. So it is known as Exception generation Block.

(ii) Except Block - If exception occurs, this block will handle the exception. So it is known as Exception Handles. A python code can have number of except block.

(iii) else block - If there is no exception this block will get executed.

(iv) Finally block - It must be written at last for clean-up process. It will get executed whether exception occurs or not.

Exceptions can be handled in the following ways:

Try block with single except block:

- In this only one exception type can be handled.

Syntax:

```
try:
    ≠ Stmt
except  Exception:
```

≠ Stmt

else:

≠ Stmt.

Example:

```
x = 10
y = 0
try:
    z = x/y ≠ Exception Occurs here
except      ZeroDivisionError:
    Print ("Zero Division Error Occurs " )
else:
    Print ("Quotient = ", z )
```

Output:

Zero Division

Error Occurs.

(iii) Try block with Multiple except block:

- Multiple except block is used to handle move than one type of exception separately.
- When an exception raised exception handless are searched in order for correct match and it will get executed.
- When no match is found, program will be terminated.

Syntax:

```
try:
   ≠ Stmt
except       Exception1:
  ≠ Stmt
except       Exception2:
  ≠ Stmt
   except      Exception N:
     ≠ Stmt
  else:
```

Example:

```
x = 10
      y = 0
      try:
          z = x/y
      except     Zero Division Error:
          print ("Zero Division Error Occurs " )
      except Name Error:
        Print ("Name Error Occurs " )
      except Index Error:
```

≠ Stmt

Print (“Index Error Occurs “)

else:

Print (“Quotient = “, z)

O/P:

Zero Division Error Occurs.

Explanation:

Here zero Division Error is raised so from the multiple Exception only it’s except block gets executed.

(iii) Except clause with Multiple Exceptions:

- Using this multiple exception can be handled using a single except block.
- To achieve this exception list must be added along with except statement.
- If the raised exception name is present in the exception list it will be handled. Otherwise, program will be terminated.

Syntax:

try:

≠ Stmt

except (Exception1, Exception2, Exception N):

≠ Stmt

else:

≠ Stmt

Example:

x = 10

y = 0

try:

z = x/y

except (Zero Division Error, Name Error, Index Error):

Print (“Exception is Handled “)

else:

Print (“Quotient = “, z)

O/P:

Exception is handled.

(iv) Except Clause with No Exception: (or) Catch All Exception

- Using this all types of Exceptions can be handled inside a single except block.
- But programmer cannot able to identify the root cause of the exception.
- If this block is used it must be written at last.

Syntax:

```
try:
  ≠ Stmt
except:
  ≠ Stmt
else:
  ≠ Stmt
```

Example:

```
x = 10
y = 0
try:
  z = x/y
except:
print ("Exception is Handled " )
else:
  Print ("Quotient = ", z )
```

Output:

Exception is handled.

(v) Except clause with finally Block:

- Finally block is used at last to write clean up codes.
- It will get executes whether exception is raised or not.

Syntax:

```
try :
  ≠ Stmt
except     Exception:
  ≠ Stmt
finally:
  ≠ Stmt
```

Example:

```
x = 10
y = 0
try :
    z = x/y
except:
    print ("Exception Occurs " )
else:
    Print ("Quotient = " · z )
finally:
   Print ("END " )
```

Output:

Exception Occurs

END

Raising on Exception:

- Exception can be raised from a function or block to be handled.
- Syntax:
  raise Exception (argument)
- Example:

```
def    positive (n):
       if (n < 1):
              raise    Exception (n)
       else :
              Print ("Positive " )
try:
    Positive (-3)
except Exception:
    Print ("Negative " )
```

Output:
Negative.
from the function positive an exception is raised and it is handled.

PACKAGES IN PYTHON: [Explain about packages in python]

Definition:

Package is a way to group modules, functions, classes etc. It is a Hierarchical Directory Structure that consist of sub packages.

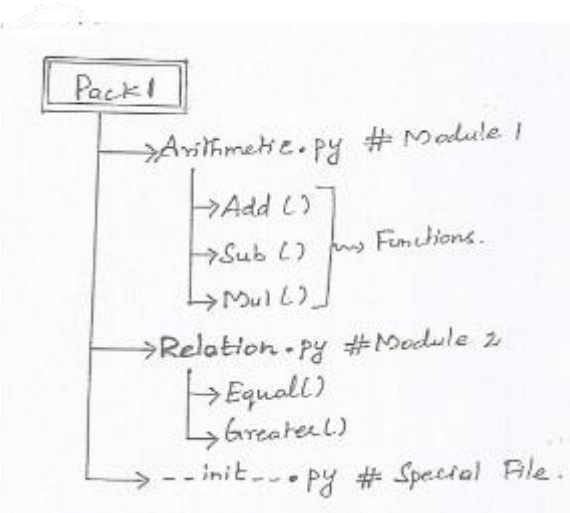
In python a package must contain a special file named as "– init -- · py ", Interpreter will identify python package with ordinary directory.

Steps to create python Package:

1. Create a directory in a Disk drive and name it with package name.
2. Define the needed modules within the directory.
3. Create the special file -- init -- · py inside the directory.
4. From a python program access the package. Accessing can be done using the following syntax :
   Package Name · Module Name · Function Name (arg)

Example:

In the following example a package named as "pack 1 "was created with two Modules, Arithmetic & Relation. The outline is as follows:



Step1:

Create a Directory named as “Pack 1 “in a Disk drive.

Step 2:

Within pack 1 directory create two modules named as Arithmetic · py and Relation · py

Arithmetic · py ≠ Module 1

```
def     Add ( x , y ) :
        Print ( “ Sum = “ , ( x + y ) )
def     Sub ( x , y ) :
        Print ( “ Difference = “ , ( x – y ) )
def     Mul ( x , y ) :
        Print ( “ Product = “ , ( x * y ) )
```

The above module Arithmetic contains three functions named as Add (), Sub () and Mul ().

Relation · py ≠ Module 2

```
def     Equal ( x , y ) :
        if ( x = = y ) :
            print ( " Values are Equal " )
        else :
            print ( " Values are Not Equal " )
def     Greater ( x , y ) :
        if ( x > y ) :
            print ( " x is Greater " )
        else :
            print ( " y is Greater " )
```

The above module Relation contains two functions named as Equal () and Greater ().

Step 3:

Create the special file named as – init -- · py inside pack 1 directory and include the following statements.

-- init --· py ≠ Special File.


```
from    Pack  I    import    Arithmetic
from    Pack  I    import    Relation
```

Step 4:

Finally create a python program to access the functions.

import Pack I ≠ import package Pack I

Pack I · Arithmetic · Add (21, 35) ≠ Call Add function in

Pack I · Arithmetic · Mul (4, 5) Arithmetic Module

Pack I · Relation · Greater (35, 25) ≠ Call Greater function in Relation Module.

Output:

Sum = 56

Product = 20

X is Greater.

ILLUSTRATIVE PROGRAMS;

1. Word Count ≠ Program used to count No. of words in a File.

```
f1. Open ("File · txt " · "w " )
f1 · Write ("Hai Welcome to python programming " )
f1 · close ()
f1 · Open ("File · txt ", "r " )
Words = f1 · read ()
NumWords = Words · Split ()
Print ("No. Of Words: " , len, (Num) Words))
f1 · close ()
```

Output:

No. of Words: 5

Explanation:

In the above program a file named as "File · txt "Is opened and the content was written. Again open the same file in read mode and read the contents using read (), then to find no. of words use the split () function to split the contents in file as words. Finally use len () function to display length. (i.e.). It will display the no. of words in a file.

2. Copy File: ≠ Copy the content from one file to another file.

```
f1 = Open ("One · txt ", "w " )
f1 · Write ("Hai Welcome to python programming " )
f1 · close ()

f1 · Open ("One · txt ", "r " )
data = f1 · read ()
f1 · close ()
f2 = Open ("two · txt ", "w " )
f2 · write (data)
f2 · close ()
```

Output:

File – One · txt

```
Hai     Welcome to
Python programming
```

File – two · txt

```
Hai Welcome to python
programming
```

Explanation:

- Open a new file one · txt in write mode and write the needed content within it and close it.
- Again, open the same file again and read the content within a variable and close it.
- Open a new file two · txt in write mode using write function write the data in the variable.
- Finally close the file.