II. COMMAND LINE ARGUMENTS

- It is possible to pass some values from the command line to python programs when they are executed. These values are called command line arguments control program from outside instead of hard coding.
- The command line arguments are handled using sys module. We can access command- line arguments via the sys.argv This serves two purposes –

Example 1

Consider the following script command.py

```
import sys
program_name =
sys.argv[0] arguments =
sys.argv[1:] count =
len(arguments)
print(program_name)
print(arguments)
print(count)C
```

>>>C:\Python30>python.exe command.py

Hello

World 11

www.binils.com

- sys.argv is the list of command-line arguments.
- len(sys.argv) is the number of command-line arguments.
- Here sys.argv[0] is the program name.

III. ERRORS AND EXCEPTIONS ERRORS

Errors or mistakes in a program are often referred to as bugs. The process of finding and eliminating the errors is called debugging. Errors can be classified into three major groups:

- Syntax Error
- Runtime Error
- Logical Error

i) Syntax Error

- Python will find these kinds of errors when it tries to parse your program, and exit with an error message without running anything.

Syntax errors are mistakes in the use of the Python Language, and are analogous to spelling or grammar mistakes in a language like English

Common Python Syntax Errors include:

- Leaving out a Keyword
- Putting a keyword in a wrong place
- Misspelling a Keyword
- Incorrect Indent
- Unnecessary Spaces
- Empty Spaces

ii) Runtime Error

If a program is syntactically correct that is free from syntax errors, however the program exits unexpectedly it is due to runtime error. When a program comes to halt because of runtime error, then it has crashed.

Common Python Runtime Errors include:

- Division by Zero
- Performing an operation on different data type
- Using an identifier which has not been defined
- Accessing a list element which doesn't exist
- Trying to access a file which doesn't exist

Example
a=10
b=0
c=a/
b

www.binils.com

iii) Logical Error

- Logical Errors are the most difficult one to fix. They occur when the program runs without crashing but it produces an incorrect result. The error is occurred by a mistake in program logic.

Common Python Logical Errors include:

- Using the wrong variable name
- Indenting a block to the wrong level
- Using Integer division instead of float division
- Getting Operator Precedence wrong (Priority)

HANDLING EXCEPTION

Exception

An Exception is an event which occurs during the execution of a program that disrupts the normal flow of the program's instructions. If a python script encounters a situation it cannot cope up, it raises an exception

HOW TO HANDLE EXCEPTION?

There are four blocks which help to handle the exception. They are

- try block
- except statement
- else block
- finally block

i) try block

- In the try block a programmer will write the suspicious code that may raise an exception. One can defend their program from a run time error by placing the codes inside the try block.

Syntax:

```
try:
   #The operations here
```

www.binils.com

ii) except statement

- Except statement should be followed by the try block.
- A single try block can have multiple except statement.
- The except statement which handles the exception.
- Multiple except statements require multiple exception names to handle the exception separately.

```
except Exception 1:
   #Handle Exception 1
except Exception 2:
   #Handle Exception 2
```

iii) else block

If there is no exception in the program the else block will get executed.

Syntax:

```
else:
   #If no Exception, it will execute
```

iv)finally block:

A finally block will always execute whether an exception happens or not the block will always execute.

```
finally:
   #Always Execute
```

Syntax:

```
try:
   #The operations here
except Exception 1:
   #Handle Exception 1
except Exception 2:
   #Handle Exception 2
else:
   #If no Exception, it will execute
finally:
   #Always Execute
```

(i) Write a python program to write a file with exception handling.

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    f.seek(0)
    print(f.read())
except IOError:
    print("File not Found")
else:
    print("File not Found")
    f.close()
finally:
     print("Close a file")
```

.binils.com

(ii) Write a python program to read a file which raises an exception

Assume test.txt is not created in the computer and the following program is executed which raises an exception.

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    f.seek(0)
    print(f.read())
except IOError:
    print("File not Found")
else:
    print("File not Found")
    f.close()
finally:
     print("Close a file")
```

a) Except Clause with no exception

An except statement without the exception name, the python interpreter will consider as default 'Exception' and it will catch all exceptions.

Syntax:

```
try:
    #The operations here
except:
      #Handles Exception
else:
    #If no Exception, it will execute
finally:
    #Always Execute
```

www.binils.com

b) Except clause with multiple Exception

An except statement can have multiple exceptions, We call it by exception name

Syntax:

```
try:
    #The operations here
except (Exception 1, Exception 2):
      #Handles Exception
else:
    #If no Exception, it will execute
finally:
    #Always Execute
```

(i) Write a python program to read a file with multiple exceptions

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    print(f.read())
except (IOError, ValueError,
ZeroDivisionError):
    print("File not Found")
else:
    print("File not Found")
    f.close()
finally:
     print("Close a file")
```

c) Argument of an Exception

An exception can have an argument which is a value that gives additional information about the problem. The content of an argument will vary by the exception.

```
try:
   #The operations here
except Exception Type, Argument:
     #Handles Exception with Argument
else:
   #If no Exception, it will execute
finally:
   #Always Execute
```

binils.com

d) Raising an Exception

You can raise exceptions in several ways by using raise statement

Syntax:

```
raise Exception, Argument:
```

Example:

```
def fun(level):
        if level<10:
                raise "Invalid Level", level
fun(5)   #raise an Exception
fun(11)
```

TYPES OF EXCEPTION

There are two types of Exception:

- Built-in Exception
- User Defined Exception

i) Built-in Exception

There are some built-in exceptions which should not be changed.

The Syntax for all Built-in Exception

```
except Exception_Name
```

S.No	Exception Name	Description
1	Exception	It is the Base class for all Exceptions
2	ArithmeticError	It is the Base class for all Errors that occur on numeric calculations.
3	ZeroDivisionError	Raised when a number is divided by zero
4	IOError	Raised when an Input or Output operation fails such as open() function. When a file is not exist in the folder
5	TypeError	Raised when operation or function is invalid for a specified data type.
6	ValueError	Raised when built-in function for a data type has valid arguments and it has invalid values.
7	RuntimeError	Raised when a generated error does not fall into any category.
8	KeyboardInterrupt	Raised when the user interrupts program execution by pressing Ctrl+C
9	FloatingPointError	Raised when floating calculation Fails.

S.No	Exception Name	Description
10	AssertionError	Raised in case of failure of assert statement.
11	OverFlowError	Raised when a calculation exceeds maximum limit for a numeric types.
12	StandardError	Base class for all built-in exception except 'StopIteration and SystemExit'
13	StopIteration	Raised when next() method of an iteration does not exist
14	SystemExit	Raised by the sys.exit() function
15	SyntaxError	Raised when there is an error in Python Syntax
16	IndentationError	Raised when Indentation is not specified properly
17	AssertionError	Raised in case of failure of assert statement.

www.binils.com

USER DEFINED EXCEPTION

In Python a user can create their own exception by deriving classes from standard Exceptions. There are two steps to create a user defined exception.

Step-1

A User Defined Exception should be derived from standard Built-in

Step-2

Exceptions. After referring base class the user defined exception can be

used in the program

```
class NetworkError(RuntimeError):
def __init__ (self,arg):
                self.args=arg
try:
    raise NetworkError("Bad host name")
except NetworkError,e:
    print(e.args)
```

ASSERTION

An assertion is a sanity check which can turn on (or) turn off when the program is in testing mode.

- The assertion can be used by assert keyword. It will check whether the input is valid and after the operation it will check for valid output.

Syntax:

```
assert(Expression)
```

Example

```
def add(x):
        assert(x<
        0)
        return(x)
add(10)
```

www.binils.com

FILE MANIPULATIONS

File manipulation means change or access the content in the file.

- File Positions
- Renaming and Delete a File
- Directories in Python

1. File Positions

There are two methods to access the positions of the file.

1. tell()
2. seek ()

i) tell() method

This method is used to tell the current position within a file. It starts from the beginning of the file and this method followed by read() and write() method.

Syntax:

```
file_object.tell()
```

www.binils.com

Example

```
f=open("F:/Python/test.txt","r")
print(f.tell())          ; #0th position
print(f.read())
print(f.tell())          ; #Last Position
f.close()
```

Output

```
0
This is the First Line in the file
This is the Second Line in the file
73
```

ii) seek() method

This method is used to change the current file position.

Syntax:

```
file_object.seek(offset, from)
```

- seek() method set the file's current position at the offset.

- The default argument of offset is 0. The offset represents the number of the bytes to be moved.

The from argument represents three values.

0 → represents the beginning of the file

1→ represents the current position as reference 2→ represents the end of the file

Program to use read(), tell(),seek() methods

sample.txt - Problem solving and python programming

```
f1=open("F:/Python/sample.txt","r")
str=f1.read(10)
print("read string is : ",str)
position=f1.tell()
print("current file position:",position)
position=f1.seek(0,0)
str=f1.read(10)
print("again read string is :",str)
f1.close()
```

www.binils.com

Output:

```
read string is: Problem
So current file position
                              :
10
```

RENAMING AND DELETING A FILE

Two file processing operations are there, they are

- rename() method
- remove() method

i) rename() method

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

```
os.rename(current_filename, new_filename)
```

Syntax:
Exampl
e

```
import os
os.rename("test.txt","new.txt ")
```

ii) remove() method

The remove() method is used to delete the file. The argument contains file name.

Syntax:

```
os.remove(filename)
```

Example:

```
import os
os. remove("new.txt ")
```

www.binils.com

DIRECTORIES IN PYTHON

All files are contained within various directories. The os module has several methods to create, remove and change directories.

S.No	Name	Syntax	Description	Example
1	mkdir()	os.mkdir("new_dir")	This method is used to create a directory	os.mkdir("test")
2	chdir()	os.chdir("new_dir")	This method is used to change a directory	os.chdir("new_dir")
3	getcwd()	os.getcwd()	This method is used to display current directory	os.getcwd()
4	rmdir()	rmdir('dir_name')	This method is used to remove a directory	os.rmdir('new_dir')

FORMAT

The argument of write has to be a string, so if we want to put other values in a file, we have to convert them to strings. The easiest way to do that is with str:

f=open('stringsample.txt','w

') f.write(5)

#TypeErr

or f.write(str(5))

An alternative is to use the format operator, %. The first operand is the format string, which contains one or more format sequences, which specify how the second operand is formatted. The result is a string.

Example:

var=8

print("The Value is : %d"%var)

Output:

The Value is : 8

Some of the format strings are.

Sl No	Conversion	Meaning
1	d	Signed integer decimal.
2	i	Signed integer decimal
3	o	Unsigned octal.
4	u	Unsigned decimal.
5	x	Unsigned hexadecimal (lowercase).
6	X	Unsigned hexadecimal (uppercase).
7	e	Floating point exponential format (lowercase).
8	E	Floating point exponential format (uppercase).
9	f	Floating point decimal format.
10	F	Floating point decimal format.

UNIT V (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

I. FILES

- File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).
- Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

There are many different types of files.

- Data File
- Text File
- Program File
- Directory File

Types of files

Python provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python,

- Text files
- Binary files (written in binary language,0s and 1s).

www.binils.com

1. Text Files:

- Text files are structured as a sequence of lines where each line includes a sequence of characters.
- Each line is terminated with a special character, called the End of Line character.

Examples for Text files:

- **Web standards:** html, XML, CSS, JSON etc.
- **Source code:** c, app, js, py, java etc.
- **Documents:** txt, tex, RTF etc.
- **Tabular data:** csv, tsv etc.
- **Configuration:** ini, cfg, reg etc.

2. Binary files

- All binary files follow a specific format. Because all the binary files will be encoded in the binary format, which can be understood only by a computer or machine.
- Binary files can only be processed by application.
- For handling such binary files we need a specific type of software to open it.

Binary files can be used to store text, images, audio and video.

Example:

- Executables
- Databases
- Application data
- Configuration files
- Device driver files

FILE OPERATIONS

When the user want to do some operations in the file there is a sequence of steps that should be followed.

- Open the File
- Read or Write data in the file(perform operation)
- Close the File

1. OPENING A FILE The open() method

www.binils.com

- Python has a built-in function open() to open a file. This function returns a file object and has two arguments which are file name & opening mode. The opening modes are reading, writing or appending.
- *Syntax:*

```
file_object=open("filename", "mode")
```

where file_object is the variable to add the object and the mode tells the interpreter which way the file will be used.

Example

```
f=open("E:/Python/text.txt", 'r')
f=open("E:/Python/text.txt",
'w')
```

FILE OPENING MODES

S.NO	Modes	Description
1	**r**	**Read Mode:** Read mode is used only to read data from the file.

2	**rb**	Open a file for the read-only mode in the binary format.
3	**r+**	Opens a file for both reading and writing.
4	**rb+**	Open a file for read and write only mode in the binary format.
5	**w**	**Write Mode:** This mode is used when you want to write data into the file or modify it. Remember write mode overwrites the data present in the file.
6	**wb**	Open a file for write only mode in the binary format.
7	**w+**	Opens a file for both reading and writing.
8	**wb+**	Opens a file for both writing and reading in binary format.
9	**a**	**Append Mode:** Append mode is used to append data to the file. Remember data will be appended at the end of the file pointer.
10	**ab**	Open a file for appending only mode in the binary format.
11	**a+**	**Append or Read Mode:** This mode is used when we want to read data from the file or append the data into the same file.
12	**ab+**	Open a file for appending and read-only mode in the binary format.

www.binils.com

2. CLOSING A FILE The close() Method

To close a file object we will use close() method. The file gets closed and cannot be used again for reading and writing.

The close() method of the file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Syntax:

```
file_object.close()
```

Program for file close method

```
fi=open("F:/Python/sample.txt","wb")
print("Name of the file:",fi.name)
fi.close()
```

Output:

```
Name of the file: sample.txt
```

www.binils.com

ILLUSTRATIVE PROGRAMS

1. WORD COUNT

Program

```
num_words = 0
with open("E:\\AHB\\sample.txt", 'r') as f:
   for line in f:
       words = line.split()
       num_words += len(words)
print("Number of words:")
print(num_words)
```

2. COPY THE CONTENTS OF A FILE

Program 1

```
with open("hello.txt") as f:
  with open("copy.txt", "w") as f1:
        for line in f:
            f1.write(line)
```

www.binils.com

Program 2

```
new_file = open("copy.txt", "w")
with open("hello.txt", "r") as f:
   new_file.write(f.read())
new_file.close()
```

www.binils.com

IV. MODULES

A module allows you to logically organize the python code. Grouping related code into a module makes the code easy to use. A module is a file consisting python code.

- A module can define functions, classes and variables.
- A module can also include runnable code.

Ex

The Python code for a module support normally resides in a file named support.py.

```
def print_func (par):
print ("Hello : ", par)
return
```

We can invoke a module by two statements

- import statement
- from…import statement

i) The import Statement

www.binils.com

- You can use any Python source file as a module by executing an import statement in some other Python source file

Syntax :

```
Import math
```

Example:

```
import math                              # Import module support
print(" the value of pi is ",math.pi)    # Now we can call the
                                         function in that module
as
```

Output:

The value of pi is 3.141592653589793

(ii) The from..import statement

Python from statement lets you import specific attributes from a module into the current namespace.

Syntax

```
from module_name import function_name
```

(iii) The from…import * statement

It is also possible to import all names from a module

Syntax

```
from module_name import *
```

Example

To import the function fib1 and /or fib2 from the module fib

from fib import fib1

Let us import the each functions from the program defined fib.py

```
>>>from fib import fib1
>>>fib1(10)
1 1 2 3 5 8
>>> from fib import fib2
>>>fib2(10)
[1,1,2,3,5,8]
>>>
```

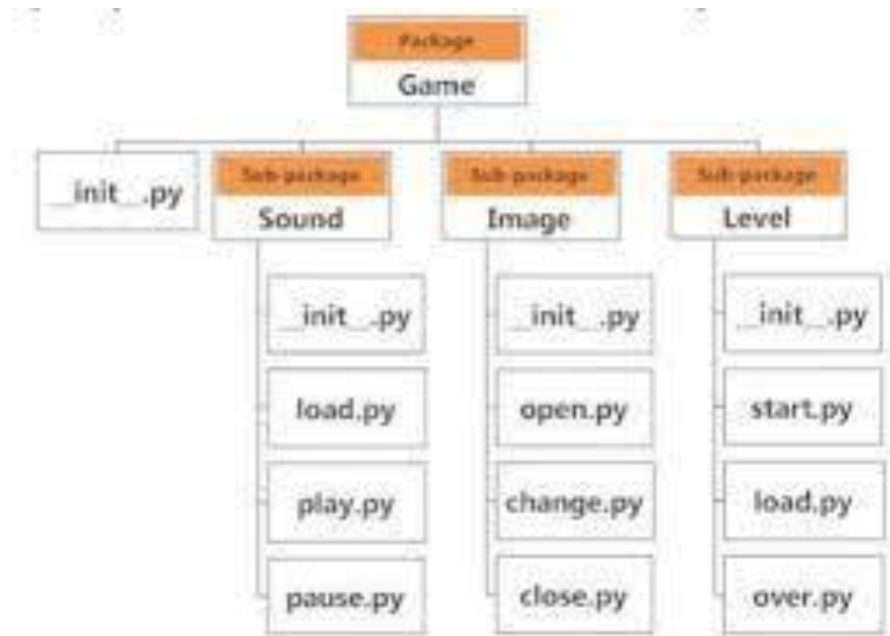
Let us use the from…import * statement

www.binils.com

```
>>>from fib import *
>>>fib1(10)
1 1 2 3 5 8
>>> fib2(10)
[1,1,2,3,5,8]
>>>
```

V. PACKAGES

- A **package** is a collection of modules. A Python package can have sub-packages and modules.
- A directory must contain a file named__ init__ .py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.
- Here is an example. Suppose we are developing a game, one possible organization of packages and modules could be as shown in the figure below.



Importing module from a package

We can import modules from packages using the dot (.) operator.

- For example, if want to import the start module in the above example, it is done as follows. *Import Game.Level.start*
- Now if this module contains a function named *select_difficulty(),* we must use the full name to reference it.

  Game.Level.start.select_difficulty(2)
- If this construct seems lengthy, we can import the module without the package prefix as follows.

  from Game.Level import start
- We can now call the function simply as follows.

  start.select_difficulty(2)
- Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.

  from Game.Level.start import select_difficulty
- Now we can directly call this function.

  select_difficulty(2)
- Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding.

www.binils.com

- While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.

www.binils.com

READING AND WRITING A FILE

There are two methods to access the data in the file.

1. read()
2. write()

1. read() Method

This method reads a string from an open file.

Syntax:

```
file_object.read([count])
```

- Here passed parameter is a number of bytes to be read from the opened file.

For Example if the file have the content,

sample.txt – Problem Solving and Python Programming.

Example Program

```
f=open("F:/Python/sample.txt","r")
str1=f.read(10)
str2=f.read(20)
str3=f.read(30)
print("Read string is :",str1)
print("Read string is :",str2)
print("Read string is :",str3)
f.close()
```

www.binils.com

Output

Read string is : Problem So

Read string is : lving and Python Pro

Read string is : gramming

2. write() Method

This method writes a string from an open file.

Syntax:

```
file_object.write("String")
```

Example:

```
f=open("F:/Python/test.txt","w+"
) print(f.write("This is my first
file")) print(f.read())
f.close()
```

Output:

```
This is my first file
```

WORKING WITH TEXT FILES

1) readline() method

There are two functions to read a file line by line, then there are two methods

- readline()
- readlines()

1.readline()

Every time a user runs the method, it will return a string of characters that contains a single line of information from the file.

Syntax:

```
file_object.readline()
```

Example

Consider the file test.txt contain these three lines This is the First Line in the file
This is the Second Line in the file This is the Third Line in the file

www.binils.com

Now the Python source code is

```
f=open("F:/Python/test.txt","r")
print(f.readline())
f.close()
```

Output:

```
This is the First Line in the file
```

2. readlines() method

This method is used to return a list containing all the lines separated by the special character (
\n).

Syntax:

```
file_object.readlines()
```

Example

Consider the same file

Now the Python source code is

```
f=open("F:/Python/test.txt","r")
print(f.readlines())
f.close()
```

Output

['This is the First Line in the file\n', 'This is the Second Line in the file\n', 'This is the Third
Line in the file']

LOOPING OVER THE FILE

(i) Write a python program to read a file without using read() function.

```
f=open("test.txt","r")
for line in f:
        print(line
) f.close()
```

www.binils.com

Output:

This is the First Line in the file This is the Second Line in the file This is the Third Line in the file

(ii) Write a python program to access a file using with statement.

Syntax:

```
with open("filename") as file:
```

Example:

```
with open("test.txt") as f:
for line in f:
        print(line)
```

Output:
This is the First Line in the file
This is the Second Line in the file This is the Third Line in the file