UNIT V   FILES, MODULES, PACKAGES

Files and exception: text files, reading and writing files, format operator; command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file.

Files :

There are two types of Files they are

- Binary files
- Text files

File operation and file functions

A file is one, which will enable the user to read, write and store a group of related data, without losing them even if the program is over. To perform these functions there are several basic file operations as,

- Naming a file - It is conventional, and convenient to name files in relation to the data stored.
- Opening a file
- Reading data from the file
- Writing data to the file and
- Closing a file

open()

Before going to do any file operation, the concerned file should be opened.

Syntax:

file_object=open("file_name","file_mode")

The **open()** function creates a file object. Finally it should be given, as to what purpose the file is being used. It is also called the file mode.

file_mode	**Description**
R	Opens a file for reading purpose and this is the default file opening mode/
W	Opens a file for writing purpose
A	Opens a file for appending data to it.
r+	Existing file is opened to the beginning for both reading and writing
w+	Opens a file for reading and writing purpose. The file pointer is positioned at the beginning of the file.
a+	Opens a file for reading and writing purpose. The file pointer is positioned at the end of the file
Rb	Opens a file for reading in binary format and this is the default file opening mode.
Ab	Opens a file for appending data in binary format

Wb	Opens a file for writing in binary format
rb+	Opens a file for reading and writing purpose in a binary format and the file pointer is positioned at the beginning of the file
wb+	Opens a file for reading and writing purpose in a binary format. If the file already exists it overwrites the file. If the file does not exist it will create a new one for both reading and writing.
ab+	Opens a file for appending and reading data in binary format. Here the file pointer is positioned at end if the file already exists. It will create a new one if the file does not exists.

For example, a simple open() function call is as follows.

out=open("abc.dat","w")

close()

This function closes a file that was opened by a call **open()**.

The general form of the function call to **close()** is

```
close(file-object)
```

(e.g.)

```
p1=open("abc.dat","w");
p2=open("def.dat","r");
 ______
 ______
close(p1);
close(p2);
```

read() and write()

These two functions are considered as most basic of all input/output functions in file handling. The write**()** function writes characters to a disk file that was previously opened for writing,through the use of the function **open()**.Similarly read**()** function is used to read characters from a file opened in read mode by **open()**.The general format for read() method is:

```
file_object.read(no_of_bytes_to_be_read)
```

The write() method is used to write data to a file. This method requires only one parameter, that must be a string and writes that string into a file. The general for write() method is:

file_object.write(string)

Text file –example:

```
F1=open(“abc.txt”,”x”)
out=open("abc.dat","w")
str= input("Enter string : ")
out.write(str)
out.close()
out=open("abc.dat","r")
str=out.read()
print("File contains")
print(str)
out.close()
```

Output

```
Enter string : Welcome to Python file handling
File contains
Welcome to Python file handling
```

Command line arguments:

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 and can be fetched using "sys" module.

Important steps to be followed:

- Import the module ‘sys’.
- Use sys.argv for getting the list of command line arguments.
- Use len(sys.argv) for getting total number of arguments.

Example program:

```
import sys
noargs=len(sys.argv)
print ("Number of arguments :%d" %noargs)
arguments= str(sys.argv)
print ("Arguments are : %s" %arguments)
```

Output

```
C:\Python27>python cmd1.py one two
Number of arguments :3
        Arguments are : ['cmd1.py', 'one', 'two']
```

Errors and Exceptions:

Errors – referred as bugs in the program**.**

Errors occurs maximum by the fault of the programmer.

Debugging – Process of finding and correcting errors.

Two types of errors.:

•**Syntax errors**
– python interpreter find the syntax error when it executes the coding. Once find the error, it displays the error by stopping the execution.

Common occurring syntax errors are

➢Putting a keyword at wrong place
➢Misspelling the keyword
➢Incorrect indentation
➢Forgetting symbols like comma, brackets, quotes (“ or ‘)
➢Empty block

•**Run time errors**
– if a program is free of syntax errors then it runs by the interpreter and the errors occurs during the run time of the program due to logical mistake is called runtime errors.

Examples:

➢Trying to access a file that doesn’t exists
➢Performing the operations like division by zero
➢Using an identifier which is not defined

These errors are handled using exception handling mechanism

Handling Exceptions:

•**Definition**
– An exception is an event, which occurs during the execution of the program that disrupts the normal flow of the program.

•When the program raises an exception, then python must handle the exception otherwise it terminates and quits

• The handling mechanism is done by
•try, except and else blocks

•try block – suspicious code (code that makes exception)

placed here

•except block – code that handles the exception placed here
and gets executed during exception
•else block – code that is to be executed if no exception is
placed here for normal execution

Structure of blocks of exceptions

try:

write the suspicious code here

except exception1:

If exception1 occurs then this block will be executed

except exception2:

If exception2 occurs then this block will be executed

.
..
else:

If there is no exception then this code will be executed

Example program:

```
try:
     n=int(input(“enter  a value”))
expert:
      print(“you didn’t enter the integer input”)
else:
      print(“value entered correctly and stored”)
```

output:

enter a value:5

value entered correctly and stored

Packages:

Packages are namespaces which contain multiple packages and modules themselves. They are simply directories, but with a twist.

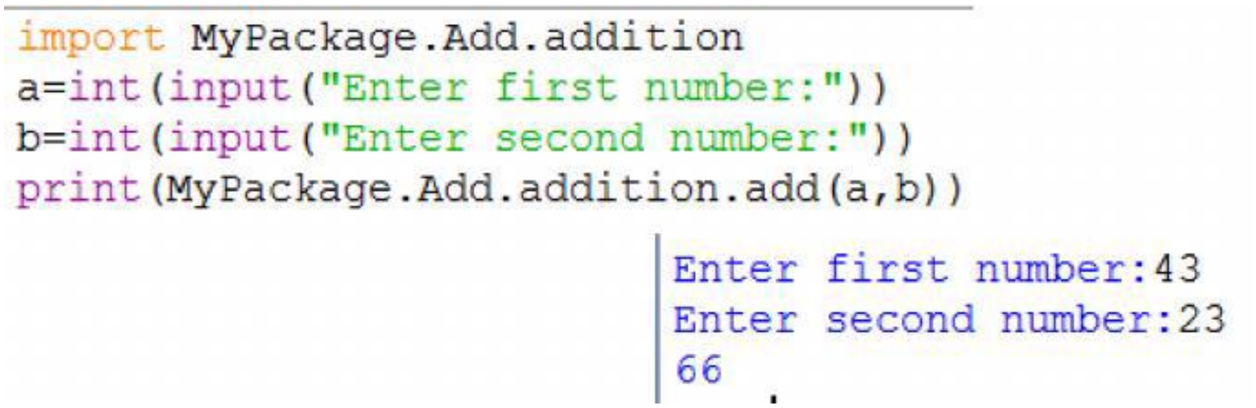
Each package in Python is a directory which must contain a special file called _ _init_ _.py
To be a package the folder must contain a file called __init__.py
Packages can be nested to any depth i.e. it contains many sub packages and modules in it.

11/20/2017
8

Accessing Packages:

Step 1: Create a folder name “MyPackage” in the folder where the python files are storing.
Step 2: Create a subfolder name “Add” in the folder “MyPackage”.
Step 3: Type a python program containing the function to add two numbers with function name “add” and save the file inside the folder
“Add” by the name “addition”

Example Program:



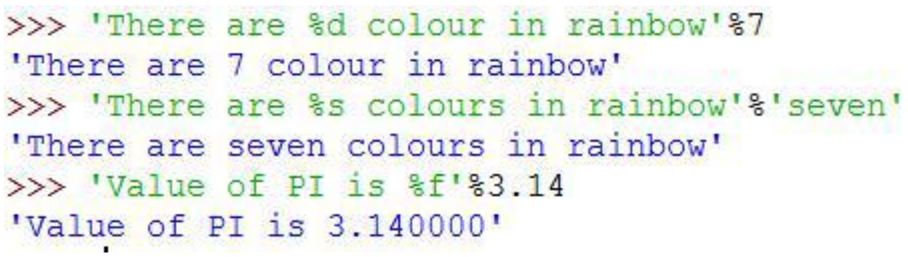
```
import MyPackage.Add.addition
a=int(input("Enter first number:"))
b=int(input("Enter second number:"))
print(MyPackage.Add.addition.add(a,b))
```

```
Enter first number:43
Enter second number:23
66
```

Format Operator – '%'

Symbol is '%'.
Python uses C-style string formatting to create new strings. The "%" operator is used to **format** a set of variables enclosed in a "tuple" along with a **format** string. The format string contains text with argument specifier symbols like "%f" and "%d".

To display the integer value – then use %d
To display a character value – then use %c
To display a string value – then use %s
To display a float value – then use %f



```
>>> 'There are %d colour in rainbow'%7
'There are 7 colour in rainbow'
>>> 'There are %s colours in rainbow'%'seven'
'There are seven colours in rainbow'
>>> 'Value of PI is %f'%3.14
'Value of PI is 3.140000'
```

The use os module in Python

The tasks performed by this module are

- To find the name of the current working directory
- To change the current directory
- To create an new directory
- To delete a file
- To delete a directory
- To check the file present in the current directory or not

Modules

In Python module is a file that contains definitions of functions, variables and classes. The module name is the same as the file name. We have used some scientific functions that present in **math** module and it is a built in Python module. The main advantage of using module is it allows us to make our programs more robust and powerful. We can have our own module and in our example program we created a module by name **"myfunctions.py"**. The module contains two functions definition namely **fact()** and **maximum()**. The coding is as follows.

```
def fact(no):
   f=1
   for i in range(1,no+1):
       f=f*i
   return (f)
def maximum(arr):
   max=arr[0]
   for i in range(1,len(arr)):
      if (max <arr[i]):
         max=arr[i]
   return max
```

```
# Module test example - moduletest.py
import myfunctions
no=int(input("Enter number to find factorial :"))
print ("Factorial of a given number is %d"%myfunctions.fact(5))
a=[8,10,30,15,20]
print("The maximum number is %d"%myfunctions.maximum(a))
```

Output

```
Enter number to find factorial :6
Factorial of a given number is 120
The maximum number is 30
```

Illustrative programs:

Word Count of a file:

```
import sys
fname=sys.argv[1]
n=0
with open(fname,'r') as f:
    for line in f:
        words=line.split()
        n+=len(words)
print("Number of words:",n)
```

Copy file:

```
f1=open(“sourcefile.txt”,”r”)
f2=open(“destinationfile.txt”,”w”)
for line in f1:
            f2.write(“\n”+line)
f1.close( )
f2.close( )
print(“Content of Source file:”)
f1=open(“sourcefile.txt”,”r”)
print(f1.read( ))
print(“Content of Copied file:”)
f2=open(“destinationfile.txt”,”r”)
print(f2.read( ))
```