Scripting Language

UNIT-IV

SYLLABUS

IIIT BASARA

UNIT – IV [12 Lectures]

- Basic Input /Output Operations
- Handling Files & Operations
- Modules
- Error handling – Errors and Exceptions, Exception Handling
- Multi processing and Multi threading
- Formatting - Formatting, Formatting positional, Formatting width precision type, formatting practice, formatting fill align, formatting sign
- Example Programs

python
Programming Language

Basic Input /Output Operations

- Python can be used to read and write data. Also it supports reading and writing data to Files.

“print" statement:

- "print" statement is used to print or display the output on the screen.
- print statement is used to take string as input and place that string to standard output.
- Whatever you want to display on output place that expression inside the inverted commas. The expression whose value is to printed place it without inverted commas.
- Syntax: print(“expression”)
- Ex: >>> print("Welcome")

Input from Keyboard

- Python offers two in-built functions for taking input from user. They are:
- input( )
  - **input( ) function** is used to take input from the user. Whatever expression is given by the user, it is evaluated and result is returned back.
  - Syntax: input("Expression")
  - Ex:
  - >>> n=input("Enter your expression ")
  - Enter your expression 10
  - >>> print ("The evaluated expression is ", n)
  - The evaluated expression is 10

- raw_input( )
- **raw_input( ) function** is used to take input from the user. It takes the input from the Standard input in the form of a string and reads the data from a line at once.
  - Syntax: raw_input("Statement")
  - Ex: n=raw_input("Enter your name");

    print("Welcome",n)
- Note: raw_input() function returns a string. Hence in case an expression is to be evaluated, then it has to be type casted to its following data type.

```
Ex: prn=int(raw_input("Enter Principal"))
     r=int(raw_input("Enter Rate"))
     t=int(raw_input("Enter Time"))
     si=(prn*r*t)/100
     print ("Simple Interest is ",si)
```

#Program to enter details of an user and print them.

```
name=raw_input("Enter your name ")
math=float(raw_input("Enter your marks in Math"))
physics=float(raw_input("Enter your marks in Physics"))
chemistry=float(raw_input("Enter your marks in Chemistry"))
rollno=int(raw_input("Enter your Roll no"))
print ("Welcome ",name  )
print ("Your Roll no is ",rollno  )
print ("Marks in Maths is ",math  )
print ("Marks in Physics is ",physics  )
print ("Marks in Chemistry is ",chemistry  )
print ("Average marks is ",(math+physics+chemistry)/3)
```

Handling Files & Operations

- Python provides the facility of working on 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.
- When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed
- A File is an external storage on hard disk from where data can be stored and retrieved.
- A file is a chunk of logically related data or information which can be used by computer programs.

Python provides some basic functions to manipulate files

Operations on Files:

Opening a File:

- Before working with Files you have to open the File. To open a File, Python built in function open() is used. It returns an object of File which is used with other functions.
- Having opened the file now you can perform read, write, etc. operations on the File.
- **Example**
- f = open("python.txt",'w') # open file in current directory
- f = open("/home/rgukt/Desktop/python/hello.txt")

  # specifying full path

- Used to open or create a file
- Syntax:

  fileObject=open(file_name , access_mode , buffering)

here,

file_name: It is the name of the file which you want to access.

access_mode: It specifies the mode in which File is to be opened.
There are many types of mode like read, write, append.
Mode depends the operation to be performed on File.
Default access mode is read.

buffering: If the buffering value is set to 0, no buffering takes place.
If the buffering value is 1, line buffering is performed while accessing a file

Closing a File:

- Once you are finished with the operations on File at the end you need to close the file.
- It is done by the close( ) method. close( ) method is used to close a File.
  - Syntax: fileobject.close( )

Writing to a File:

- write() method is used to write a string into a file.
- Strings can have binary data or text data
  - Syntax: fileobject.write(string)

Reading from a File:

- read() method is used to read data from the file.
- Parameter ‘count’ is the number of bytes to be read from the opened file
  - Syntax: fileobject.read(count)

Append Mode: Syntax: fileobject.open(“filename”,”a”)

Program to read and write data from a file.

```
obj=open("abcd.txt","w")
obj.write("Welcome to the world of Python\nYeah its great\n")
obj.close()

obj1=open("abcd.txt","r")
s=obj1.read( )
print (s )
obj1.close()

obj2=open("abcd.txt","r")
s1=obj2.read(20)
print (s1 )
obj2.close()
```

Output:

Ex: 2

```
f=open("test.txt","w")
f.write("my first file\n")
f.write("This file\n\n")
f.write("contains three lines\n")
f.write('''hello
this is
first program
in files''')#we can write multiple lines
f.close()
```

Output:

append mode

- Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.

- **Example**

```
f = open("try.txt","a")
f.writelines("hello\nhi\nhow\n")
f.writelines("hello\nhi\nhow\n")
f.close()
```

read mode

- To read the content of a file, we must open the file in reading mode. There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

- **Example**

```
f = open("test.txt",'r')
print (f.read(4))
print (f.read())
```

Output:

readline( ) and readlines( ) methods

- we can use readline() method to read individual lines of a file. This method reads a file till the newline.Reads in at most n bytes/characters if specified.
- the readlines() method returns a list of remaining lines of the entire file. It , including the newline character.Reads in at most n bytes/characters if specified.
- **Example**

```
f = open("try.txt","r")
print(f.readline())
print (f.readline())
print(f.readlines())
f.close()
```

Output:

Close the file

- After opening a file one should always close the opened file. We use method close() for this.
- Always make sure you explicitly close each open file, once its job is done and you have no reason to keep it open. Because - There is an upper limit to the number of files a program can open. If you exceed that limit, there is no reliable way of recovery, so the program could crash.
- Each open file consumes some main-memory for the data-structures associated with it, like file descriptor/handle or file locks etc. So you could essentially end-up wasting lots of memory if you have more files open that are not useful or usable.
- Open files always stand a chance of corruption and data loss.

- **Example**

```
f = open("test.txt")
# perform file operations
f.close()
```

Output:

- **Note:**This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file. A safer way is to use a **try...finally** block.

seek and tell method

- We can change our current file cursor (position) using the **seek()** method.
- Similarly, the **tell()** method returns our current position (in number of bytes).
- **Example**

```
#filename=input("Enter your file name")
f = open("try.txt","r+w")
f.seek(6)
print (f.tell())
f.write("that")
print (f.read())
f.close()
```

- **Replace a word with another word**

```
# Read in the file
with open('try.txt', 'r') as file :
filedata = file.read()
# Replace the target string
filedata = filedata.replace('that', 'somes')
# Write the file out again
with open('try.txt', 'w') as file:
file.write(filedata)
```

Output:

Attributes of File

- There are following File attributes.
- Once a file is opened following list of attributes can get information related a file

Attribute	**Returns [Description]**
Name	Returns the name of the file
Mode	Returns the mode in which file is being opened
Closed	Returns Boolean Value. True, in case if file is closed else false

- #File Program on Attributes

```
result = open("data.txt", "w")
print ("Name of the file:",result.name  )
print ("Opening mode:",result.mode  )
print  ("Closed or not:",result.closed)
result.close()
print  ("Closed or not:",result.closed)
```

- Output:

Modes of File

- There are different modes of file in which it can be opened. They are mentioned in the following table
  - A File can be opened in two modes: 1. Text Mode 2.Binary Mode
- The default is reading in text mode. In this mode, we get strings when reading from the file.
- On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

Mode	Description
r	It opens in Reading mode. It is default mode of File. Pointer is at beginning of the file.
w	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists
a	Opens file in Appending mode. If file already exists, then append the data at the end of existing file, else create a new file.
t	Open in text mode (Default)

b	Open in Binary mode
+	Open a file for updating (reading and writing)
rb	It opens in Reading mode for binary format. It is the default mode. Pointer is at beginning of file.
r+	Opens file for reading and writing. Pointer is at beginning of file.
rb+	Opens file for reading and writing in binary format. Pointer is at beginning of file.
wb	Opens file in Writing mode in binary format. If file already exists, then overwrite the file else create a new file.
w+	Opens file for reading and writing. If file already exists, then overwrite the file else create a new file.
wb+	Opens file for reading and writing in binary format. If file already exists, then overwrite the file else create a new file.

ab	Opens file in Appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.
a+	Opens file in reading and appending mode. If file already exists, then append the data at the end of existing file, else create a new file.
ab+	Opens file in reading and appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.

Note: Moreover, the default encoding is platform dependent.

In windows, it is 'cp1252' but 'utf-8' in Linux.

•So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

•Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

f = open("test.txt",mode = 'r',encoding = 'utf-8')

Python File Methods

- There are various methods available with the file object

Method	**Description**
close()	Close an open file. It has no effect if the file is already closed.
detach()	Separate the underlying binary buffer from the TextIOBase and return it.
fileno()	Return an integer number (file descriptor) of the file.
flush()	Flush the write buffer of the file stream.
isatty()	Return True if the file stream is interactive.
read(n)	Read atmost n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.

Method	**Description**
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resize the file stream to size bytes. If sizeis not specified, resize to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Write string s to the file and return the number of characters written.
writelines(lines)	Write a list of lines to the file.

IIIT BASARA

Infosys | Campus Connect

For Details Contact Me @ :

9247448766

ravikanth27787@gmail.com

python

Programming Language