Bootcamp Training

Python

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

Agenda: Day -5: I/O Handling

- Python File I/O
  - Sending Output to STDOUT Using the print() Method
  - Reading Input with the input() Method
  - Creating File Objects with the open() Method
  - Controlling File Access Modes
  - Working with File Object Attributes
  - Seek() and tell() methods
  - Python OS Module
  - Closing File Objects with the close() Method
  - Reading and Writing to File Objects with read() and write()

Python File I/O

- 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.
- Hence, in Python, a file operation takes place in the following order.

  - **Open a file**
  - **Read or write (perform operation)**
  - **Close the file**

Opening a file

- Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly..

```
>>> f = open("test.txt")    # open file in current directory
>>> f = open("C:/Python33/README.txt")  # specifying full path
```

- We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or 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.

Python File Modes

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

Example

```
f = open("test.txt")      # equivalent to 'r' or 'rt'
f = open("test.txt",'w')  # write in text mode
f = open("img.bmp",'r+b') # read and write in binary mode
```

- Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).
- 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')

Closing a File

- When we are done with operations to the file, we need to properly close it.
- Closing a file will free up the resources that were tied with the file and is done using the close() method.
- Python has a gar ced objects but, we must not rely on

```
try:
    f = open("test.txt",encoding = 'utf-8')
    # perform file operations
finally:
   f.close()
```

Writing to a File

- In order to write into a file we need to open it in write 'w', append 'a' or exclusive creation 'x' mode.
- We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.
- Writing a string o es) is done using write() method. T of characters written to the file.

```
with open("test.txt",'w',encoding = 'utf-8') as f:
    f.write("my first file\n")
    f.write("This file\n\n")
    f.write("contains three lines\n")
```

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten. We must include the newline characters ourselves to distinguish different lines.

Reading From a File

- 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.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4)    # read the first 4 data
'This'
>>> f.read(4)    # read the next 4 data
' is '
>>> f.read()     # read in the rest till end of file
'my first file\nThis file\ncontains three lines\n'
>>> f.read()  # further reading returns empty sting
''
```

We can see, that read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading.

Seek() and tell() methods

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

```
>>> f.tell()    # get the current file position
56
>>> f.seek(0)   # bring file cursor to initial position
0
>>> print(f.read())  # read the entire file
This is my first file
This file
contains three lines
```

Reading line by line

```
for line in f:
...     print(line, end = '')
```

Alternately, we can use readline() method to read individual lines of a file. This method reads a file till the newline, including the newline character.

Python File Methods

Method	Description
readline(n=-1)	Read and return one line from the file. Reads in at most nbytes 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 size is 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.

Python OS Module

- The OS module in Python provides a way of using operating system dependent functionality.
- The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.
- You can find important information about your location or about the process.

Python OS Module

- import os
- Executing a shell command: os.system()
- Get the users environment : os.environ()
- #Returns the current working directory: os.getcwd()
- Return the real group id of the current process: os.getgid()
- Return the current process's user id: os.getuid()
- Returns the real process ID of the current process: os.getpid()
- Set the current numeric umask and return the previous umask: os.umask(mask)
- Return information identifying the current operating system: os.uname()

Python OS Module

- Change the root directory of the current process to path: os.chroot(path)
- Return a list of the entries in the directory given by path: os.listdir(path)
- Create a directory named path with numeric mode mode: os.mkdir(path)
- Recursive directory creation function: os.makedirs(path)
- Remove (delete) the file path: os.remove(path)
- Remove directories recursively: os.removedirs(path)
- Rename the file or directory src to dst: os.rename(src, dst)
- Remove (delete) the directory path: os.rmdir(path)

Any Queries