Data Science with Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

Working with Files

Learning outcomes:

- Opening and Closing Files
- The open Function
- The file Object Attributes
- The close() Method
- Reading and Writing Files
- MORE OPERATIONS ON FILES

Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do your most of the file manipulation using a **file** object.

The *open* Function

Before you can read or write a file, you have to open it using Python's built-in ***open()*** function. This function creates a **file** object, which would be utilized to call other support methods associated with it. We use **open()** function in Python to open a file in read or write mode.

Syntax :

file object = open(file_name [, access_mode][, buffering])

The *open* Function

Here are parameter details:

file_name: The **file_name** argument is a string value that contains the name of the file that you want to access.

access_mode: The **access_mode** determines the mode in which the file has to be opened, i.e., read, write, append, etc.

" **r** ", for reading.

" **w** ", for writing.

" **a** ", for appending.

" **r+** ", for both reading and writing

"**ab**" for appending in binary format

The *open* Function

"**rb**" for reading only in binary format
"**rb+**" for both reading and writing in binary format
"**wb** " for writing only in binary format
"**w+**" for both reading and writing
"**wb+**"for both reading and writing in binary format

One must keep in mind that the mode argument is not mandatory. If not passed, then Python will assume it to be " **r** " by default.

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. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default (default behavior).

The file Object Attributes

Once a file is opened and you have one *file* object, you can get various information related to that file. Here is a list of all attributes related to file object:

file.closed : Returns true if file is closed, false otherwise.

file.mode : Returns access mode with which file was opened.

file.name : Returns name of the file.

file.softspace : Returns false if space explicitly required with print, true otherwise.

The close() Method

The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done. Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax :

fileObject.close();

Reading and Writing Files

The *file* object provides a set of access methods to make our lives easier. We would see how to use *read()* and *write()* methods to read and write files.

write() methods:

The *write()* method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The *write()* method does not add a newline character ('\n') to the end of the string:

Syntax

fileObject.write(string);

Reading and Writing Files

read() method:

The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data, apart from text data.

Syntax

fileObject.read([count]);

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if *count* is missing, then it tries to read as much as possible, maybe until the end of file.

Reading and Writing Files

Till now we have seen the theoretical concept of file I/O.

Now its time for us to write the program for files in python.

File Positions

The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The *seek(offset[, from])* method changes the current file position. The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.

File Positions

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Let's see the example.

Renaming and Deleting Files

rename() Method :

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

Syntax

os.rename(current_file_name, new_file_name)

Example:

```
import os
# Rename a file from Good.txt to Good1.txt
os.rename( " Good.txt ", " Good1.txt " )
```

Renaming and Deleting Files

remove() Method :

You can use the ***remove()*** method to **delete** files by supplying the name of the file to be deleted as the argument.

Syntax :

os.remove(file_name)

Example:

```
import os
# Delete file Good.txt
os.remove("Good.txt")
```