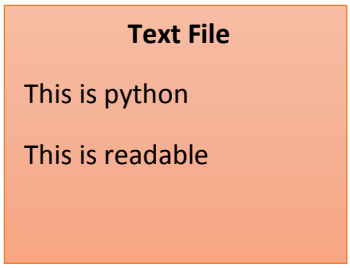
Python File Handling

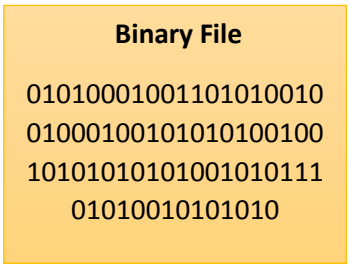
Python supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short.

Python treats file differently as **text** or **binary** and this is important. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun.

Types of file: Text file and Binary File

- A **text file** stores data in the form of alphabets, digits and other special symbols by storing their ASCII values and are in a human readable format. For example, any file with a .txt, .c, etc extension. In the text file, some internal translations takes place when this EOL characters is read or written. In Python, this EOL character is new line ('\n') character by default.
- A **binary** file contains a sequence or a collection of bytes which are not in a human readable format. For example, files with .exe, .mp3, etc extension. It represents custom data. A binary file contains information in the same format in which the information is held in the memory ie in raw format. There is no delimiter in the binary file and also no translation occurs. As a result, binary files are faster and easier for a program to read and write the are text files.





In this section, we will learn all about file handling in python including, creating a file, opening a file, closing a file, writing and appending the file, etc.

Creating a New File/Opening a File

Python provides the open() function which accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like reading, writing, etc.

1. Syntax: file object = open(<file-name>, <access-mode>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

Access mode	**Description**
r	It opens the file to read-only. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
rb	It opens the file to read only in binary format. The file pointer exists at the beginning of the file.
r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.
wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.
a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.
ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.
a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.

ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Let's look at the simple example to open a file named "file.txt" (stored in the same directory) in read mode and printing its content on the console.

Example

```
#opens the file file.txt in read mode
fileptr = open("file.txt","r")

if fileptr:
    print("file is opened successfully")
```

Output:

```
<class '_io.TextIOWrapper'>
file is opened successfully
```

The close() method

Once all the operations are done on the file, we must close it through our python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.

We can perform any operation on the file externally in the file system is the file is opened in python, hence it is good practice to close the file once all the operations are done.

The syntax to use the close() method is given below.

1. **fileobject.close()**

Consider the following example.

Example

```
# opens the file file.txt in read mode
fileptr = open("file.txt","r")

if fileptr:
    print("file is opened successfully")

#closes the opened file
fileptr.close()
```

Reading the file

To read a file using the python script, the python provides us the read() method. The read() method reads a string from the file. It can read the data in the text as well as binary format.

The syntax of the read() method is given below.

1. **fileobj.read(<count>)**

Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

Consider the following example.

Example

```
#open the file.txt in read mode. causes error if no such file exists.
fileptr = open("file.txt","r");

#stores all the data of the file into the variable content
content = fileptr.read(9);

# prints the type of the data stored in the file
print(type(content))

#prints the content of the file
print(content)

#closes the opened file
fileptr.close()
```

Output:

```
<class 'str'>
Hi, I am
```

Read Lines of the file

Python facilitates us to read the file line by line by using a function readline(). The readline() method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

Consider the following example which contains a function readline() that reads the first line of our file **"file.txt"** containing three lines.

Example

```
#open the file.txt in read mode. causes error if no such file exists.
fileptr = open("file.txt","r");

#stores all the data of the file into the variable content
content = fileptr.readline();

# prints the type of the data stored in the file
print(type(content))

#prints the content of the file
print(content)

#closes the opened file
fileptr.close()
```

Output:

```
<class 'str'>
Hi, I am the file and being used as
```

Looping through the file

By looping through the lines of the file, we can read the whole file.

Example

```
#open the file.txt in read mode. causes an error if no such file exists.


fileptr = open("file.txt","r");

#running a for loop
for i in fileptr:
    print(i) # i contains each line of the file
```

Output:

```
Hi, I am the file and being used as
an example to read a
file in python.
```

Read complete file at a time

Python facilitates us to read complete data of the file at one go by using a function readlines(). The readlines() method reads complete data of the file from beginning till the end of character and return contents of the file in a list.

Consider the following example which contains a function readlines() that reads all data of our file **"file.txt"** containing three lines.

Example

```
#open the file.txt in read mode. causes error if no such file exists.
fileptr = open("file.txt","r");

#stores all the data of the file into the variable content
content = fileptr.readlines();

# prints the type of the data stored in the file
print(type(content))

#prints the content of the file
print(content)

#closes the opened file
fileptr.close()
```

Output:

<class 'list>

['Hi, I am the file and being used.\n','Looping through the file\n','By looping through the lines of the file\n', 'we can read the whole file.\n']

Writing to the file

To write some text to a file, we need to open the file using the open method with one of the following access modes.

a: It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.

w: It will overwrite the file if any file exists. The file pointer is at the beginning of the file.

Consider the following example.

Example 1

```
#open the file.txt in append mode. Creates a new file if no such file exists.
fileptr = open("file.txt","a");

#appending the content to the file
fileptr.write("Python is the modern day language. It makes things so simple.")


#closing the opened file
fileptr.close();
```

Now, we can see that the content of the file is modified.

File.txt:

```
Hi, I am the file and being used as
an example to read a
file in python.
Python is the modern day language. It makes things so simple.
```

Example 2

```
#open the file.txt in write mode.
fileptr = open("file.txt","w");

#overwriting the content of the file
fileptr.write("Python is the modern day language. It makes things so simple.")


#closing the opened file
fileptr.close();
```

Now, we can check that all the previously written content of the file is overwritten with the new text we have passed.

File.txt:

```
Python is the modern day language. It makes things so simple.
```

Creating a new file

The new file can be created by using one of the following access modes with the function open(). **x:** it creates a new file with the specified name. It causes an error a file exists with the same name.

a: It creates a new file with the specified name if no such file exists. It appends the content to the file if the file already exists with the specified name.

w: It creates a new file with the specified name if no such file exists. It overwrites the existing file.

Consider the following example.

Example

```
#open the file.txt in read mode. causes error if no such file exists.
fileptr = open("file2.txt","x");

print(fileptr)

if fileptr:
    print("File created successfully");
```

Output:

```
File created successfully
```

Using with statement with files

The with statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files. The with statement is used in the scenario where a pair of statements is to be executed with a block of code in between.

The syntax to open a file using with statement is given below.

```
with open(<file name>, <access mode>) as <file-pointer>:
    #statement suite
```

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the with statement in the case of file s because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file. It doesn't let the file to be corrupted.

Consider the following example.

Example

```
1. with open("file.txt",'r') as f:
2.     content = f.read();
3.     print(content)
```

Output:

Python is the modern day language. It makes things so simple.

File Pointer positions

Python provides the tell() method which is used to print the byte number at which the file pointer exists. Consider the following example.

Example

```
1. # open the file file2.txt in read mode
2. fileptr = open("file2.txt","r")
3.
4. #initially the filepointer is at 0
5. print("The filepointer is at byte :",fileptr.tell())
6.
7. #reading the content of the file
8. content = fileptr.read();
9.
10.#after the read operation file pointer modifies. tell() returns the location of the fileptr.
11.
12.print("After reading, the filepointer is at:",fileptr.tell())
```

Output:

The filepointer is at byte : 0

After reading, the filepointer is at 26

Modifying file pointer position/Accessing contents of file in Random order

In the real world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.

For this purpose, the python provides us the **seek()** method which enables us to place pointer/cursor position in the file as per user's choice. In a file, read/write will take place from the same location where the pointer is.

The syntax to use the seek() method is given below.

1. <file-ptr>.seek(position)

The seek() method accepts one parameters:

position: It refers to the new position of the file pointer within the file from beginning.

from: It indicates the reference position from where the bytes are to be moved. If it is set to 0, the beginning of the file is used as the reference position. If it is set to 1, the current position of the file pointer is used as the reference position. If it is set to 2, the end of the file pointer is used as the reference position.

Consider the following example.

Example

```
1. # open the file file2.txt in read mode
2. fileptr = open("file2.txt","r")
3.
4. #initially the filepointer is at 0
5. print("The filepointer is at byte :",fileptr.tell())
6.
7. #changing the file pointer location to 10.
8. fileptr.seek(10);
9.
10.#tell() returns the location of the fileptr.
11.print("After reading, the filepointer is at:",fileptr.tell())
```

Output:

```
The filepointer is at byte : 0
After reading, the filepointer is at 10
```

Python os module

The os module provides us the functions that are involved in file processing operations like renaming, deleting, etc.

Let's look at some of the os module functions.

Renaming the file

The os module provides us the rename() method which is used to rename the specified file to a new name. The syntax to use the rename() method is given below.

1. rename(?current-name?, ?new-name?)

Example

```
1. import os;
2.
3. #rename file2.txt to file3.txt
4. os.rename("file2.txt","file3.txt")
```

Removing the file

The os module provides us the remove() method which is used to remove the specified file. The syntax to use the remove() method is given below.

1. remove(?file-name?)

Example

```
1. import os;
2.
3. #deleting the file named file3.txt
4. os.remove("file3.txt")
```

Creating the new directory

The mkdir() method is used to create the directories in the current working directory. The syntax to create the new directory is given below.

1. mkdir(?directory name?)

Example

```
1. import os;
2.
3. #creating a new directory with the name new
4. os.mkdir("new")
```

Changing the current working directory

The chdir() method is used to change the current working directory to a specified directory.

The syntax to use the chdir() method is given below.

```
chdir("new-directory")
```

Example

```
import os;

#changing the current working directory to new

os.chdir("new")
```

The getcwd() method

This method returns the current working directory.

The syntax to use the getcwd() method is given below.

```
os.getcwd()
```

Example

```
import os;

#printing the current working directory
print(os.getcwd())
```

Deleting directory

The rmdir() method is used to delete the specified directory.

The syntax to use the rmdir() method is given below.

```
os.rmdir(?directory name?)
```

Example

```
import os;

#removing the new directory
os.rmdir("new")
```

Writing python output to the files

In python, there are the requirements to write the output of a python script to a file.

The **check_call()** method of module **subprocess** is used to execute a python script and write the output of that script to a file.

The following example contains two python scripts. The script file1.py executes the script file.py and writes its output to the text file **output.txt**

file.py:

```
temperatures=[10,-20,-289,100]
def c_to_f(c):
    if c< -273.15:
        return "That temperature doesn't make sense!"
    else:
        f=c*9/5+32
        return f
for t in temperatures:
    print(c_to_f(t))
```

file.py:

```
import subprocess

with open("output.txt", "wb") as f:
    subprocess.check_call(["python", "file.py"], stdout=f)
```

Output:50 -4

That temperature doesn't make sense!

212

The file related methods

The file object provides the following methods to manipulate the files on various operating systems.

SN	Method	Description
1	file.close()	It closes the opened file. The file once closed, it can't be read or write any more.
2	File.flush()	It flushes the internal buffer.
3	File.fileno()	It returns the file descriptor used by the underlying implementation to request I/O from the OS.
4	File.next()	It returns the next line from the file.
5	File.read([size])	It reads the file for the specified size.

6	File.readline([size])	It reads one line from the file and places the file pointer to the beginning of the new line.
7	File.readlines([sizehint])	It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function.
8	File.seek(offset)	It modifies the position of the file pointer to a specified offset with the specified reference.
9	File.tell()	It returns the current position of the file pointer within the file.
10	File.truncate([size])	It truncates the file to the optional specified size.
11	File.write(str)	It writes the specified string to a file
12	File.writelines(seq)	It writes a sequence of the strings to a file.

Reading from one file and Writing on another file simultaneously

An example to read a file "Story.txt" and transfer all the contents to another file "File1.txt"

```
try:
   filename=open("Story.txt","r") #opening file for reading
   filecontents=filename.readlines() #reading contents from file
   fwrite=open("File1.txt","w") #opening another file for writing
   fwrite.writelines(filecontents)
   print("Contents has been written on another file")
except:
   print("Error in file accessing")
```

An example to read a file "Story.txt" and display all the contents on the screen after removing all the space from file

```
try:
  fileobj=open(r"D:\Files\story.txt","r")
  filetext=fileobj.readlines()

  for lines in filetext:
    for ch in lines: #space checking
      if ch==' ':
        continue
```