

Lecture 15 - 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.

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

1. Open a file
2. Read
3. Write to a file from a file
4. Close the file
5. File methods

Open or Create a file:

To open a file we use the **open()** function, this function will create a new file if there isn't any or else it will use the old file.

Opening file Syntax :

f is the file object known as handle

```
f = open("file name", file_mode)
```

Example :

```
f = open("file.txt", w)
```

*Here we have passed the 'w' mode which means we are opening this file to **write** something*

Usually , **w**, **a** and **r** mode are used to write, append and read respectively.

Python File Modes:

- **r** for **reading** – The file pointer is placed at the beginning of the file. This is the default mode.
- **r+** Opens a file for **both** reading and writing. The file pointer will be at the beginning of the file.
- **w** Opens a file for **writing** only. **Overwrites** the file if the file exists. **If the file does not exist**, creates a new file for writing.
- **w+** Opens a file for **both** writing and reading. Overwrites the existing file if the file exists. **If the file does not exist**, it creates a new file for reading and writing.
- **rb** Opens a file for **reading** only in binary format. The file pointer is placed at the beginning of the file.
- **rb+** Opens a file for **both** reading and writing in binary format.
- **wb+** Opens a file for **both** writing and reading in binary format. Overwrites the existing file if the file exists. **If the file does not exist**, it creates a new file for reading and writing.
- **a** Opens a file for **appending**. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. **If the file does not exist**, it creates a new file for writing.
- **ab** Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. **If the file does not exist**, it creates a new file for writing.
- **a+** Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. **If the file does not exist**, it creates a new file for reading and writing.
- **ab+** Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. **If the file does not exist**, it creates a new file for reading and writing.
- **x** open for exclusive creation, failing if the file already exists (Python 3)

Examples :

```
f = open("file.txt") # by default read mode  
f = open("file.txt", 'w') # open file for write mode  
f = open("file.txt", 'a') #open file for append  
f = open("file.txt", 'r+b') #open file in read and binary  
mode
```

Once a file is opened and done with reading or writing methods, it must be close. To close a file, use the **close()** method

Example :

```
f = open("file.txt", 'w')  
f.close() # close a file
```

Write in a file

To write in a file , the file can be opened in 3 modes 'w' write mode or 'a' append mode or 'x' exclusive creation mode.

When we write in a file with 'w' mode it **overwrites the previous data** if the file is already existing. So, we normally open file with 'w' mode when we writing in a new file or we want to overwrite the old one.

When we write in a file with 'a' **append** mode it **does not** overwrite the previous data instead it continues from it.

Once the file is open and a mode was assigned one can write in the file with the help of **write()** method.

This **write()** method return a sequence of character and write it in the txt file.

Syntax to write in a file :

```
with open ("file name with txt extension", 'w') as file_object:
```

```
    file_object.write("statement which you want to write in  
                        the txt file")
```

Example

```
with open ("file.txt", 'w') as f:
```

```
    f.write("hello this is written in file.txt ")  
    f.write("it's in 2nd line")  
    print("this statement would not be written in the  
file ")
```

Output on the screen

this statement would not be written in the file

The content of file.txt

hello this is written in file.txt

it's in 2nd line

The above code is equivalent to:

```
f=open("file.txt", 'w')
f.write("hello this is written in file.txt")
f.write("\nit's in 2nd line")
print("this statement would not be written in the file ")
f.close()
```

*Example using **append()** mode :*

```
file = open('file.txt','a')  
file.write("This will add this line")  
file.close()
```

Sample Run

The content of file.txt

hello this is written in file.txt
it's in 2nd line
This will add this line

*# Write text data to a file - **Create the file if it does not exist***

```
with open('filename.txt' , 'wt') as f:  
    f.write ('hi there, this is a first line of file.\n')  
    f.write ('and another line.\n')
```


Content of filename.txt after running this program

*hi there, this is a first line of file.
and another line.*

Read from the file:

To read from a file first we need to open the file in read mode.

Once the file is open with the help of ***read()*** method we can read the text written in the file. In the ***read()*** method we can also pass some integer value as an argument and the read method will only read that many characters.

Syntax to read form a file

```
with open ("file name with txt extension", r) as  
file_object:  
    file_object.read(size)
```

OR

```
file_object = open("file name", r)  
file_object.read()
```

Example:

Here we will read that file which we have created using the write mode in the above example:

```
f=open("file.txt", 'r')    #Opening the file
print(f.read())           #read the entire file
```

Sample Run

```
hello this is written in file.txt
it's in 2nd line
>>>
```

Example 2

```
f=open ("file.txt", 'r')    #Opening the file
print(f.read(6))           #read first 6 characters
print(f.read(5))           # read next 5 characters
print(f.read())             # read all that's left
```

Sample Run

```
hello
this
is written in file.txtit's in 2nd line
>>>
```

Example : Using a for loop to print each line in a file

a file named "file.txt", will be opened with the reading mode.

```
file = open('file.txt', 'r')
```

```
# This will print every line one by one in the file  
for each in file:  
    print (each)
```

Sample Run

```
hello  
this  
is  
written  
in file.txtit's in 5th line  
>>>
```

Example : Reading one character a time

```
file = open('file.txt', 'r')
```

```
while 1:
```

```
    # read by character
```

```
    char = file.read(1)
```

```
    if not char:
```

```
        break
```

```
    print(char)
```

```
file.close()
```

Sample Run

```
h
```

```
e
```

```
|
```

```
|
```

```
o
```

```
>>>
```

Example using append() mode

```
file = open('file.txt', 'a')  
file.write("This will add this line")  
file.close()
```

This will print every line one by one in the file

```
file = open('file.txt', 'r')  
for each in file:  
    print (each)
```

Sample run

```
hello  
this  
is  
written  
in file.txtit's in 5th line  
This will add this line  
>>>
```

Reading a file word by word

```
# opening the text file  
with open('file.txt','r') as file:
```

```
    # reading each line  
    for line in file:
```

```
        # reading each word  
        for word in line.split():
```

```
            # displaying the words  
            print(word)
```


Sample run

```
hello  
this  
is  
written  
in  
file.txtit's  
in  
5th  
line  
This  
will  
add  
this  
line  
>>>
```

Example : Check if a string exists in a file

```
def check_if_string_infile():  
    # opening the text file  
    with open('file.txt','r') as file:  
        # reading each line  
        for line in file:  
            # reading each word  
            if 'this' in line:  
                return True  
  
    return False  
  
print(check_if_string_infile())
```

Sample Run

True

Example - 2

```
def check_if_string_infile():  
    # opening the text file  
    with open('file.txt','r') as file:  
        # reading each line  
        for line in file:  
            # reading each word  
            if 'Texas' in line:  
                return True  
  
    return False  
  
print(check_if_string_infile())
```

Sample Run

False

Renaming and Deleting Files

Python `os` module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions

The `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

Following is the example to rename an existing file *test1.txt*

```
#!/usr/bin/python
```

```
import os
```

```
# Rename a file from test1.txt to test2.txt
```

```
os.rename( "test1.txt", "test2.txt" )
```

The `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

Following is the example to delete an existing file `test2.txt`

```
# !/usr/bin/python
```

```
import os
```

```
# Delete file test2.txt
```

```
os.remove("text2.txt")
```