



File

# FILES

- Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory

# open Function

- `file object = open(file_name [, access_mode][, buffering])`
- **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. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **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).

# Opening Files in Python

- Python has a built-in `open()` function 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:/Python38/README.txt") # specifying  
full path
```

# Modes & Description

- **r**
- Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
- **rb**
- Opens a file for reading only in binary format. 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 placed at the beginning of the file.

# Modes & Description

- **rb+**
- Opens a file for both reading and writing in binary format. The file pointer placed 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.
- **wb**
- Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

# Modes & Description

- **w+**
- Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
- **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, 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.

# Modes & Description

- **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.



# Modes

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

# Modes

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

# Modes

- 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')
```

# *file* Object Attributes

- ❑ **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.

# *close()* Method

- `fileObject.close()`



# close a file is by using the with statement

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

# *write()* Method

- `fileObject.write(string)`

# Writing to Files in Python

In order to write into a file in Python, 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. Due to this, all the previous data are erased.

```
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 in the current directory if it does not exist. If it does exist, it is overwritten



# *read()* Method

- `fileObject.read([count])`  
passed parameter is the number of bytes to be read from the opened file

# Reading Files in Python

To read a file in Python, we must open the file in reading r mode. There are various methods available for this purpose. We can use the read(size) method to read in the size number of data. If the 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
```

# File Positions

- *tell()* method tells you the current position within the file
- *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.

- *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.

# Seek and tell

- 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
```

# Read line by line

- We can read a file line-by-line using a for loop. This is both efficient and fast.

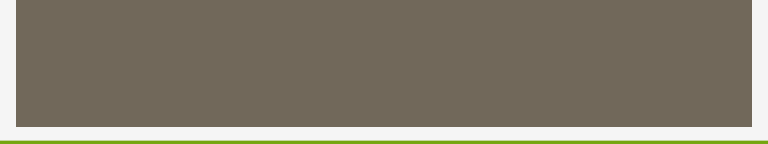
for line in f:

    print(line, end = '')

This is my first file

This file

contains three lines



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

```
f.readline()
'This is my first file\n'
f.readline()
'This file\n'
f.readline()
'contains three lines\n'
f.readline()
''
```

Lastly, the `readlines()` method returns a list of remaining lines of the entire file. All these reading methods return empty values when the end of file (EOF) is reached.

```
f.readlines()
['This is my first file\n', 'This file\n', 'contains three lines\n']
```

# rename() Method

□ `os.rename(current_file_name, new_file_name)`



# The *remove()* Method

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

- ❑ `fileno()` Return an integer number (file descriptor) of the file.
- ❑ `writelines(lines)` Write a list of lines to the file.

# Copy the data from 1 text file to another text file

```
# open both files
```

```
with open('file.txt','r') as firstfile, open('second.txt','a') as  
secondfile:
```

```
# read content from first file
```

```
for line in firstfile:
```

```
    # append content to second file
```

```
    secondfile.write(line)
```

# Display File Contents in Reverse Order

```
□ import os
□
□ filename = input("Enter name of file with extension : ")
□ if filename not in os.listdir():
□     # File with given name does not exist in current directory
□     print("Oops! seems like file does not exist.")
□ else:
□     file = open(filename,"r")
□     str = reversed(file.readlines())
□     for i in str:
□         print(i.rstrip()[::-1])
□     file.close()
```

# Read the file and append in list

```
fl = []  
f = open("file.txt")  
for line in f :  
    line=line.rstrip("\n")  
    f.append(line)  
print(fl)
```