**What is a file?**
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

---

**How to open 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.

File handling is an important part of any web application.
Python has several functions for creating, reading, updating, and deleting files.

# File Handling

The key function for working with files in Python is the `open()` function.
The `open()` function takes two parameters; *filename*, and *mode*.
There are four different methods (modes) for opening a file:
`"r"` - Read - Default value. Opens a file for reading, error if the file does not exist
`"a"` - Append - Opens a file for appending, creates the file if it does not exist
`"w"` - Write - Opens a file for writing, creates the file if it does not exist
`"x"` - Create - Creates the specified file, returns an error if the file exists
In addition you can specify if the file should be handled as binary or text mode
`"t"` - Text - Default value. Text mode
`"b"` - Binary - Binary mode (e.g. images)

| | |
|---|---|
| '+' | Open a file for updating (reading and writing) |

# Syntax

To open a file for reading it is enough to specify the name of the file:
```
f = open("demofile.txt")
```
The code above is the same as:
```
f = open("demofile.txt", "rt")
```
Because `"r"` for read, and `"t"` for text are the default values, you do not need to specify them.
**Note:** Make sure the file exists, or else you will get an error.

# Open a File on the Server

Assume we have the following file, located in the same folder as Python:
demofile.txt
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
To open the file, use the built-in `open()` function.
The `open()` function returns a file object, which has a `read()` method for reading the content of the file:

## Example

```
f = open("demofile.txt", "r")
print(f.read())
```

# Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

## Example

Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

# Read Lines- You can return one line by using the readline() method:

## Read one line of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
```

By calling readline() two times, you can read the two first lines:

### Read two lines of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

### Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

# Close Files-It is a good practice to always close the file when you are done with it.

Close the file when you are finish with it:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

# Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

## Example

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
```

```
#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```
**Note:** the "w" method will overwrite the entire file.

# Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:
"x" - Create - will create a file, returns an error if the file exist
"a" - Append - will create a file if the specified file does not exist
"w" - Write - will create a file if the specified file does not exist

## Example--Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```
Result: a new empty file is created!

## Example---Create a new file if it does not exist:

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

# Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

## Example

Remove the file "demofile.txt":
```
import os
os.remove("demofile.txt")
```

---

# Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

## Example

Check if file exists, *then* delete it:
```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

### How to read files in Python?

To read a file in Python, 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')
>>> 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, the `read()` method returns newline as `'\n'`. Once the end of file is reached, we get empty string on further reading.

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
```

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
```

The lines in file itself has a newline character `'\n'`.

Moreover, the `print()` end parameter to avoid two newlines when printing.

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.

```
>>> 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 method return empty values when end of file (EOF) is reached.

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