Python provides inbuilt Python provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files (written in binary language, 0s and 1s).

* **Text files:** In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.
* **Binary files:** In this type of file, there is no terminator for a line and the data is stored after converting it into machine-understandable binary language.

#### **Access mode**

Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once it’s opened. These modes also define the location of the File Handle in the file. File handle is like a cursor, which defines from where the data has to be read or written in the file. Different access modes for reading a file are –

1. **Read Only (‘r’) :** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.
2. **Read and Write (‘r+’) :** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.
3. **Append and Read (‘a+’) :** Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
4. **Write (‘w’) :**  Write python strings into a text file
5. **Create a new text file (‘x’) :** Creates a new text file. Throws an error if the file already exists.

**Creating a new File**

**Syntax :**

file = open('file.txt','x')

Creates a new file by the name ‘file.txt’. In addition to creating a new file, we can also specify if we want this file to be interpreted as binary or text.

‘t’ = Text

‘B’ = Binary

file = open('file.txt','xt')

By default, the file creation is of ‘text’ type.

If the file already exists, a *FileExistsError* is thrown.

file = open('file.txt','xt')

**---------------------------------------------------------------------------**

**FileExistsError** Traceback (most recent call last)

**<ipython-input-16-47565d383243>** in <module>

**----> 1** file **=** open**('file.txt','x')**

**FileExistsError**: [Errno 17] File exists: 'file.txt'

## **Opening a File**

It is done using the open() function. No module is required to be imported for this function.

**Syntax:**

File\_object = open(r"File\_Name", "Access\_Mode")

The file should exist in the same directory as the python program file else, full address of the file should be written on place of filename.

**Note:** The r is placed before filename to prevent the characters in filename string to be treated as special character. For example, if there is \temp in the file address, then \t is treated as the tab character and error is raised of invalid address. The r makes the string raw, that is, it tells that the string is without any special characters. The r can be ignored if the file is in same directory and address is not being placed.

| # Open function to open the file "MyFile1.txt"  # (same directory) in read mode and  file1 = open("MyFile.txt", "r")    # store its reference in the variable file1  # and "MyFile2.txt" in D:\Text in file2  file2 = open(r"D:\Text\MyFile2.txt", "r+") |
| --- |

Here, file1 is created as object for MyFile1 and file2 as object for MyFile2.

## **Closing a file**

close() function closes the file and frees the memory space acquired by that file. It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.

**Syntax:**

File\_object.close()

| # Opening and Closing a file "MyFile.txt"  # for object name file1.  file1 = open("MyFile.txt", "r")  file1.close() |
| --- |

## **Reading from a file**

There are three ways to read data from a text file.

**read() :** Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the entire file.  
File\_object.read([n])

**readline() :** Reads a line of the file and returns in form of a string.For specified n, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.  
File\_object.readline([n])

**readlines() :** Reads all the lines and return them as each line a string element in a list.  
File\_object.readlines()

**Note:** ‘\n’ is treated as a special character of two bytes.

**Example:**

| # Program to show various ways to  # read data from a file.    # Creating a file  file1 = open("myfile.txt", "w")  L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]    # Writing data to a file  file1.write("Hello \n")  file1.writelines(L)  file1.close() # to change file access modes    file1 = open("myfile.txt", "r+")    print("Output of Read function is ")  print(file1.read())  print()    # seek(n) takes the file handle to the nth  # bite from the beginning.  file1.seek(0)    print("Output of Readline function is ")  print(file1.readline())  print()    file1.seek(0)    # To show difference between read and readline  print("Output of Read(9) function is ")  print(file1.read(9))  print()    file1.seek(0)    print("Output of Readline(9) function is ")  print(file1.readline(9))  print()    file1.seek(0)    # readlines function  print("Output of Readlines function is ")  print(file1.readlines())  print()  file1.close() |
| --- |

**Output:**

Output of Read function is

Hello

This is Delhi

This is Paris

This is London

Output of Readline function is

Hello

Output of Read(9) function is

Hello

Th

Output of Readline(9) function is

Hello

Output of Readlines function is

['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']

#### **With statement**

with statement in Python is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams. Unlike the above implementations, there is no need to call file.close() when using with statement. The with statement itself ensures proper acquisition and release of resources.

**Syntax:**

with open filename as file:

| # Program to show various ways to  # read data from a file.    L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]    # Creating a file  with open("myfile.txt", "w") as file1:  # Writing data to a file  file1.write("Hello \n")  file1.writelines(L)  file1.close() # to change file access modes    with open("myfile.txt", "r+") as file1:  # Reading form a file  print(file1.read()) |
| --- |

**Output:**

Hello

This is Delhi

This is Paris

This is London

ilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files (written in binary language, 0s and 1s).

* **Text files:** In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.
* **Binary files:** In this type of file, there is no terminator for a line and the data is stored after converting it into machine-understandable binary language.

#### **Access mode**

Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once it’s opened. These modes also define the location of the File Handle in the file. File handle is like a cursor, which defines from where the data has to be read or written in the file. Different access modes for reading a file are –

1. **Read Only (‘r’) :** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.
2. **Read and Write (‘r+’) :** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.
3. **Append and Read (‘a+’) :** Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

**Note:** To know more about access mode [click here](https://www.geeksforgeeks.org/reading-writing-text-files-python/).

## **Opening a File**

It is done using the open() function. No module is required to be imported for this function.

**Syntax:**

File\_object = open(r"File\_Name", "Access\_Mode")

The file should exist in the same directory as the python program file else, full address of the file should be written on place of filename.

**Note:** The r is placed before filename to prevent the characters in filename string to be treated as special character. For example, if there is \temp in the file address, then \t is treated as the tab character and error is raised of invalid address. The r makes the string raw, that is, it tells that the string is without any special characters. The r can be ignored if the file is in same directory and address is not being placed.

| # Open function to open the file "MyFile1.txt"  # (same directory) in read mode and  file1 = open("MyFile.txt", "r")    # store its reference in the variable file1  # and "MyFile2.txt" in D:\Text in file2  file2 = open(r"D:\Text\MyFile2.txt", "r+") |
| --- |

Here, file1 is created as object for MyFile1 and file2 as object for MyFile2.

## **Closing a file**

close() function closes the file and frees the memory space acquired by that file. It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.

**Syntax:**

File\_object.close()

| # Opening and Closing a file "MyFile.txt"  # for object name file1.  file1 = open("MyFile.txt", "r")  file1.close() |
| --- |

## **Reading from a file**

There are three ways to read data from a text file.

**read() :** Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the entire file.  
File\_object.read([n])

**readline() :** Reads a line of the file and returns in form of a string.For specified n, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.  
File\_object.readline([n])

**readlines() :** Reads all the lines and return them as each line a string element in a list.  
File\_object.readlines()

**Note:** ‘\n’ is treated as a special character of two bytes.

**Example:**

| # Program to show various ways to  # read data from a file.    # Creating a file  file1 = open("myfile.txt", "w")  L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]    # Writing data to a file  file1.write("Hello \n")  file1.writelines(L)  file1.close() # to change file access modes    file1 = open("myfile.txt", "r+")    print("Output of Read function is ")  print(file1.read())  print()    # seek(n) takes the file handle to the nth  # bite from the beginning.  file1.seek(0)    print("Output of Readline function is ")  print(file1.readline())  print()    file1.seek(0)    # To show difference between read and readline  print("Output of Read(9) function is ")  print(file1.read(9))  print()    file1.seek(0)    print("Output of Readline(9) function is ")  print(file1.readline(9))  print()    file1.seek(0)    # readlines function  print("Output of Readlines function is ")  print(file1.readlines())  print()  file1.close() |
| --- |

**Output:**

Output of Read function is

Hello

This is Delhi

This is Paris

This is London

Output of Readline function is

Hello

Output of Read(9) function is

Hello

Th

Output of Readline(9) function is

Hello

Output of Readlines function is

['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']

#### **With statement**

with statement in Python is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams. Unlike the above implementations, there is no need to call file.close() when using with statement. The with statement itself ensures proper acquisition and release of resources.

**Syntax:**

with open filename as file:

| # Program to show various ways to  # read data from a file.    L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]    # Creating a file  with open("myfile.txt", "w") as file1:  # Writing data to a file  file1.write("Hello \n")  file1.writelines(L)  file1.close() # to change file access modes    with open("myfile.txt", "r+") as file1:  # Reading form a file  print(file1.read()) |
| --- |

**Output:**

Hello

This is Delhi

This is Paris

This is London

**Exploring the write function**

file1 = open('myfile.txt','w')

words = ['Welcome','to','GeeksforGeeks']

for word in words :

file1.write(word + '\n')

**Handling *FileExistsError***

try:

with open('file.txt','x') :

print('File Created')

except FileExistsError:

print('File with the same name already present')

**Output:**

File with the same name already present