File Handling in Python – How to Create, Read, and Write to a File

File handling is an important activity in every web app. The types of activities that you can perform on the opened file are controlled by Access Modes. These describe how the file will be used after it has been opened.

These modes also specify where the file handle should be located within the file. Similar to a pointer, a file handle indicates where data should be read or put into the file.

Python has several functions for creating, reading, updating, and deleting files.

Opening Files in Python

The **open()** Python method is the primary file handling function. The basic syntax is:

```
file_object = open('file_name', 'mode')
```

The **open()** function takes two elementary parameters for file handling:

1. The **file_name** includes the file extension and assumes the file is in the current working directory. If the file location is elsewhere, provide the absolute or relative path.

2. The **mode** is an optional parameter that defines the file opening method. The table below outlines the different possible options:

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)

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.

Open a File on the Server

Assume we have the following file, located in the same folder as Python:

demofile.txt

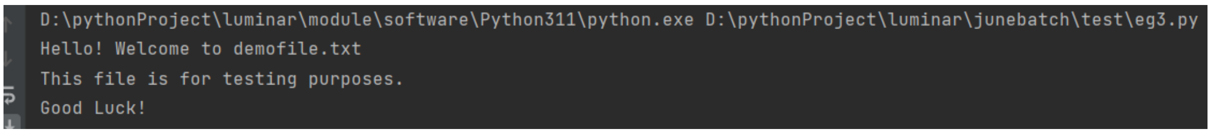
```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

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

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



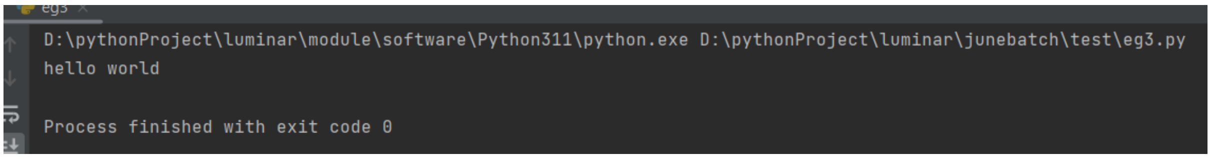
```
D:\pythonProject\luminar\module\software\Python311\python.exe D:\pythonProject\luminar\junebatch\test\eg3.py
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

If the file is located in a different location, you will have to specify the file path, like this:

Example

Open a file on a different location:

```python
f = open("D:/demofile/demo.txt ,"r")
print(f.read())
```



```
D:\pythonProject\luminar\module\software\Python311\python.exe D:\pythonProject\luminar\junebatch\test\eg3.py
hello world

Process finished with exit code 0
```

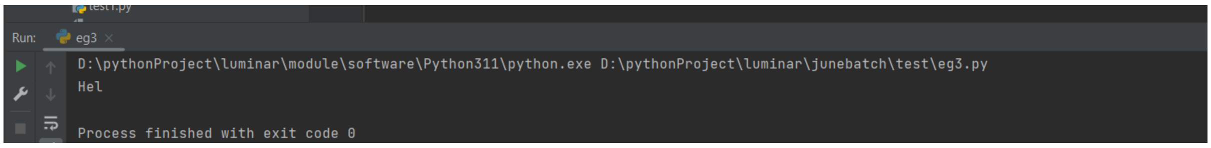
Read Only Parts of the File

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(3))
```



```
D:\pythonProject\luminar\module\software\Python311\python.exe D:\pythonProject\luminar\junebatch\test\eg3.py
Hel

Process finished with exit code 0
```

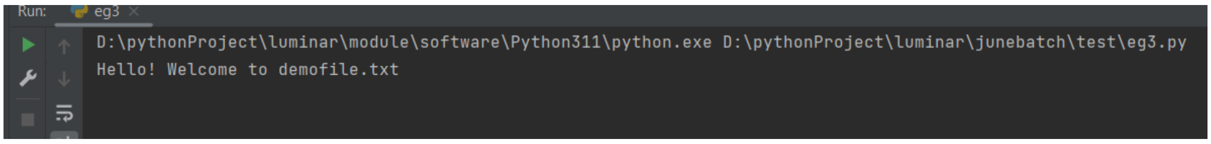
Read Lines

You can return one line by using the readline() method:

Example

Read one line of the file:

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



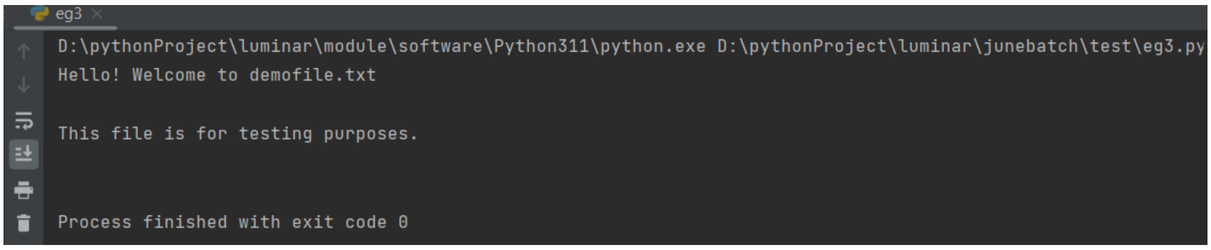
```
D:\pythonProject\luminar\module\software\Python311\python.exe D:\pythonProject\luminar\junebatch\test\eg3.py
Hello! Welcome to demofile.txt
```

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

Example

Read two lines of the file:

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



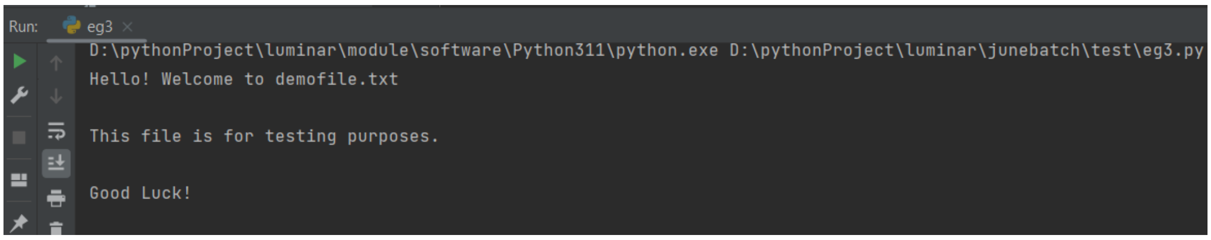
```
D:\pythonProject\luminar\module\software\Python311\python.exe D:\pythonProject\luminar\junebatch\test\eg3.py
Hello! Welcome to demofile.txt

This file is for testing purposes.


Process finished with exit code 0
```

By looping through the lines of the file, you can read the whole file, line by line:

Example

Loop through the file line by line:

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



```
D:\pythonProject\luminar\module\software\Python311\python.exe D:\pythonProject\luminar\junebatch\test\eg3.py
Hello! Welcome to demofile.txt

This file is for testing purposes.

Good Luck!
```

Close Files

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

Example

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.

Python File Write

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())
```

Example

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 overwriting:
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")
```

Python Delete File

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")
```

Delete Folder

To delete an entire folder, use the `os.rmdir()` method:

Example

Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

Note: You can only remove *empty* folders.

Advantages of File Handling

- **Versatility**: File handling in Python allows you to perform a wide range of operations, such as creating, reading, writing, appending, renaming, and deleting files.
- **Flexibility**: File handling in Python is highly flexible, as it allows you to work with different file types (e.g. text files, binary files, CSV files, etc.), and to perform different operations on files (e.g. read, write, append, etc.).
- **User–friendly**: Python provides a user-friendly interface for file handling, making it easy to create, read, and manipulate files.
- **Cross-platform**: Python file-handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.

Disadvantages of File Handling

- **Error-prone:** File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.).
- **Security risks**: File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system.

- **Complexity**: File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely.

- **Performance**: File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.