Python - File Handling

File Handling in Python

File handling in Python involves interacting with files on your computer to read data from them or write data to them. Python provides several built-in functions and methods for creating, opening, reading, writing, and closing files. This tutorial covers the basics of file handling in Python with examples.

Opening a File in Python

To perform any file operation, the first step is to open the file. Python's built-in open() function is used to open files in various modes, such as reading, writing, and appending. The syntax for opening a file in Python is −

```
file = open("filename", "mode")
```

Where, **filename** is the name of the file to open and **mode** is the mode in which the file is opened (e.g., 'r' for reading, 'w' for writing, 'a' for appending).

File Opening Modes

Following are the file opening modes −

Sr.No.	Modes & Description
1	**r** Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	**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.
3	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	**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.
6	**b** Opens the file in binary mode
7	**t** Opens the file in text mode (default)
8	**+** open file for updating (reading and writing)
9	**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.
10	**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.
11	**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.
12	**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.
13	**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.
14	**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.
15	**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.
16	**x** open for exclusive creation, failing if the file already exists

Once a file is opened and you have one file object, you can get various information related to that file.

Example 1

In the following example, we are opening a file in different modes –

```
# Opening a file in read mode
file = open("example.txt", "r")

# Opening a file in write mode
file = open("example.txt", "w")

# Opening a file in append mode
file = open("example.txt", "a")

# Opening a file in binary read mode
file = open("example.txt", "rb")
```

Example 2

In here, we are opening a file named "foo.txt" in binary write mode ("wb"), printing its name, whether it's closed, and its opening mode, and then closing the file –

</> Open Compiler

```
# Open a file
fo = open("foo.txt", "wb")
print ("Name of the file: ", fo.name)
print ("Closed or not: ", fo.closed)
print ("Opening mode: ", fo.mode)
fo.close()
```

It will produce the following **output** –

```
Name of the file: foo.txt
Closed or not: False
Opening mode: wb
```

Learn **Python** in-depth with real-world projects through our **Python certification course**. Enroll and become a certified expert to boost your career.

Reading a File in Python

Reading a file in Python involves opening the file in a mode that allows for reading, and then using various methods to extract the data from the file. Python provides several methods to read data from a file −

- **read() −** Reads the entire file.
- **readline() −** Reads one line at a time.
- **readlines −** Reads all lines into a list.

> *To read a file, you need to open it in read mode. The default mode for the open() function is read mode ('r'), but it's good practice to specify it explicitly.*

Example: Using read() method

In the following example, we are using the read() method to read the whole file into a single string −

```
with open("example.txt", "r") as file:
   content = file.read()
   print(content)
```

Following is the output obtained −

```
Hello!!!
Welcome to TutorialsPoint!!!
```

Example: Using readline() method

In here, we are using the readline() method to read one line at a time, making it memory efficient for reading large files line by line −

```
with open("example.txt", "r") as file:
   line = file.readline()
   while line:
      print(line, end='')
      line = file.readline()
```

Output of the above code is as shown below −

```
Hello!!!
Welcome to TutorialsPoint!!!
```

Example: Using readlines() method

Now, we are using the readlines() method to read the entire file and splits it into a list where each element is a line −

```
with open("example.txt", "r") as file:
   lines = file.readlines()
   for line in lines:
      print(line, end='')
```

We get the output as follows −

```
Hello!!!
Welcome to TutorialsPoint!!!
```

Writing to a File in Python

Writing to a file in Python involves opening the file in a mode that allows writing, and then using various methods to add content to the file.

To write data to a file, use the write() or writelines() methods. When opening a file in write mode ('w'), the file's existing content is erased.

Example: Using the write() method

In this example, we are using the write() method to write the string passed to it to the file. If the file is opened in 'w' mode, it will overwrite any existing content. If the file is opened in 'a' mode, it will append the string to the end of the file −

</> Open Compiler

```
with open("foo.txt", "w") as file:
   file.write("Hello, World!")
   print ("Content added Successfully!!")
```

Output of the above code is as follows −

```
Content added Successfully!!
```

Example: Using the writelines() method

In here, we are using the writelines() method to take a list of strings and writes each string to the file. It is useful for writing multiple lines at once −

```
</>                                                    Open Compiler

lines = ["First line\n", "Second line\n", "Third line\n"]
with open("example.txt", "w") as file:
   file.writelines(lines)
   print ("Content added Successfully!!")
```

The result obtained is as follows −

```
Content added Successfully!!
```

Closing a File in Python

We can close a file in Python using the close() method. Closing a file is an essential step in file handling to ensure that all resources used by the file are properly released. It is important to close files after operations are completed to prevent data loss and free up system resources.

Example

In this example, we open the file for writing, write data to the file, and then close the file using the close() method −

```
</>                                                    Open Compiler

file = open("example.txt", "w")
file.write("This is an example.")
file.close()
print ("File closed successfully!!")
```

The output produced is as shown below –

```
File closed successfully!!
```

Using "with" Statement for Automatic File Closing

The with statement is a best practice in Python for file operations because it ensures that the file is automatically closed when the block of code is exited, even if an exception occurs.

Example

In this example, the file is automatically closed at the end of the with block, so there is no need to call close() method explicitly –

</> Open Compiler

```
with open("example.txt", "w") as file:
   file.write("This is an example using the with statement.")
   print ("File closed successfully!!")
```

Following is the output of the above code –

```
File closed successfully!!
```

Handling Exceptions When Closing a File

When performing file operations, it is important to handle potential exceptions to ensure your program can manage errors gracefully.

In Python, we use a **try-finally** block to handle exceptions when closing a file. The "finally" block ensures that the file is closed regardless of whether an error occurs in the try block –

</> Open Compiler

```
try:
   file = open("example.txt", "w")
   file.write("This is an example with exception handling.")
finally:
```

```
file.close()
print ("File closed successfully!!")
```

After executing the above code, we get the following output −

```
File closed successfully!!
```