File handling in Python

File operations | python3 | file handling | communitycreator

Python provides the basic functions and methods necessary to manipulate files by default. You can do most file manipulation using a file object.

We cannot express the data to be stored in a variable all the time because the variables are volatile in nature. So, to handle such situations, the role of files comes into the picture.

Since files are **non-volatile** in nature, the data will be stored permanently in a secondary device (e.g., **Hard Disk**) and handled with Python in our applications.

We do the operations on files in Python using some built-in methods or functions. With the Python programming language, we can handle both text files and binary files.

The following are the operations on files:

Open a file

Python has an in-built function called open() that opens a file:

```
open('filename', mode)
```

• **`filename`** : Gives the name of the file that the file object has opened.

• **`mode`** : An attribute of a file object that tells you the mode in which a file was opened.

main.py

sample.txt

```
# Opening file to read text (default mode: read)
with open("sample.txt", 'r') as my_file:
    content = my_file.read()    # read() returns
    print("Reading file: ")
    print(content)
```

`open()` takes two arguments, the file that we want to open and a string that represents the kinds of permission or operation we want to do on the file. Here we have taken `r` to open a file in reading mode.

Create a file

```
# creating a file to write text (overwrites previous content if file already
with open("sample.txt", 'w+') as my_file:
    my_file.write("Replaced text!")     # write() writes passed content to f

# Opening file to read text (default mode: read)
with open("sample.txt", 'r') as my_file:
    content = my_file.read()    # read() returns the content of file
    print("Reading file: ")
    print(content)
```

Here we used the `w` letter in our argument, which indicates “write” and will create a file if it does not exist in the library. The `+` sign indicates both read and write mode.

Appending data to the file

main.py

sample.txt

```
# Opening file to append text
with open("sample.txt", 'a+') as my_file:
    my_file.write(" Added more text!")   # write
# Opening file to read text (default mode: read)
with open("sample.txt", 'r') as my_file:
    content = my_file.read()    # read() returns
    print("Reading file: ")
    print(content)
```

The following shows the output of the text file “sample.txt”. You can clearly identify the difference between write and append: append will not replace the data; it will add the data to the end of the file; whereas, write will replace the data.

Closing a file

The `close()` method of a file object flushes any unwritten information and closes the file object.

> After the file object is closed, no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

main.py

sample.txt

```
# Opening file to read text (default mode: read)
with open("sample.txt", 'r') as my_file:
    content = my_file.read()    # read() returns
    print("Reading file: ")
    print(content)
    my_file.close()
```

In the above example, the file object holds the “sample.txt” with reading mode. After writing, we close the file object using the `close()` method.

Renaming a file

Python provides us with an `os` module whose built-in methods help us to perform file operations (e.g., **renaming and deleting**) to the file.

In order to use this module, we need to import the `os` module in our program and then call the related methods.

The `rename()` method accepts two arguments, i.e., the current file name and the new file name.

```
import os
os.rename('sample.txt', 'example.txt')
```

Here, *sample.txt* is the current file name, and *example.txt* is the new file name.

You can specify the path instead of filenames as well.

Deleting a file

We use the `remove()` method to delete the file by supplying the file name or the file location that you want to delete.

```
import os
os.remove('sample.txt')
```

Along with the above functions, Python also has various other functions that help to manipulate the files and its contents.