File Handling in Python

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short. Python treats file differently as text or binary and this is important. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let’s start with Reading and Writing files.

**Working of open() function**

We use **open ()** function in Python to open a file in read or write mode. As explained above, open ( ) will return a file object. To return a file object we use **open()** function along with two arguments, that accepts file name and the mode, whether to read or write. So, the syntax being: **open(filename, mode)**. There are three kinds of mode, that Python provides and how files can be opened:

* “**r**“, for reading.
* “**w**“, for writing.
* “**a**“, for appending.
* “**r+**“, for both reading and writing

One must keep in mind that the mode argument is not mandatory. If not passed, then Python will assume it to be “**r**” by default. Let’s look at this program and try to analyze how the read mode works:

|  |
| --- |
| # a file named "a.txt", will be opened with the reading mode.  file = open('a.txt', 'r')  # This will print every line one by one in the file  for each in file:      print (each) |

The open command will open the file in the read mode and the for loop will print each line present in the file.

**Working of read() mode**

There is more than one way to read a file in Python. If you need to extract a string that contains all characters in the file then we can use **file.read()**. The full code would work like this:

|  |
| --- |
| # Python code to illustrate read() mode  file = open("file.text", "r")  print (file.read()) |

Another way to read a file is to call a certain number of characters like in the following code the interpreter will read the first five characters of stored data and return it as a string:

|  |
| --- |
| # Python code to illustrate read() mode character wise  file = open("file.txt", "r")  print (file.read(5)) |

**Creating a file using write() mode**

Let’s see how to create a file and how write mode works:  
To manipulate the file, write the following in your Python environment:

|  |
| --- |
| # Python code to create a file  file = open('a.txt','w')  file.write("This is the write command")  file.write("It allows us to write in a particular file")  file.close() |

The close() command terminates all the resources in use and frees the system of this particular program.

**Working of append() mode**

Let’s see how the append mode works:

|  |
| --- |
| # Python code to illustrate append() mode  file = open('a.txt','a')  file.write("This will add this line")  file.close() |

There are also various other commands in file handling that is used to handle various tasks like:

rstrip(): This function strips each line of a file off spaces from the right-hand side.

lstrip(): This function strips each line of a file off spaces from the left-hand side.

It is designed to provide much cleaner syntax and exceptions handling when you are working with code. That explains why it’s good practice to use them with a statement where applicable. This is helpful because using this method any files opened will be closed automatically after one is done, so auto-cleanup.  
Example:

|  |
| --- |
| # Python code to illustrate with()  with open("file.txt") as file:      data = file.read()  # do something with data |

**Using write along with with() function**

We can also use write function along with with() function:

|  |
| --- |
| # Python code to illustrate with() alongwith write()  with open("file.txt", "w") as f:      f.write("Hello World!!!") |

**split() using file handling**

We can also split lines using file handling in Python. This splits the variable when space is encountered. You can also split using any characters as we wish. Here is the code:

|  |
| --- |
| # Python code to illustrate split() function  with open("file.text", "r") as file:      data = file.readlines()      for line in data:          word = line.split()          print (word) |