Python

File Handling in Python

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 like other concepts of Python, this concept here is also easy and short. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a 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 the reading and writing files.

Working of open() function

Before performing any operation on the file like reading or writing, first, we have to open that file. For this, we should use Python's inbuilt function open() but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

```
f = open(filename, mode)
```

Where the following mode is supported:

1. **r:** open an existing file for a read operation.
2. **w:** open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.
3. **a:** open an existing file for append operation. It won't override existing data.
4. **r+:** To read and write data into the file. The previous data in the file will be overridden.
5. **w+:** To write and read data. It will override existing data.
6. **a+:** To append and read data from the file. It won't override existing data.

Take a look at the below example:

Python3

```
# a file named "geek", will be opened with the reading mode.
file = open('geek.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:

Python3

```
# Python code to illustrate read() mode
file = open("file.txt", "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:

Python3

```
# 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 to write mode works, so in order to manipulate the file, write the following in your Python environment:

Python3

```
# Python code to create a file
file = open('geek.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 us see how the append mode works:

Python3

```
# Python code to illustrate append() mode
file = open('geek.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 exception 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:

Python3

```
# Python code to illustrate with()
with open('file.txt') as file:
    data = file.read()
# do something with data
```

Using write along with the with() function

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

Python3

```
# 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:

Python3

```
# 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)
```

There are also various other functions that help to manipulate the files and their contents. One can explore various other functions in Python Docs.

Advantages:

- **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:

- **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.