Python provides several ways to manipulate files. Today, we will discuss how to handle files in Python.

Opening a File

Before we can perform any operations on a file, we must first open it. Python provides the open() function to open a file. It takes two arguments: the name of the file and the mode in which the file should be opened. The mode can be 'r' for reading, 'w' for writing, or 'a' for appending.

Here's an example of how to open a file for reading:

```
f = open('myfile.txt', 'r')
```

By default, the open() function returns a file object that can be used to read from or write to the file, depending on the mode.

Modes in file

There are various modes in which we can open files.

1. read (r): This mode opens the file for reading only and gives an error if the file does not exist. This is the default mode if no mode is passed as a parameter.
2. write (w): This mode opens the file for writing only and creates a new file if the file does not exist.
3. append (a): This mode opens the file for appending only and creates a new file if the file does not exist.
4. create (x): This mode creates a file and gives an error if the file already exists.
5. text (t): Apart from these modes we also need to specify how the file must be handled. t mode is used to handle text files. t refers to the text mode. There is no difference between r and rt or w and wt since text mode is the default. The default mode is 'r' (open for reading text, synonym of 'rt').
6. binary (b): used to handle binary files (images, pdfs, etc).

Reading from a File

Once we have a file object, we can use various methods to read from the file.

The read() method reads the entire contents of the file and returns it as a string.

```
f = open('myfile.txt', 'r')
contents = f.read()
print(contents)
```

Writing to a File

To write to a file, we first need to open it in write mode.

```
f = open('myfile.txt', 'w')
```

We can then use the write() method to write to the file.

```
f = open('myfile.txt', 'w')
f.write('Hello, world!')
```

Keep in mind that writing to a file will overwrite its contents. If you want to append to a file instead of overwriting it, you can open it in append mode.

```
f = open('myfile.txt', 'a')
f.write('Hello, world!')
```

Closing a File

It is important to close a file after you are done with it. This releases the resources used by the file and allows other programs to access it.

To close a file, you can use the close() method.

```
f = open('myfile.txt', 'r')
# ... do something with the file
f.close()
```

The 'with' statement

Alternatively, you can use the with statement to automatically close the file after you are done with it.

```
with open('myfile.txt', 'r') as f:
    # ... do something with the file
```

readlines() method

The readline() method reads a single line from the file. If we want to read multiple lines, we can use a loop.

```
f = open('myfile.txt', 'r')
while True:
   line = f.readline()
   if not line:
      break
   print(line)
```

The readlines() method reads all the lines of the file and returns them as a list of strings.

writelines() method

The writelines() method in Python writes a sequence of strings to a file. The sequence can be any iterable object, such as a list or a tuple.

Here's an example of how to use the writelines() method:

```
f = open('myfile.txt', 'w')
lines = ['line 1\n', 'line 2\n', 'line 3\n']
f.writelines(lines)
f.close()
```

This will write the strings in the lines list to the file myfile.txt. The \n characters are used to add newline characters to the end of each string.

Keep in mind that the writelines() method does not add newline characters between the strings in the sequence. If you want to add newlines between the strings, you can use a loop to write each string separately:

```
f = open('myfile.txt', 'w')
lines = ['line 1', 'line 2', 'line 3']
for line in lines:
   f.write(line + '\n')
f.close()
```

It is also a good practice to close the file after you are done with it.

seek() and tell() functions

In Python, the seek() and tell() functions are used to work with file objects and their positions within a file. These functions are part of the built-in io module, which provides a consistent interface for reading and writing to various file-like objects, such as files, pipes, and in-memory buffers.

seek() function

The seek() function allows you to move the current position within a file to a specific point. The position is specified in bytes, and you can move either forward or backward from the current position. For example:

```
with open('file.txt', 'r') as f:
  # Move to the 10th byte in the file
  f.seek(10)

  # Read the next 5 bytes
  data = f.read(5)
```

tell() function

The tell() function returns the current position within the file, in bytes. This can be useful for keeping track of your location within the file or for seeking to a specific position relative to the current position. For example:

```
with open('file.txt', 'r') as f:
  # Read the first 10 bytes
  data = f.read(10)

  # Save the current position
  current_position = f.tell()

  # Seek to the saved position
  f.seek(current_position)
```

truncate() function

When you open a file in Python using the open function, you can specify the mode in which you want to open the file. If you specify the mode as 'w' or 'a', the file is opened in write mode and you can write to the file. However, if you want to truncate the file to a specific size, you can use the truncate function.

Here is an example of how to use the truncate function:

```
with open('sample.txt', 'w') as f:
  f.write('Hello World!')
```

```
  f.truncate(5)

with open('sample.txt', 'r') as f:
  print(f.read())
```