Python File I/O

What is a file?

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

- 1. Open a file
- 2. Read or write (perform operation)
- 3. Close the file

How to open a file?

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
>>> f = open("test.txt")  # open file in current directory
>>> f = open("C:/Python33/README.txt")  # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file.

On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

Python File Modes		
Mode	Description	
'r'	Open a file for reading. (default)	
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.	

'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

```
f = open("test.txt")  # equivalent to 'r' or 'rt'
f = open("test.txt",'w')  # write in text mode
f = open("img.bmp",'r+b')  # read and write in binary mode
```

The default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux.

So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

```
f = open("test.txt", mode = 'r', encoding = 'utf-8')
```

How to close a file Using Python?

When we are done with operations to the file, we need to properly close the file.

Closing a file will free up the resources that were tied with the file and is done using Python close() method.

Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```
f = open("test.txt",encoding = 'utf-8')
# perform file operations
f.close()
```

How to write to File Using Python?

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode.

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

```
f=open("test.txt",'w')
f.write("my first file\n")
f.write("This file\n\n")
f.write("contains three lines\n")
f.close()
```

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten.

We must include the newline characters ourselves to distinguish different lines.

How to read files in Python?

To read a file in Python, we must open the file in reading mode.

There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

```
>>> f = open("test.txt",'r')
>>> f.read(4)  # read the first 4 data
'This'

>>> f.read(4)  # read the next 4 data
' is '

>>> f.read()  # read in the rest till end of file
'my first file\nThis file\ncontains three lines\n'

>>> f.read()  # further reading returns empty sting
```

•

We can see that, the read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading.

We can change our current file cursor (position) using the seek() method. Similarly, the tell() method returns our current position (in number of bytes).

```
>>> f.tell()  # get the current file position
56

>>> f.seek(0)  # bring file cursor to initial position
0

>>> print(f.read())  # read the entire file
This is my first file
This file
contains three lines
```

We can read a file line-by-line using a for loop. This is both efficient and fast.

```
>>> for line in f:
... print(line, end = '')
...
This is my first file
This file
contains three lines
```

The lines in file itself has a newline character '\n'.

Moreover, the print() end parameter to avoid two newlines when printing.

Alternately, we can use readline() method to read individual lines of a file. This method reads a file till the newline, including the newline character.

```
>>> f.readline()
'This is my first file\n'
>>> f.readline()
'This file\n'
```

```
>>> f.readline()
'contains three lines\n'
>>> f.readline()
''
```

Lastly, the readlines() method returns a list of remaining lines of the entire file. All these reading method return empty values when end of file (EOF) is reached.

```
>>> f.readlines()
['This is my first file\n', 'This file\n', 'contains three lines\n']
```

Python File Methods

There are various methods available with the file object. Some of them have been used in above examples.

Here is the complete list of methods in text mode with a brief description.

Python File Methods			
Method	Description		
close()	Close an open file. It has no effect if the file is already closed.		
read(n)	Read atmost n characters form the file. Reads till end of file if it is negative or None.		
readable()	Returns True if the file stream can be read from.		
seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).		
seekable()	Returns True if the file stream supports random access.		
tell()	Returns the current file location.		
truncate(size=None)	Resize the file stream to size bytes. If size is not specified, resize to current location.		
writable()	Returns True if the file stream can be written to.		
write(s)	Write string s to the file and return the number of characters written.		

writelines(lines) Write a list of lines to the file.

Python Directory and Files Management

What is Directory in Python?

If there are a large number of files to handle in your Python program, you can arrange your code within different directories to make things more manageable.

A directory or folder is a collection of files and sub directories. Python has the os module, which provides us with many useful methods to work with directories (and files as well).

Get Current Directory

We can get the present working directory using the getcwd() method.

This method returns the current working directory in the form of a string.

```
>>> import os

>>> os.getcwd()
'C:\\Program Files\\PyScripter'
```

The extra backslash implies escape sequence. The print() function will render this properly.

```
>>> print(os.getcwd())
C:\Program Files\PyScripter
```

Changing Directory

We can change the current working directory using the chdir() method.

The new path that we want to change to must be supplied as a string to this method. We can use both forward slash (/) or the backward slash (\) to separate path elements.

It is safer to use escape sequence when using the backward slash.

```
>>> os.chdir('C:\\Python33')
```

```
>>> print(os.getcwd())
C:\Python33
```

List Directories and Files

All files and sub directories inside a directory can be known using the listdir() method.

This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

```
>>> print(os.getcwd())
C:\Python33
>>> os.listdir()
['DLLs',
'Doc',
'include',
'Lib',
'libs',
'LICENSE.txt',
'NEWS.txt',
'python.exe',
'pythonw.exe',
'README.txt',
'Scripts',
'tcl',
'Tools']
>>> os.listdir('G:\\')
['$RECYCLE.BIN',
'Movies',
'Music',
'Photos',
'Series',
'System Volume Information']
```

Making a New Directory

We can make a new directory using the mkdir() method.

This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

```
>>> os.mkdir('test')
>>> os.listdir()
['test']
```

Renaming a Directory or a File

The rename() method can rename a directory or a file.

The first argument is the old name and the new name must be supplies as the second argument.

```
>>> os.listdir()
['test']
>>> os.rename('test','new_one')
>>> os.listdir()
['new_one']
```

Removing Directory or File

A file can be removed (deleted) using the remove() method.

Similarly, the rmdir() method removes an empty directory.

```
>>> os.listdir()
['new_one', 'old.txt']
>>> os.remove('old.txt')
>>> os.listdir()
```

```
['new_one']

>>> os.rmdir('new_one')

>>> os.listdir()
[]
```

However, note that rmdir() method can only remove empty directories.

In order to remove a non-empty directory we can use the rmtree() method inside the shutil module.

```
>>> os.listdir()
['test']

>>> os.rmdir('test')
Traceback (most recent call last):
...

OSError: [WinError 145] The directory is not empty: 'test'

>>> import shutil
>>> shutil.rmtree('test')
>>> os.listdir()
[]
```

File Example - PhoneBook

```
def print_numbers(numbers):
    print("Telephone Numbers:")
    for k, v in numbers.items():
        print("Name:", k, "\tNumber:", v)
    print()

def add_number(numbers, name, number):
    numbers[name] = number

def lookup_number(numbers, name):
    if name in numbers:
        return "The number is " + numbers[name]
    else:
        return name + " was not found"
```

```
def remove number(numbers, name):
    if name in numbers:
        del numbers[name]
    else:
        print(name," was not found")
def load numbers(numbers, filename):
    in file = open(filename, "rt")
    while True:
        in line = in file.readline()
        if not in line:
            break
        in line = in line[:-1] # remove \n characters
        name, number = in line.split(",")
        numbers[name] = number
    in file.close()
def save numbers(numbers, filename):
    out file = open(filename, "wt")
    for k, v in numbers.items():
        out file.write(k + ", " + v + " \setminus n")
    out file.close()
def print menu():
   print('1. Print Phone Numbers')
   print('2. Add a Phone Number')
   print('3. Remove a Phone Number')
   print('4. Lookup a Phone Number')
   print('5. Load numbers')
   print('6. Save numbers')
   print('7. Quit')
   print()
phone list = {}
menu choice = 0
print menu()
while True:
    menu choice = int(input("Type in a number (1-7): "))
    if menu choice == 1:
        print_numbers(phone_list)
    elif menu choice == 2:
        print("Add Name and Number")
        name = input("Name: ")
        phone = input("Number: ")
        add number (phone list, name, phone)
    elif menu choice == 3:
        print("Remove Name and Number")
```

```
name = input("Name: ")
        remove number(phone list, name)
    elif menu_choice == 4:
        print("Lookup Number")
        name = input("Name: ")
        print(lookup_number(phone_list, name))
    elif menu_choice == 5:
        filename = input("Filename to load: ")
        load_numbers(phone_list, filename)
    elif menu_choice == 6:
        filename = input("Filename to save: ")
        save numbers(phone list, filename)
    elif menu_choice == 7:
        break
    else:
        print menu()
print("Goodbye")
```