Python File Open

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

File Handling

The key function for working with files in Python is the `open()` function.

The `open()` function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

`"t"` - Text - Default value. Text mode

`"b"` - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")    OR

f = open("demofile.txt", "rt")
```

Because `"r"` for read, and `"t"` for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Open a File

Assume we have the following “demofile.txt” file, located in the same folder as Python:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

To open the file, use the built-in `open()` function.

The `open()` function returns a file object, which has a `read()` method for reading the content of the file:

```
f = open("demofile.txt", "r")
print(f.read())
```

Output:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Read Specific Parts of the File

By default the `read()` method returns the whole text, but you can also specify how many characters you want to return:

Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

```
Output: Hello
```

Read One Line

You can return one line by using the `readline()` method:

```
f = open("demofile.txt", "r")
print(f.readline())
```

```
Output: Hello! Welcome to demofile.txt
```

By calling `readline()` two times, you can read the two first lines:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

```
Output:
```

Hello! Welcome to demofile.txt
This file is for testing purposes.

By looping through the lines of the file, you can read the whole file, line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Output:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Close Files

It is a good practice to always close the file when you are done with it.

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Write to an Existing File

To write to an existing file, you must add a parameter to the `open()` function:

`"a"` - Append - will append to the end of the file

`"w"` - Write - will overwrite any existing content

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Output:

```
Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
```

```
#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Output: `Woops! I have deleted the content!`

Note: the "w" method will overwrite the entire file.

writelines() to write in File

```
lst=[]
for i in range(2):
    n=input("Enter Name: ")
    lst.append(n+"\n")
f=open("test","w")
f.writelines(lst)
f.close()
print(open("test","r").read())
```

Create a New File

To create a new file in Python, use the `open()` method, with one of the following parameters:

`"x"` **- Create - will create a file, returns an error if the file exist**

`"a"` **- Append - will create a file if the specified file does not exist**

`"w"` **- Write - will create a file (if file exist then delete and create new)**

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Create a new file if it does not exist:

```
f = open("myfile.txt", "a")
```

Python Delete File

To delete a file, you must import the OS module, and run its `os.remove()` function:

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the `os.rmdir()` method:

```
import os
os.rmdir("myfolder")
```

Note: You can only remove *empty* folders.

Working with Binary Files

Modes of Binary File:

1. b = Open the binary file
2. rb = Open binary file in read only mode (file must be existed)
3. wb = Open binary file in write only mode (new file created, over written if existed)
4. ab = Open binary file in append (write only) mode. (open if existed otherwise create new file)
5. rb+ = Open binary file in read & write mode. (file must be existed)
6. wb+ = Open binary file in write & read mode. (new file created, over written if existed)
7. ab+ = Open binary file in append (write & read) mode. (open if existed otherwise create new file)

To store object like dictionary, list, class object into binary file python provide a package that is pickle. Python provides pickle with very powerful algorithm to manage all the binary operations on file. The pickle package should be import in program to use the functions of pickle. The two operations are belongs to pickle.

1. Pickling: python object structure is converted into byte stream to store in physical media i.e. in file at hard disk.
2. Un–Pickling: it is inverse operation of pickling where the byte stream converted back into an object structure.

There are two type of functions which mostly used for binary operations on file.

1. **dump() : it is used for writing purpose in file**
2. **load() : it is used for reading purpose from file**

Syntax:

```
import pickle
pickle.dump(object of structure, object of file-handler)
```

where object of structure may be list, tuple, dictionary and class object

Example-1:

Write the list of numbers into binary file and read it.

```
import pickle
lst=[]
for i in range(2):
   n=int(input("Enter No.: "))
   lst.append(n)
f=open("test","wb")
pickle.dump(lst,f)
f.close()
```

```
f=open("test","rb")
lst=pickle.load(f)
sum=0
for x in lst:
    sum+=x
print("List in file\n",lst)
print("sum of list: ",sum)
print("Max No. in list: ",max(lst))
```

Output:

```
Enter No.: 10
Enter No.: 20
List in file
 [10, 20]
sum of list:  30
Max No. in list:  20
```

Example-2: **Write program to write dictionary in binary file.**

```
import pickle
dic={'RN':[],'name':[]}
for i in range(1):
   n1=int(input("Enter First No.: "))
   n2=int(input("Enter Second No.: "))
   dic['RN']=[n1,n2]
   nm1=input("Enter First name: ")
   nm2=input("Enter Second name: ")
   dic['name']=[nm1,nm2]
f=open("test","ab")
pickle.dump(dic,f)
f.close()
f=open("test","rb")
lst=pickle.load(f)
print("Dictionary in file\n",dic)
print("List of Keys of dictionary: ",dic.keys())
print("List of Values of dictionary: ",dic.values())
```

Output:

```
Enter First No.: 10
Enter Second No.: 20
Enter First name: seeta
Enter Second name: Ram
Dictionary in file
 {'name': ['seeta', 'Ram'], 'RN': [10, 20]}
List of Keys of dictionary:  dict_keys(['name', 'RN'])
List of Values of dictionary:  dict_values([['seeta', 'Ram'], [10, 20]])
```