File I/O

File I/O

`open()` Function

The `open()` function is the Standard Library option for reading and writing both text and binary files. It returns a file object, the exact type depending on the type of file you read.

The file object has methods for reading from and writing to the file.

- The file object is iterable
- Signature `python  open(filename, mode = 'r')`

File Modes

The different modes for `open()` are (taken from the docstring):

Character	Meaning
'r'	open for reading (**default**)
'w'	open for writing, truncating the file first
'x'	create a new file and open it for writing
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (**default**)
'+'	open a disk file for updating (reading and writing)
'U'	universal newline mode (deprecated)

Modes can be combined. For example, the default mode is `'rt'`, or "read text-file". If you wanted to open a binary file in write mode, you could use `'wb'`. See the docstring for more of an explanation.

Open Files in a `with` Statement

A good practice when opening files with `open()` is to use a with statement:

```
with open('new_file.txt', 'w') as f:
    #file object can be used here
#file object is closed
```

Here the variable `f` refers to the file object that `open()` returns. When control leaves the `with` statement, the file is closed (see the section at the bottom of this page on how to do this manually). Outside the `with` the file object will still exist, but you won't be able to read from or write to it.

Writing to Files

You can write to files using the `'w'` mode in `open()`. This creates a new file if the file specified doesn't already exist, or **over-writes** the file if it already does (replacing the content). To write to the file use the `.write()` method on the file object:

[1]:
```
with open('new_file.txt', 'w') as f:
    f.write('A line of text in the file.')
```

In the example above we have written to a file called `'new_file.txt'`. This file will be located in the same directory as your script/notebook. The contents of the file is a single line:

```
A line of text in the file
```

The `.rite()` method writes strings to the file. Unlike `print`, `write` **only** takes strings. Also, if you want to write to a new line you need to use the new line special character `'\n'`:

[2]:
```
with open('new_file.txt', 'w') as f:
    f.write('First line of the file.\n')
    f.write('Second line of the file. ')
    f.write('This is still the second line of the file.')
```

The contents of **new_file.txt** now reads as follows:

```
First line of the file.
Second line of the file. This is still the second line of the file.
```

Alternatively you could write the contents as a multi-lined string literal:

[3]:
```
with open('new_file.txt', 'w') as f:
    f.write('''First line.
Second line.
Third line.''')
```

As you can see the string literal above does not have indentations. This is because those indentations would be a part of the string literal itself. Multi-lined string literals can, thus make it difficult to read indented code blocks. The contents of **new_file.txt** now reads:

```
First line.
Second line.
Third line.
```

Reading Files

You can read a function using the `'r'` mode of `open()`. The file object returned has a few options for reading the content.

The `.read()` Method

If you want to read the entire contents of the file into a single string, you can use the `.read()` method of the file object:

[4]:
```
with open('new_file.txt', 'r') as f:
    data = f.read()

print(data)
```

```
First line.
Second line.
Third line.
```

Note that the file object keeps track of where you have read to in the file. When you have reached the end of the file (as is the case after using `.read()`) you cannot read more content.

[5]:
```
with open('new_file.txt', 'r') as f:
    data1 = f.read()
    data2 = f.read() #A second reading

print('First reading:')
print(data1)
print('')
print('Second reading')
print(data2)
```

```
First reading:
First line.
Second line.
Third line.

Second reading
```

The `.readline()` Method

If you want to read the next line of the file object, you can use the `.readline()` method:

[10]:
```
with open('new_file.txt', 'r') as f:
    print('1', f.readline())
    print('2', f.readline())
```

```
1 First line.

2 Second line.
```

The `.readlines()` Method

If you want to create a list of the lines in the file, you can use the `.readlines()` method:

[11]:
```
with open('new_file.txt', 'r') as f:
    lines = f.readlines()
```

```
print(lines)
```

```
['First line.\n', 'Second line.\n', 'Third line.']
```

Iterating Through File Objects

The file object returned by `open()` is iterable. Each iteration call returns a line of the file. We can use this in a `for` loop:

[6]:
```
with open('new_file.txt', 'r') as f:
    for line in f:
        print(line)
```

```
First line.

Second line.

Third line.
```

Let's print the corresponding line numbers of each line to further illustrate what is happening:

[8]:
```
with open('new_file.txt', 'r') as f:
    for i,line in enumerate(f):
        print(i+1, line)
```

```
1 First line.

2 Second line.

3 Third line.
```

Opening Files Without `with`

If, for some reason, you don't want to make use of the `with` statement when opening your files, make sure to close your file objects when you are done with them:

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

#file object used

f.close()
```