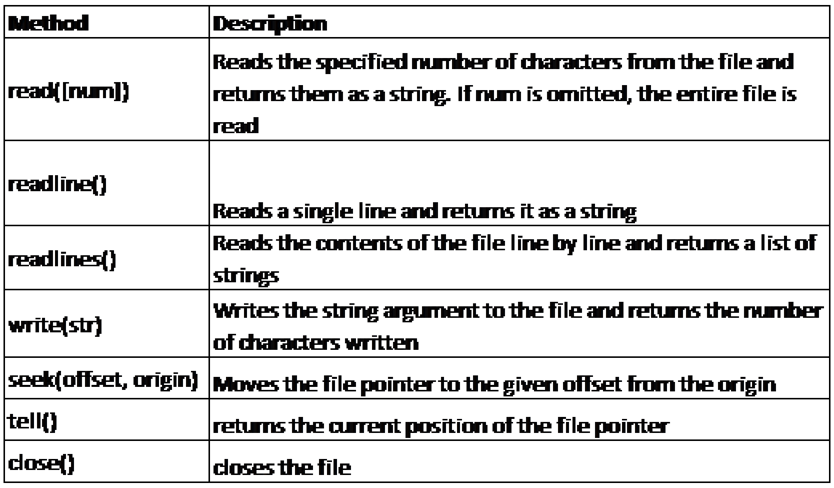
Files - II

In this notebook, we will look at how to read a text file.

The file handle returned when we use the `open()` method is an object of the `_io.TextIOWrapper` class. The table below contains some of the more commonly used methods in this class.



Method	Description
read([num])	Reads the specified number of characters from the file and returns them as a string. If num is omitted, the entire file is read
readline()	Reads a single line and returns it as a string
readlines()	Reads the contents of the file line by line and returns a list of strings
write(str)	Writes the string argument to the file and returns the number of characters written
seek(offset, origin)	Moves the file pointer to the given offset from the origin
tell()	returns the current position of the file pointer
close()	closes the file

In this cell we look at the syntax for reading a text file.

To read from a file, we use the mode value 'r'. Note that this is the default value for the `open()` function and can therefore be omitted. If the file does not exist, the program will generate errors and halt. It is therefore a good idea to enclose the syntax within `try/catch` blocks (which we will discuss later).

The `read()` method is then used to read the contents of the whole file.

To close the file, use the `close()` method.

In [24]:

```
f = open('out.txt', 'r') #open for read

file_data = f.read()
print(file_data)
print(type(file_data))

f.close()
```

```
This is line 0
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
This is line 8
This is line 9
This is line 10

<class 'str'>
```

The `read()` method has an optional 'num' parameter. If this value is provided, the `read()` method reads the specified number of characters. Otherwise, as seen in the example above the entire file is read.

In [25]:

```
f = open('out.txt', 'r')
file_data = f.read(20)
print(file_data)
f.close()
```

```
This is line 0
This
```

This cell demonstrates the use of the `tell()` method. The `tell()` method returns the current position of the file pointer.

In [26]:

```python
'''
  Print out what has been read so far and then print a blank line.
'''
f = open('out.txt', 'r')
print('Current position of the file pointer:', f.tell())
file_data = f.read(20)

#Print out what has been read so far and then print a blank line
print('After reading 20 characters: \n', file_data + '\n', sep='')

# Returns the current position of the file pointer
print('Current position of the file pointer:', f.tell())
f.close()
```

```
Current position of the file pointer: 0
After reading 20 characters:
This is line 0
This

Current position of the file pointer: 21
```

This cell demonstrates the use of the `seek()` method. The `seek()` method accepts two arguments.

The seek() method resets the file pointer to the position indicated by the first argument.
The second argument can be: 0, 1, or 2.
If the second argument is 0, the position is relative to the start of the file.
If the second argument is 1, the position is relative to where the pointer is currently located.
If the second argument is 2, the first argument must be zero and the pointer points to the end of the file.

However only the default value of 0 is valid for text files. This is the only one we will consider in this course. The other two options (1 and 2) only work when the text file is open in binary mode.

In [27]:

```python
'''
  Use of the seek() method.
'''
f = open('out.txt', 'r')
print('position of the file pointer before the seek() method', f.tell(), '\n')

f.seek(10, 0) # This will reset the pointer to the 11th character from the start
print('position of the file pointer after the seek() method', f.tell(), '\n')

#Contents of the rest of the file after repositioning the pointer
print(f.read())
f.close()
```

```
position of the file pointer before the seek() method 0

position of the file pointer after the seek() method 10

ne 0
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
This is line 8
This is line 9
This is line 10
```

The `readline()` method is then used to read only a single line, where the current position of the file pointer is.

In [28]:

```
'''
  Print out what has been read so far and then print a blank line.
'''
f = open('out.txt', 'r')
print('Current position of the file pointer:', f.tell())

one_line = f.readline()
print(one_line)

#Print out what has been read so far and then print a blank line
file_data = f.read(30)
print('After reading 30 characters: \n', file_data, sep='')

one_line = f.readline()
print(one_line)

# Returns the current position of the file pointer
print('Current position of the file pointer:', f.tell())
f.close()
```

```
Current position of the file pointer: 0
This is line 0

After reading 30 characters:
This is line 1
This is line 2

This is line 3

Current position of the file pointer: 64
```

The `read()` method used above reads the entire file as one unit and hence each line in the file cannot be easily processed separately.
If you want to read and process each line separately, you can do that by using the 'for' loop to iterate through the file object.

In order for the 'for' loop in the cell below to work as desired, the data should have been written to the file with each line ending with a newline character to signal the end of the line.

In [29]:

```
'''
  Read each line separately
'''

f = open('out.txt')  # Note that I have omitted the mode argument since the default is anyway 'r'

for line in f:
    print(line, end = '')

f.close()
```

```
This is line 0
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
This is line 8
This is line 9
This is line 10
```

The `readlines()` method is used to pack lines together. Iterating through the lines, we unpack them, and print them one

at a time. The code below also processes every line separately.

In [30]:

```
'''
  Read each line separately
'''
f = open('out.txt')

for line in f.readlines():
    print(line, end = '')

f.close()
```

```
This is line 0
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
This is line 8
This is line 9
This is line 10
```

The code below also processes every line separately.

In [31]:

```
'''
  Read each line separately
'''
f = open('out.txt')

for line in f.read():
    print(line, end = '')

f.close()
```

```
This is line 0
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
This is line 8
This is line 9
This is line 10
```