Files

We have, up to now, been storing data only in the variables and data structures of programs. However, such data is not available once a program terminates. Therefore, in order for information to persist from one program execution to the next, the data must be stored in a data file.

Python uses file objects to interact with external files on your computer. These file objects can be any sort of file you have on your computer, whether it be an audio file, a text file, emails, Excel documents, etc. Note: You will probably need to install certain libraries or modules to interact with those various file types, but they are easily available.

In this chapter we discuss the use of one particular type of data file, text files.

Python has a built-in open function that allows us to open and play with basic file types.

IPython Writing a File

This function is specific to jupyter notebooks! Alternatively, quickly create a simple .txt file with *notepad* text editor.

In [ ]:
```
pwd  # print working dir.
```

In [ ]:
```
cd ..  # previous directory
```

In [ ]:
```
cd CH7 # sub directory
```

In [85]:
```
cd cem # if not exist
```

```
[WinError 2] The system cannot find the file specified: 'cem # if not exist'
C:\Users\Cem Ergun\2023_2024_fall\CH7
```

In [ ]:
```
pwd
```

In [ ]:
```
ls # list files and folders
```

In [ ]:
```
# giving full path
cd "/Users/Cem Ergun/2023_2024_Fall/CH7"
```

In [ ]:
```
# quick way to create file from Jupyter only
```

In [98]:
```
%%writefile myfile.txt
Hello, this is a quick test file.
CMPE107
fall 2023/2024
good bye
```

```
Overwriting myfile.txt
```

In [99]:
```
pwd # check existing directory
```

Out[99]:
```
'C:\\Users\\Cem Ergun\\2023_2024_fall\\CH7'
```

Python Opening a file

Let's being by opening the file myfile.txt that is located in the same directory as this notebook. For now we will work with files located in the same directory as the notebook or .py script you are using.

It is very easy to get an error on this step: if file is not available in existing directory

In [103]:
```
my_file_id = open('myfile.txt','r')
```

In [104]:
```
my_file_id.close() # close the file
```

In [105]:
```
my_file_id
```

Out[105]:
```
<_io.TextIOWrapper name='myfile.txt' mode='r' encoding='cp1254'>
```

- The first argument is the file name to be opened, 'myfile.txt'.
- The second argument, 'r', indicates that the file is to be opened for reading. (The second argument is optional when opening a file for reading.)
- If the file is successfully opened, a file object is created and assigned to the provided identifier, in this case identifier my_file.

To avoid this error,make sure your .txt file is saved in the same location as your notebook, to check your notebook location, use **pwd**:

- to show current directory
- pwd Print Work Directory"

However, an alternate location can be specified in the call to open by providing a path to the file,

In [106]:
```
my_file = open("C:\\Users\\Cem Ergun\\2023_2024_Fall\\CH7\\myfile.txt", 'r')
```

In [107]:
```
my_file.close()
```

**Alternatively, to grab files from any location on your computer, simply pass in the entire file path. **

For Windows you need to use double \ so python doesn't treat the second \ as an escape character, a file path is in the form:

```
my_file = open("C:\\Users\\YourUserName\\myfile.txt")
```

For MacOS and Linux you use slashes in the opposite direction:

```
my_file = open("/Users/YouUserName/myfile.txt")
```

The first argument is the file name to be opened, 'myfile.txt'. The second argument, 'r', indicates that the file is to be opened for reading. (The second argument is optional when opening a file for reading.) If the file is successfully opened, a file object is created and assigned to the provided identifier, in this case identifier my_file.

In [108]:
```
# Open the myfile.txt we made earlier
my_file = open('myfile.txt')
```

In [109]:
```
# We can now read all string in the file
content = my_file.read()
```

In [110]:
```
content
```

Out[110]:
```
'Hello, this is a quick test file.\nCMPE107\nfall 2023/2024\ngood bye\n'
```

In [111]:
```
print(content)
```

```
Hello, this is a quick test file.
CMPE107
fall 2023/2024
good bye
```

In [112]:
```
A= content.split('\n')  # spliting string with \n
```

In [113]:
```
A[0] # first line text
```

Out[113]:
```
'Hello, this is a quick test file.'
```

In [114]:
```
line = A[0].split(' ')  # split into words
line
```

Out[114]:
```
['Hello,', 'this', 'is', 'a', 'quick', 'test', 'file.']
```

In [115]:
```
print(content)
```

```
Hello, this is a quick test file.
CMPE107
fall 2023/2024
good bye
```

In [116]:
```
content1 = my_file.read()  # read all content again
```

In [117]:
```
content1 # Since pointer shows end nothing to read
```

Out[117]:
```
''
```

In [118]:
```
# But what happens if we try to read and read it again???
my_file.read()
```

Out[118]:
```
''
```

This happens because you can imagine the reading "cursor" is at the end of the file after having read it. So there is nothing left to read. We can

In [119]:
```python
# Seek to the start of file (index 0)
my_file.seek(0)
```
Out[119]:
```
0
```

In [120]:
```python
# Now read first 10 characters
my_file.read(10)
```
Out[120]:
```
'Hello, thi'
```

In [121]:
```python
my_file.read(10) # keep reading
```
Out[121]:
```
's is a qui'
```

You can read a file line by line using the readlines method. Use caution with large files, since everything will be held in memory. We will learn how to iterate over large files later in the course.

In [122]:
```python
# Readlines returns a list of the lines in the file
my_file.seek(0)
content2=my_file.readlines()
```

In [123]:
```python
content2
```
Out[123]:
```
['Hello, this is a quick test file.\n',
 'CMPE107\n',
 'fall 2023/2024\n',
 'good bye\n']
```

In [125]:
```python
# use replace method for unwanted charters
#content3= content2[0].replace('\n', '') # replace new line with empty string
content3= content3.replace('.', '')  # replace dot with empty string
content3
```
Out[125]:
```
'Hello, this is a quick test file'
```

When you have finished using a file, it is always good practice to close it.

In [126]:
```python
my_file.close()
```

In [127]:
```python
# lets open it again to see the .readline() method
my_file = open('/Users/Cem Ergun/2023_2024_Fall/CH7/myfile.txt')
```

Lets use new method **.readline()**

In [128]:
```python
my_file.readline()
```
Out[128]:
```
'Hello, this is a quick test file.\n'
```

In [129]:
```python
my_file.readline()
```
Out[129]:
```
'CMPE107\n'
```

In [130]:
```python
my_file.readline()
```
Out[130]:
```
'fall 2023/2024\n'
```

In [131]:
```python
my_file.readline()
```
Out[131]:
```
'good bye\n'
```

In [133]:
```python
# End of file is reached
my_file.readline()
```
Out[133]:
```
''
```

In [134]:
```python
# go back to first position
my_file.seek(0)
```
Out[134]:
```
0
```

In [135]:
```python
my_file.readline()
```
Out[135]:
```
'Hello, this is a quick test file.\n'
```

```
In [136]: my_file.close()
```

we can read text line by line using `readline()` method

- The readline method returns as a string the next line of a text file, including the end-of-line character, \n.
- When the end-of-file is reached, it returns an empty string

```
In [137]:
my_file = open('myfile.txt','r')
empty_str=''  # stopping criteria

line = my_file.readline()  # first line

while line!= empty_str:  # keep reading till end of line
    print(line, end="")
    line = my_file.readline()  # read nextline

my_file.close()
```

```
Hello, this is a quick test file.
CMPE107
fall 2023/2024
good bye
```

or `readlines()` method

- The method readlines() reads until EOF using readline() and
- This method returns a list containing the lines.

```
In [138]:
my_file = open('myfile.txt','r')

lines = my_file.readlines()
print(lines)

my_file.close()
```

```
['Hello, this is a quick test file.\n', 'CMPE107\n', 'fall 2023/2024\n', 'good bye\n']
```

```
In [139]: lines[1]
```

```
Out[139]: 'CMPE107\n'
```

- Finally, note the blank lines in the screen output. Since read_line returns the newline character, and print adds a newline character, two newline characters are output for each line displayed, resulting in a skipped line after each. We will see an easy way to correct this when we discuss string methods.

or `read()` method

The method read() reads at most size bytes from the file. If the read hits EOF before obtaining size bytes, then it reads only available bytes.

```
In [140]:
my_file = open('myfile.txt','r')
message=my_file.read()
#print(my_file.read())
#print(my_file.read(20))
my_file.close()
```

```
In [141]: message
```

```
Out[141]: 'Hello, this is a quick test file.\nCMPE107\nfall 2023/2024\ngood bye\n'
```

It is also possible to read the lines of a file by use of the `for` statement,

```
In [142]:
input_file = open('myfile.txt','r')
for line in input_file:
    print(line, end='')  # Extra New line is removed
    #print(line)
input_file.close()
```

```
Hello, this is a quick test file.
CMPE107
fall 2023/2024
good bye
```

Writing to a File

By default, the `open()` function will only allow us to read the file. We need to pass the argument `'w'` to write over the file. For example:

In [143]:
```python
empty_str=''
input_file = open('myfile.txt','r')
output_file = open('myfile_copy.txt','w')
line = input_file.readline()

while line!= empty_str:
    print(line, end ='')
    output_file.write(line)
    line = input_file.readline()

input_file.close()
output_file.close()
```

```
Hello, this is a quick test file.
CMPE107
fall 2023/2024
good bye
```

This code copies the contents of the input file, 'myfile.txt', line by line to the output file, 'myfile_copy.txt'.

- In contrast to print when writing to the screen, the write method does not add a newline character to the output string.

Lets Try it

In [144]:
```python
input_file = open('myfile.txt','r')
output_file = open('newfile.txt','w')

line = input_file.readline()
output_file.write(line)

line = input_file.readline()
output_file.write(line)

line = input_file.readline()
output_file.write(line)

input_file.close()
output_file.close()
```

In [ ]:
```python
# OR writing all content to output file
```

In [145]:
```python
empty_str=''
input_file = open('myfile.txt','r')
output_file = open('newfile.txt','w')
lines = input_file.read()  # read all lines at once

output_file.write(lines)   # write all lines at once
input_file.close()
output_file.close()
```

String Traversal

- We saw in Chapter 4 how any sequence can be traversed, including strings.
- This is usually done by the use of a for loop. For example,
- if we want to read a line of a text file and determine the number of blank characters it contains, we could do the following,

In [146]:
```python
spaces = ' '
count = 0

input_file = open('newfile.txt','r')

line = input_file.readline()
print(line)
for k in range(0,len(line)):
    if line[k] == spaces:
        count = count + 1

print('number of spaces=' , count)
input_file.close()
```

```
Hello, this is a quick test file.

number of spaces= 6
```

In [147]:
```
string = 'Hello, this is a quick test file.'
```

In [148]:
```
string.count(' ')
```

Out[148]:
```
6
```

We also saw that the traversal can be done more simply without the explicit use of an index variable,

In [149]:
```
space = ' '
num_spaces = 0

input_file = open('newfile.txt','r')

line = input_file.readline()
print(line)
for Chr in line:
    if Chr == space:
        num_spaces = num_spaces + 1

print('number of spaces=' , num_spaces)
input_file.close()
```

```
Hello, this is a quick test file.

number of spaces= 6
```

In [ ]:
```
Or using string methods .count()
```

In [150]:
```
some_text='My nAme is Ahmed'
numberOfBlank=some_text.count('A')
print(numberOfBlank)
```

```
2
```

In [153]:
```
# Total number of spaces
input_file = open('newfile.txt','r')
num_spaces = input_file.read().count(' ')
input_file.close()
```

In [154]:
```
print(num_spaces)
```

```
8
```

In [156]:
```
# word with shortest length in test
min(some_text.split(), key=len)
```

Out[156]:
```
'My'
```

In [ ]:
```
# Or Lets count number of blank charters in evey line
```

In [157]:
```
empty_str=''
space = ' '
num_spaces = 0

input_file = open('newfile.txt','r')
line = input_file.readline()
while line!= empty_str:
    print(line, end='')
    for chr in line:
        if chr == space:
            num_spaces = num_spaces + 1
    line = input_file.readline()

print('number of spaces=' , num_spaces)
input_file.close()
```

```
Hello, this is a quick test file.
CMPE107
fall 2023/2024
good bye
number of spaces= 8
```

In [158]:
```
# lets save number of spaces for each line in a list
empty_str=''
space = ' '
spacesList = []

input_file = open('newfile.txt','r')
line = input_file.readline()
while line!= empty_str:
    print(line, end='')
    num_spaces = 0
    for chr in line:
        if chr == space:
            num_spaces = num_spaces + 1
    spacesList.append(num_spaces)
    line = input_file.readline()

print('number of spaces=' , spacesList)
input_file.close()
```

```
Hello, this is a quick test file.
CMPE107
fall 2023/2024
good bye
number of spaces= [6, 0, 1, 1]
```

In [ ]:
```
# Or counting spaces using string method
```

In [159]:
```
empty_str=''
space = ' '

input_file = open('newfile.txt','r')
line = input_file.readline()
while line!= empty_str:
    num_spaces = 0
    print(line, end='')
    num_spaces+=line.count(' ')
    line = input_file.readline()

    print('number of spaces=' , num_spaces)
input_file.close()
```

```
Hello, this is a quick test file.
number of spaces= 6
CMPE107
number of spaces= 0
fall 2023/2024
number of spaces= 1
good bye
number of spaces= 1
```

In [160]:
```
# Add a second argument to the function, 'w' which stands for write.
# Passing 'w+' lets us read and write to the file

my_file = open('myfile.txt','w+')
```

In [161]:
```
# lets write a line
my_file.write('This is a first line\n')
```

Out[161]:
```
21
```

Use caution!

Opening a file with `'w'` or `'w+'` truncates the original, meaning that anything that was in the original file **is deleted**!

In [162]:
```
# Write to the file again
my_file.write('This is a second line\n')
```

Out[162]:
```
22
```

In [163]:
```
# Read the file
my_file.seek(0)
my_file.read()
```

Out[163]:
```
'This is a first line\nThis is a second line\n'
```

In [164]:
```
my_file.close()  # always do this when you're done with a file
```

Appending to a File

Passing the argument `'a'` opens the file and puts the pointer at the end, so anything written is appended. Like `'w+'`, `'a+'` lets us read and write to a file. If the file does not exist, one will be created.

In [165]:
```
my_file = open('myfile.txt','a+')
my_file.write('n\This is text being appended to myfile.txt')
my_file.write('\nAnd another line here.')
```

Out[165]:
```
23
```

In [166]:
```
my_file.seek(0)
print(my_file.read())
```

```
This is a first line
This is a second line
n\This is text being appended to myfile.txt
And another line here.
```

In [167]:
```
my_file.close()
```

Appending with `%%writefile`

We can do the same thing using IPython cell magic:

In [168]:
```
%%writefile -a myfile.txt

This is text being appended to myfile.txt from Interactive mode
And another line here.
```

```
Appending to myfile.txt
```

Add a blank space if you want the first line to begin on its own line, as Jupyter won't recognize escape sequences like `\n`

Iterating through a File

Lets get a quick preview of a for loop by iterating over a text file. First let's make a new text file with some IPython Magic:

In [169]:
```
%%writefile myfile.txt
First Line
Second Line
```

```
Overwriting myfile.txt
```

Now we can use a little bit of flow to tell the program to for through every line of the file and do something:

In [170]:
```
for line in open('myfile.txt'):
    print(line)
```

```
First Line

Second Line

```

It's important to note a few things here:

1. We could have called the "line" object anything (see example below).
2. By not calling `.read()` on the file, the whole text file was not stored in memory.
3. Notice the indent on the second line for print. This whitespace is required in Python.

In [171]:
```
# Pertaining to the first point above
for file_id in open('myfile.txt'):
    print(file_id, end='')
#file_id.close()
```

```
First Line
Second Line
```