FILES

The variables we create in our program remain in memory until the end of the program. They are lost when the program terminates. We require files to make the information permanent. We call this feature persistence. The files persist not only the program termination, but also rebooting or restarting a computer.

The files belong to the operating system that runs the computer. The naming convention depends on the operating system. In Microsoft Windows, The total file name (called the path) has the drive name, then ":", then \directory name – this could repeat number of times – then the filename. In linux, we have no drive name. The file name(path) has /directory name – repeats number of times -then the filename.

To use a file in whose name depends on the operating system, in Python, we make an association of an entity called file object with the file. This file object refers to some data structures outside of our program which in turn refers the file. We call this entity a resource. We get the resource when we want using a function called open. We should release this whenever we do not require any more. This is like taking loan from a bank and repaying it. We release the file resource by calling a function called close on the file object.

A file may be used for reading if it already exists. A file may be created or overwritten. A file may be used for reading as well as writing. We indicate our intention by specifying what is called the mode - 'r' for read, 'w' for write. There a few more modes – we shall not discuss in this course.

A file opening may not always succeed. If it fails, it throws an exception. A few possible exceptions:

- opening a non-existent file for reading
- opening a file for writing where directory does not allow us to create files

Let us examine an example for opening a non-existent file for reading.

```
# filename : 1_file.py
# we specify the physical filename - the name by which the operating system
#       know the file - in a function called open.
# we open file in different modes
#       reading(r), writing(w), appending(a) ...
# we get a file handle as the result of opening.
# we use the file handle from that point onwards

#"""
# gives a runtime error if the file does not exist and we try to open the file for
reading
f = open("junk.txt", "r")
# FileNotFoundError: [Errno 2] No such file or directory: 'junk.txt'
#"""

""""
# Let us an existing file for opening.
# An open file uses the resources of the operating system.
# We return the resources by calling a function called close on the open file
handle.
f = open("t.txt", "r")
print(f, type(f))  # do not worry about this output!
f.close()
"""
```

Observe the signature of the function open. The first argument is the filename as per the rules of the operating system. It is a string. The second argument is the mode. The function returns a handle to the file – file object – if it succeeds.

```
f = open("junk.txt", "r")
```
Program terminates with the exception FileNotFoundError.

What if we open a file which exists for reading? We get a file object.
```
f = open("t.txt", "r")
print(f, type(f))  # do not worry about this output!
f.close()
```

The object f is said to be a wrapper for something. Let us ignore it.

The last statement f.close() releases the resources associated with the file object.

Reading a file:

We can read an existing file in a number of ways. We shall examine these in the next few sections. We shall consider the file t.txt for experimenting.

t.txt

```
do not
trouble trouble
till trouble
troubles you
```

We shall now examine parts of the file 2_file_read.py.

```
f = open("t.txt", "r")
line1 = f.readline()
print("first line : ", line1)  # do not
line2 = f.readline()
print("second line : ", line2) # trouble trouble
line1 = line1.strip();  line2 = line2.strip()
print("first line : ", line1)
print("second line : ", line2)
f.close()
```

```
line1 = f.readline()
```

The above statement reads the first line of the file with which f is associated by the open statement. The function readline reads a line from the file including the newline character which exists at the end of the line in a file. Observe the difference with respect to input(). The function input does not read the newline.

```
print("first line : ", line1)
```

In the output, we will see a blank line as line1 as a new line at the end and print by default outputs a newline.

```
line2 = f.readline()
```

The readline also causes the cursor or offset – the position from the file where the next operation would happen – to change to the location past the end of the line in the file. This readline reads the next line in the file into the variable.

The presence of newline in each variable read from a file causes lots of problem in processing. We would prefer to get them removed. This is the way to remove any white space at either end of a string.

```
line1 = line1.strip();  line2 = line2.strip()
```

To read a file completely, we may want to use readline in a loop.
This code is taken from the file 2_file_read.py.

```
f = open("t.txt", "r")
line = f.readline()
while line:
	line = line.strip() # remember to assign back to line!!
	print(line)
	line = f.readline()
f.close()
```

Observe the presence of the code line = f.readline() before the loop as well as the last statement of the loop. We read a line before we enter the loop and in the loop, we process that line and then again read one more line at the end of the loop. As we keep reading from a file, we would reach the end of the file at some point. When we reach the end of the file, there is nothing more to read and readline returns an empty string. So, now you can understand the condition of the while. Would this code work if the input file is empty?

The code below taken from the file 2_file_read.py outputs the lines with line number. It is left to you to follow the program!

```
f = open("t.txt", "r")
line = f.readline()
i = 0
while line:
        line = line.strip() # remember to assign back to line!!
        i += 1
        print(i, ":", line)
        line = f.readline()
f.close()
```

There are a number of ways reading using a file object.
a) readline : reads a line
b) readlines : reads the whole file into a list – where each element of the list is a line of the file.
This code segment is from the file 3_file_read.py.

```
f = open("t.txt", "r")
lines = f.readlines()
f.close()

for line in lines:
        line = line.strip()
        print(line)
```

As reading the whole file is read into the memory, the file should not be very big.

The following code from the same file counts the number of occurrences of a particular word. We leave it to you to walk through the logic.

```
# let us count the number of troubles
f = open("t.txt", "r")
lines = f.readlines()
f.close()
total_count = 0
for line in lines:
	line = line.strip()
	total_count += line.count('trouble')
print("total trouble : ", total_count)
```

There are a number of ways reading using a file object.

a) readline : reads a line

b) readlines : reads the whole file into a list – where each element of the list is a line of the file

c) read : reads a number of bytes from the file. By default reads the whole file into a string.

This code segment is from the file 4_file_read.py.

```
f = open("t.txt", "r")
all = f.read()
print(all, type(all))
f.close()
```

As reading the whole file is read into the memory, the file should not be very big.

The following code from the same file finds the unique words in the file. We leave it to you to walk through the logic.

This is the data file used in the following programs.
butter.txt:

```
betty botter bought some butter
but the butter was bitter
betty botter bought some better butter
to make the bitter butter better
```

```
# display unique words in the file
f = open("butter.txt", "r")
all = f.read()
f.close()

wordset = set()
for word in all.split():
	wordset.add(word)

for word in sorted(wordset):
	print(word)
```

There are a number of ways reading using a file object.
a) readline : reads a line
b) readlines : reads the whole file into a list – where each element of the list is a line of the file
c) read : reads a number of bytes from the file. By default reads the whole file into a string.
d) use the file object as an iterable.
One of the most interesting feature of Python is the for loop. The for loop can step through any object which is iterable – which supports a function called __iter__ to create an iterator which in turn should support the function __next__ get the next element. The file object is iterable. Each time we iterate through the for loop, we get a line of the file.

This code segment is from the file 5_file_read.py.

```
f = open("butter.txt", "r")
for line in f :
    line = line.strip()
    print(line)
f.close()
```

When we reach the end of the file, the for loop terminates.

The following program code taken from the file 5_file_read.py displays the words in the decreasing order of frequency. We leave it to you to walk through the logic.

```
# create a frequency of each word in the file
# display in the decreasing order of frequency
f = open("butter.txt", "r")
info = { }
for line in f :
    line = line.strip()
    for word in line.split():
        if word not in info :
            info[word] = 0
        info[word] += 1
f.close()
for word in sorted(info, reverse = True, key = lambda k : info[k]):
    print(word, info[word])
```

Let us look at a small program to create a file.
This code is taken from the file 6_file_write.py

```
fout = open("out.txt", "w")
print(fout, type(fout))
print("hello world", file = fout)
fout.close()
```

Observe that the file is opened in write mode.
There are a couple of ways to write to a filename
a) print with the keyword parameter file having the file object as the argument
b) write on file object – is not discussed in this course.

This code is taken from the file 6_file_write.py.
This code creates a new file with the lines having the word trouble in the file t.txt. We leave it to you to walk through the logic.

```
# pick all lines containing trouble and write to another file
fin = open("t.txt")
fout = open("t_new.txt", "w")
for line in fin:
        if 'trouble' in line :
                line = line.strip()
                print(line, file = fout)
fin.close()
fout.close()
```

Thats all we shall discuss about files in this course.