CptS 111, Spring 2023
Lect. #18, Mar. 27, 2023
Class Notes

Today's Agenda:

1. Reading from files
2. Writing or printing to files

Ch. 7

Reading & Writing Files and Modules

1. Opening Files to Read, Reading Files, and Closing Files

We've often used the `input()` function to access data we want to use in our programs. However, this is very limiting. As we saw in PA #5, we can also read data from a file, and in both PA #4 and PA #5, we saw that we can print and write to files as well. In this lecture, we'll look more closely at the ways reading from files and writing or printing to files is done.

To read from a file, we must perform the following steps:

1. Open the file
2. Read the contents of the file
3. Close the file (optional for files that are read, but good practice)

A. Opening a File
The template for opening a file to read is:

```
<file_name> = open('<file>', 'r')
```

where:

- `<file_name>` is the name of the lvalue we assign to the file in our program
- `<file>` is the name of the file we want to read from the hard drive on our computer
- `'r'` is an optional argument indicating that we're going to read a file
- If `'r'` isn't included, Python assumes the file is to be read
- Quotation marks shown are necessary unless < `file` > is a variable name, i.e., we include the name of a file elsewhere (e.g., `name = 'faves.txt'`)

Aside: In order to open a file, we have to be in the folder (often called a directory) where it's located. To determine our current directory or to change directories, we import the `os` module. We'll discuss this more on Wednesday.

```
import os
os.getcwd()     # shows current folder (current working director
y)
```

In [1]:
```
# Let's see which directory I'm in

import os
os.getcwd()  # This directory is where I started running Jupyter
```

Out[1]:
```
'/Users/shira/teaching/cs111/spr23/lectures'
```

Let's continue with opening files. When we open a file, we don't see anything "happen."

In [3]:
```
# Open novel "Pride and Prejudice"; assign to file wrapper

file_in = open('p_and_p.txt', 'r')
```

So nothing seemed to happen, but we can check to see what type of variable `file_in` is using `type()`.

In [4]:
```
# Check type of 'file_in'

type(file_in)
```

Out[4]:
```
_io.TextIOWrapper
```

Hmm. Looks strange. What this means, basically, is that `file_in` is an interface for text data (IO stands for input/output) which is all the text characters in the novel "Pride and Prejudice." However, we can think of it as an input file to be read.

B. Reading Contents of Opened Files

After we've opened an input file, we can use one of three different built-in methods for reading the contents of the file:

1. `.read()` : reads the entire file as **a single string**
2. `.readlines()` : reads the entire file as **a list**; each list element is a single line from the file and a string
3. `.readline()` : reads a single line of the file as a string

In addition, we can use an input file as an iterable in a `for` -loop. Let's look at each of these.

In [5]:
```
# Close the file, then open it again.
# read() reads the entire file as a single string

file_in.close()
file_in = open('p_and_p.txt') # Best to use lvalue name with f or file in
novel = file_in.read()
file_in.tell()          # file_in is now "empty"
```

Out[5]:
```
702366
```

- The `.tell()` method is used to determine the location a pointer is pointing to within a file.
- After using the `.read()` method, the pointer points to the last position of the pointer within the file. Here it's 702366.
- There are more than 700,000 characters -- letters, not people :) -- in the novel "Pride and Prejudice." Note that this includes spaces, tabs, and newlines.
- Once we've read the entire contents of a file, we can't read it again unless we close it and then open it again.

Let's look at what sort of variable `novel` is.

In [6]:
```
# Check type of 'novel'

print('novel is a', type(novel))
```

```
novel is a <class 'str'>
```

We see that novel is, indeed, a single string.

C. Closing Files

To close a file, we simply use the `.close()` method. We have to close a file if we want to re-read it or else if it's a file to which we've written something. Note that **to open a file we use a function**, but **to close a file we use a method.**

In [7]:
```
# Close file using a method

file_in.close()
```

Now let's re-open the file.

In [8]:
```
# Open file; use name for input file

book = 'p_and_p.txt'
file_in = open(book) # book is a name -> no quotes needed
file_in.tell()
```

Out[8]:
```
0
```

Note that when we first open a file, the pointer points to the first character with an index of 0.

Next, let's read a single line from the novel; we do this using the `.readline()` method.

In [9]:
```
# .readline() reads a single line of a file as a string

line = file_in.readline()
print('line is a', type(line))
print(file_in.tell())
line
```

```
line is a <class 'str'>
21
```

Out[9]:
```
'Pride and Prejudice\n'
```

So we see that `.readline()`, indeed, reads in one line from the file as a string. Let's read in the rest of the file using `.readlines()`:

In [10]:
```
# Use .readlines() which will read in remainder of novel as list
# Each list element is a single line

lines = file_in.readlines()
print('lines are ', type(lines))
print('One line is:', lines[42])      # lines[42] is a string element
print('Another line is:', lines[188]) # lines[188] is a string element
```

```
lines are  <class 'list'>
One line is: "What is his name?"

Another line is: if you decline the office, I will take it on myself."
```

As advertised, `.readlines()` creates a list with each item in the list a line from a file. The blank lines are due to the newline character after each line in the novel. Let's see how many elements are in the list, i.e., how many lines are in the novel.

In [11]:
```
# See how many elements (lines) are in the list

len(lines)    # This includes blank lines because '\n' is a character!
```

Out[11]:
```
14218
```

Now let's close the novel file and open a shorter file.

In [12]:
```
# Close the file and then open a different one

file_in.close()
file_in = open(input('Enter a file name: '), 'r') # Careful with the pare
```

```
Enter a file name: jabber.txt
```

We can prompt a user for the name of a file using the `input()` function before opening the file! Very cool! So now let's see how we can use a a file as an iterable:

In [13]:
```
# Using opened file as iterable; very useful (so use it when you can!)

for line in file_in:
    print(line, end='')    # Use end='' because newlines already in file
```

```
'Twas brillig, and the slithy toves
      Did gyre and gimble in the wabe:
All mimsy were the borogoves,
      And the mome raths outgrabe.

"Beware the Jabberwock, my son!
      The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
      The frumious Bandersnatch!"

He took his vorpal sword in hand;
      Long time the manxome foe he sought--
So rested he by the Tumtum tree
      And stood awhile in thought.

And, as in uffish thought he stood,
      The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
      And burbled as it came!

One, two! One, two! And through and through
      The vorpal blade went snicker-snack!
He left it dead, and with its head
      He went galumphing back.

"And hast thou slain the Jabberwock?
      Come to my arms, my beamish boy!
O frabjous day! Callooh! Callay!"
      He chortled in his joy.

'Twas brillig, and the slithy toves
      Did gyre and gimble in the wabe:
All mimsy were the borogoves,
      And the mome raths outgrabe.
```

With each iteration of the `for` -loop, a single line from `file_in` was printed. Because the file includes newlines, we use `end=''` in the `print()` function so we don't have blank lines between the lines of the poem.

Let's close the file, open it again, and then use the various methods of reading.

In [14]:
```
# Close file and then re-open it; read just first line

file_in.close()
file_in = open(input('Enter a file name: ')) # Here 'r' is the default.
line = file_in.readline()                    # Read first line only
line
```

```
Enter a file name: jabber.txt
```

Out[14]:
```
"'Twas brillig, and the slithy toves\n"
```

`.readline()` reads a single line as a string. Note the newline character `\n` in this line.

In [15]:
```
# Close file and then re-open it; read entire file as string

file_in.close()
file_in = open('jabber.txt') # Here 'r' is the default.
poem = file_in.read()        # poem is single string
poem
```

Out[15]:
```
'\'Twas brillig, and the slithy toves\n      Did gyre and gimble in the
wabe:\nAll mimsy were the borogoves,\n      And the mome raths outgrab
e.\n\n"Beware the Jabberwock, my son!\n      The jaws that bite, the cl
aws that catch!\nBeware the Jubjub bird, and shun\n      The frumious B
andersnatch!"\n\nHe took his vorpal sword in hand;\n      Long time the
manxome foe he sought--\nSo rested he by the Tumtum tree\n      And sto
od awhile in thought.\n\nAnd, as in uffish thought he stood,\n      The
Jabberwock, with eyes of flame,\nCame whiffling through the tulgey woo
d,\n      And burbled as it came!\n\nOne, two! One, two! And through an
d through\n      The vorpal blade went snicker-snack!\nHe left it dead,
and with its head\n      He went galumphing back.\n\n"And hast thou sla
in the Jabberwock?\n      Come to my arms, my beamish boy!\nO frabjous
day! Callooh! Callay!"\n      He chortled in his joy.\n\n\'Twas brilli
g, and the slithy toves\n      Did gyre and gimble in the wabe:\nAll mi
msy were the borogoves,\n      And the mome raths outgrabe.\n'
```

Again, the `.read()` method reads in the entire file as a single string! Note use of the escape character `\'` before "Twas."

In [16]:
```
# Close file and then re-open it; read entire file as list using .readlin

file_in.close()
file_in = open('jabber.txt') # Here 'r' is the default.
poem = file_in.readlines()   # poem is a list
poem
```

Out[16]:
```
["'Twas brillig, and the slithy toves\n",
 '      Did gyre and gimble in the wabe:\n',
 'All mimsy were the borogoves,\n',
 '      And the mome raths outgrabe.\n',
 '\n',
 '"Beware the Jabberwock, my son!\n',
 '      The jaws that bite, the claws that catch!\n',
 'Beware the Jubjub bird, and shun\n',
 '      The frumious Bandersnatch!"\n',
 '\n',
 'He took his vorpal sword in hand;\n',
 '      Long time the manxome foe he sought--\n',
 'So rested he by the Tumtum tree\n',
 '      And stood awhile in thought.\n',
 '\n',
 'And, as in uffish thought he stood,\n',
 '      The Jabberwock, with eyes of flame,\n',
 'Came whiffling through the tulgey wood,\n',
 '      And burbled as it came!\n',
 '\n',
 'One, two! One, two! And through and through\n',
 '      The vorpal blade went snicker-snack!\n',
 'He left it dead, and with its head\n',
 '      He went galumphing back.\n',
 '\n',
 '"And hast thou slain the Jabberwock?\n',
 '      Come to my arms, my beamish boy!\n',
 'O frabjous day! Callooh! Callay!"\n',
 '      He chortled in his joy.\n',
 '\n',
 "'Twas brillig, and the slithy toves\n",
 '      Did gyre and gimble in the wabe:\n',
 'All mimsy were the borogoves,\n',
 '      And the mome raths outgrabe.\n']
```

Again, we see that the `.readlines()` method reads in the entire file as a list, with each line a single string element in the list.

In [17]:
```
# First element == first line

poem[0]
```

Out[17]:
```
"'Twas brillig, and the slithy toves\n"
```

In [18]:
```
# Last element == last line

poem[-1]
```

Out[18]:
```
'      And the mome raths outgrabe.\n'
```

In [19]:
```
# And as a good practice, close the file

file_in.close()
```

2. Opening Files for Writing/Printing, Writing/Printing to Files, and Closing Files

The template for opening a file to write to is:

```
<file_name> = open('<file>', 'w')
```

where:

- `<file_name>` is the name of the lvalue; again use f or file in the name
- `<file>` is the name of the file we want to write to
- `'w'` is needed and will create a new file, but ***it will delete a file with the same name***
- `a` is safer because it allows you to append to an existing file or create a new file if there is no existing file; it doesn't allow you to read from a file
- `'a+'` allows you to read from a file or write (append) to a file
- Quotation marks shown are necessary unless < `file` > is a variable name
- Output files, i.e., files written to, must be closed to flush buffered text to the file; best to close all files

Let's try a few examples.

In [20]:
```
# Open file called junk.txt, write to it, then close it

file_out = open('junk.txt', 'w')  # Open output file to write
file_out.write('Hello, World!')   # Write to output file
file_out.close()                  # Close output file
```

The `.write()` method writes a single string, ***and only a single string***, to a file rather than to stdout. That's why we don't see the result. To see the result, we have to open the file we created, and read in what we wrote.

In [21]:
```
# Open file again, read it, then close it

file_in = open('junk.txt')        # Open same file as input file to read
junk = file_in.read()             # Read input file as string
file_in.close()                   # Close input file
print(junk)                       # See what we have
```

```
Hello, World!
```

This is what `junk.txt` contains! Next, let's write something else to this file.

In [22]:
```
# Open file to append text, close it, open it, read it, and close it

file_out = open('junk.txt', 'a')  # Open same file to append to
file_out.write('Go, Cougs!')      # Write to output file
file_out.close()                  # Close output file

file_in = open('junk.txt')        # Open same file as input file to read
junk = file_in.readlines()        # Read input file as list
file_in.close()                   # Close input file
print(junk)                       # See what we have
```

```
['Hello, World!Go, Cougs!']
```

Notice that when we use the append (`'a'`) option, the text continues exactly from the point where we stopped previously. ***No space is added!***

In [23]:
```
# Let's try this again

file_out = open('junk.txt', 'a')  # Open it again to append to
file_out.write('\nOuch!')         # Write to output file
file_out.close()                  # Close output file

file_in = open('junk.txt')        # Open input file
junk = file_in.read()             # Read input file as string
print(junk)                       # See what we have
file_in.close()                   # Close input file
```

```
Hello, World!Go, Cougs!
Ouch!
```

So we can add newlines or spaces when we append something to an existing file. **When using the append option, it's a good idea to start with a newline or even several of them.**

Finally, I actually prefer to use the `print()` function to write output to a file because it allows us to use string formatting, it's really easy to use (just just a file name), and it's much more flexible, i.e., you're not limited to just a single string. You can use it just as if you were writing to stdout.

In [24]:
```
# I prefer to use print() to enter output to a file; default for print
# is to write to stdout, but can change using kwarg 'file'

file_out = open('junk', 'w')
cat = 'dog'
print(f'I love my cat named{cat:->5}!', file=file_out) # Print to a file!
file_out.close()

file_in = open('junk')
string = file_in.read()
print(string)
```

```
I love my cat named--dog!
```