In-class worksheet 18

Mar 28, 2019

Reading and writing files in Python

We interact with files in python by opening the file and assigning it to a variable. This variable is called a *file handle*. The file handle is an object, and it has member fuctions such as `.read()` and `.write()` that allow us to either read the file contents or write to the file. After we are done working with the file, we must close the file handle.

We open a file and generate the associated handle with the `open()` function. This function takes two arguments:

1. the *name* of the file to open
2. the *mode* in which the file should be read. Modes, with usage examples, include,

- read-only (`"r"`). Use this mode to read a file, but not write anything to it.
  - Usage: `handle = open("filename.txt", "r")`
- write-only (`"w"`). Use this mode to write to an empty file.
  - Usage: `handle = open("filename.txt", "w")`
- append (`"a"`). Use this mode to add content to an existing file.
  - Usage: `handle = open("filename.txt", "a")`

Opening files for reading

We will first work with a simple test file, called `"testfile.txt"`. The file has the following contents:

> This is line one of the file.
> This is line two of the file.
> This is line three of the file.

(You can download this file from the class website here: http://wilkelab.org/classes/SDS348/data_sets/testfile.txt (http://wilkelab.org/classes/SDS348/data_sets/testfile.txt))

We will open this file and read it in one go:

In [1]:
```
# The name of the file we want to open
filename = "testfile.txt"

file_handle = open(filename, "r") # Open file as read-only
file_contents = file_handle.read() # Read contents of file
file_handle.close() # Close file when finished (important!)

# Print out the contents of the file
print(file_contents)
```

```
This is line one of the file.
This is line two of the file.
This is line three of the file.
```

Note that the `.read()` method saves the *entire* body of the file to a single string. Another convenient way to read a file is to retrieve it as a list of lines, so that we can easily loop over the file contents. We can do this with the `.readlines()` method:

In [2]:
```
file_handle = open(filename, "r") # Open file as read-only
file_lines = file_handle.readlines() # Read contents of file
file_handle.close() # Close file

print(file_lines)
```

```
['This is line one of the file.\n', 'This is line two of the file.\n', 'This is
line three of the file.\n']
```

We see that `.readlines()` gives us a list with three entries, where each entry is one line in the file. We can also see that all entries of this list end with the symbol `\n`. This symbols represents the new-line character. It determines when one line ends and the next one begins.

Now that we have the file lines in a list, we can easily loop over them, and perform some calculations as needed:

In [3]:
```
# Loop over each line in file, and use a counter to keep track of lines
counter = 1
for line in file_lines:
    print(counter, line)
    counter += 1
```

```
1 This is line one of the file.

2 This is line two of the file.

3 This is line three of the file.
```

You may notice that there is an empty line between each line of output. Can you guess why? See below in Problem 4 for an answer.

Opening files for writing

Opening a file in write-mode will overwrite the file, but opening in append-mode will add to the bottom of an existing file. Note that if we open a file for writing (or appending) that doesn't already exist, then a new file will be created with the specified name. By contrast, if we attempt to open a non-existing file for reading, we will receive an error message.

To write to a file, we use the `.write()` method on the file handle:

In [4]:
```
# Define the name of the file we'll be working with
filename = "testfile2.txt"

# It is often good to name the file handles in
# such a way that we know whether we are reading from
# or writing to the file. Here we open the file
# for writing, so we call it `outfile`:
outfile = open(filename, "w")

# The write function doesn't add a newline automaticall,
# so we do this explicitly:
outfile.write("I'm writing a sentence to this file.\n")
outfile.close()

# Verify that we wrote to the file correctly,
# by opening and reading it
infile = open(filename, "r") #note the "r" mode
contents = infile.read()
infile.close()
print(contents)
```

```
I'm writing a sentence to this file.
```

Note that the above code created a new file and wrote a single sentence to it. No matter how many times you execute this code, the file will have the same contents.

To add new contents to an existing file, open the file in append ("a") mode:

In [5]:
```
# Open a file for appending, and append text to it
for i in range(5):   # do this five times
    outfile = open(filename, "a")
    outfile.write("I'm writing another line to this existing file.\n")
    outfile.close()

# Verify that we wrote to the file correctly, by opening and reading it
infile = open(filename, "r") #note the "r" mode
contents = infile.read()
infile.close()
print(contents)
```

```
I'm writing a sentence to this file.
I'm writing another line to this existing file.
I'm writing another line to this existing file.
I'm writing another line to this existing file.
I'm writing another line to this existing file.
I'm writing another line to this existing file.
```

The `with` statement

As you can see from the above code examples, when we are dealing with files we need to write many blocks of code of the form `.open()`, code to interact with file, `.close()`. This can become cumbersome, and in particular we may forget to close some of the files that we opened. Not closing files can cause all sorts of trouble. For example, other applications may not be able to interact with a file until your program has properly closed it. Or, if you write a loop that opens many files but never closes them, you may crash your computer. Thus, since closing files is so critical, wouldn't it be nice if Python did this for us automatically? It will do so, if we're using the `with` statement.

In the `with` statment, instead of writing

```
file_handle = open(filename, mode)
```

we write

```
with open(filename, mode) as file_handle:
    ... # code block that operates on the file handle
```

The file will be closed automatically once we leave the block.

Thus, we could rewrite the last two examples with `with` in the following form:

In [6]:
```
filename = "testfile2.txt"
with open(filename, "w") as outfile:  # open file for writing
    outfile.write("I'm writing a sentence to this file.\n")

for i in range(5):   # do this five times
    with open(filename, "a") as appendfile:  # open file for appending
        appendfile.write("I'm writing another line to this existing file.\n")

# Verify that we wrote to the file correctly, by opening and reading it
with open(filename, "r") as infile:  # open file for reading
    contents = infile.read()
print(contents)
```

```
I'm writing a sentence to this file.
I'm writing another line to this existing file.
I'm writing another line to this existing file.
I'm writing another line to this existing file.
I'm writing another line to this existing file.
I'm writing another line to this existing file.
```

Problems

Problem 1:

Download the file "`road_not_taken.txt`" from the class website: http://wilkelab.org/classes/SDS348/data_sets/road_not_taken.txt (http://wilkelab.org/classes/SDS348/data_sets/road_not_taken.txt)
(You may have to right-click the link and choose "save as". Make sure to save the file in the same location where your Jupyter notebook is.) This file contains the famous poem "The Road not Taken" by Robert Frost.

(a) Write a program that reads the file in one go and prints out the file contents.

(b) Write a program that reads in the file line by line and counts the total number of lines.

(c) Write a program that counts the number of letters in the file. Use the function `count_letters()` we have discussed in a previous class. Then print out how offen the different vowels (a, e, i, o, u) are used in this document.

In [7]:
```
# Problem 1a:

# Open the file for reading.
infile = open("road_not_taken.txt", "r")
contents = infile.read()  # read the whole file at once
infile.close()  # close file when we're done
print(contents)
```

```
        THE ROAD NOT TAKEN
           Robert Frost

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,
And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.
I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I-
I took the one less traveled by,
And that has made all the difference.
```

In [8]:
```
# Problem 1b:

# Open the file for reading.
infile = open("road_not_taken.txt", "r")
lines = infile.readlines()
infile.close()
print('The file "road_not_taken.txt" contains', len(lines), 'lines.' )
```

```
The file "road_not_taken.txt" contains 23 lines.
```

In [9]:
```
# Problem 1c:

# count_letters function
def count_letters(s):
    counts = {} # empty dict
    for c in s:
        if c in counts: # does letter exist in dict?
            counts[c]+=1 # yes, increase count by 1
        else:
            counts[c]=1 # no, set count to 1
    return counts # return result

# Open the file for reading and count the letters
with open("road_not_taken.txt", "r") as infile:
    counts = count_letters(infile.read())

print('Vowel usage in the poem "The Road not Taken":')
for c in ['a', 'e', 'i', 'o', 'u']:
    print(c, ":", counts[c])
```

```
Vowel usage in the poem "The Road not Taken":
a : 51
e : 69
i : 20
o : 50
u : 10
```

Problem 2:

Read in the file `"road_not_taken.txt"`, loop over every line in the file, identify the lines that contain the string `"road"` (ignoring case), and write those lines into a new file called `"extracted_lines.txt"`. Then, read the file `"extracted_lines.txt"` back in and print its contents, to verify that everything worked right.

In [10]:
```
# create file with extracted lines
infile = open("road_not_taken.txt", "r")
outfile = open("extracted_lines.txt", "w")
for line in infile:  # we can iterate over the file directly
    if "road" in line.lower(): # ignore case by converting to lower case
        outfile.write(line)
infile.close()
outfile.close()

# read the generated file back in and print out its contents
infile = open("extracted_lines.txt", "r")
print(infile.read())
infile.close()
```

```
        THE ROAD NOT TAKEN
Two roads diverged in a yellow wood,
Two roads diverged in a wood, and I-
```

Problem 3:

Take the solution to one of your previous problems and rewrite them using a `with` statement. (Skip this problem if you have used `with` statements throughout.)

In [11]:
```
# Problem 1b rewritten using `with` statement:

with open("road_not_taken.txt", "r") as infile:
    lines = infile.readlines()
print('The file "road_not_taken.txt" contains', len(lines), 'lines.' )
```

```
The file "road_not_taken.txt" contains 23 lines.
```

If this was easy

Problem 4:

Using **one** `with` statement and **one** `for` loop, open the file "`road_not_taken.txt`" as input file, convert it to upper case, and write it to a new file called "`road_not_taken_upper.txt`". Then read the newly generated file back in and print out the first 10 lines (+ line numbers), to make sure everything looks right. Make sure you don't get any empty lines between the individual lines you print.

In [12]:
```
with open("road_not_taken.txt", "r") as infile, \
     open("road_not_taken_upper.txt", "w") as outfile:
    for line in infile:
        outfile.write(line.upper())

with open("road_not_taken_upper.txt", "r") as infile:
    i = 0
    for line in infile:
        # since each line ends in a newline, and the print() function
        # also adds a newline, we have to use the .rstrip() function
        # to remove the newline character at the end of each line.
        print(i+1, line.rstrip())
        i += 1
        if i>=10:
            break
```

```
1          THE ROAD NOT TAKEN
2             ROBERT FROST
3
4 TWO ROADS DIVERGED IN A YELLOW WOOD,
5 AND SORRY I COULD NOT TRAVEL BOTH
6 AND BE ONE TRAVELER, LONG I STOOD
7 AND LOOKED DOWN ONE AS FAR AS I COULD
8 TO WHERE IT BENT IN THE UNDERGROWTH;
9 THEN TOOK THE OTHER, AS JUST AS FAIR,
10 AND HAVING PERHAPS THE BETTER CLAIM,
```

Problem 5:

Instead of reading the file "`road_not_taken.txt`" from your own computer, read it directly from the web. Hint: Use the function `urlopen()` from the `urllib` package. And pay attention to the type of data that you receive from the handle created by `urlopen()`.

In [13]:
```
from urllib.request import urlopen

file_URL = "http://wilkelab.org/classes/SDS348/data_sets/road_not_taken.txt"

with urlopen(file_URL) as infile:
    i = 0
    for line in infile:
        # urllib returns encoded strings (byte objects), and
        # we need to use the `decode()` function to turn them
        # into strings we can print
        print(i+1, line.decode().rstrip())
        i += 1
        if i>=10:
            break
```

```
1         THE ROAD NOT TAKEN
2            Robert Frost
3
4 Two roads diverged in a yellow wood,
5 And sorry I could not travel both
6 And be one traveler, long I stood
7 And looked down one as far as I could
8 To where it bent in the undergrowth;
9 Then took the other, as just as fair,
10 And having perhaps the better claim,
```