Python 6

I/O and Files

I/O stands for input/output. The in and out refer to getting data into and out of your script. It might be a little surprising at first, but writing to the screen, reading from the keyboard, reading from a file, and writing to a file are all examples of I/O.

Writing to the Screen

You should be well versed in writing to the screen. We have been using the `print()` function to do this.

```
>>> print ("Hello, PFB!")
Hello, PFB!
```

> Remember this example from one of our first lessons?

Reading input from the keyboard

This is something new. There is a function which prints a message to the screen and waits for input from the keyboard. This input can be stored in a variable. It always starts as a string. Convert to an int or float if you want a number. When you are done entering text, press the enter key to end the input. A newline character is not included in the input.

```
>>> user_input = input("Type Something Now: ")
Type Something Now: Hi
>>> print(user_input)
Hi
>>> in_str = input("Enter a number: ")
>>> type(in_str)
<class 'str'>
>>> num = int(in_str)
>>> num
445
```

Reading from a File

Mostly you will read data from files.

The first thing to do with a file is open it. We can do this with the `open()` function. The `open()` function takes the file name and access mode as arguments and returns a file object.

The most common access modes are read (r) and write (w).

Open a File

```
>>> seq_file_obj = open("seq.nt.txt","r")
```

> `seq_file_obj` is a name of a variable. This can be anything, but make it a helpful name that describes what kind of file you are opening and to distinguish it from the filename.

Reading the contents of a file

Now that we have opened a file and created a file object we can do things with it, like read from the file. Let's read all the contents at once.

Before we do that, let's go to the command line and `cat` the contents of the file to see what's in it first.

```
$ cat seq.nt.txt
ACAAAATACGTTTTGTAAATGTTGTGCTGTTAACACTGCAAATAAACTTGGTAGCAAACACTTCCAAAAG
ACCGGTTTCCAAAGACAGTCTTCTAATTCCTCATTAGTAATAAGTAAAATGTTTATTGTTGTAGCTCTGG
$
```

Note the new lines. Now, lets print the contents to the screen with Python. We will use `read()` to read the entire contents of the file into a variable.

```
>>> seq_file_obj = open("seq.nt.txt","r")
>>> contents = seq_file_obj.read()
>>> print(contents)  # note newline characters are part of the file!
ACAAAATACGTTTTGTAAATGTTGTGCTGTTAACACTGCAAATAAACTTGGTAGCAAACACTTCCAAAAG
ACCGGTTTCCAAAGACAGTCTTCTAATTCCTCATTAGTAATAAGTAAAATGTTTATTGTTGTAGCTCTGG

>>> seq_file_obj.close()
```

> The complete contents can be retrieved with the `read()` method. Notice the newlines are maintained when `contents` is printed to the screen. `print()` adds another new line when it is finished printing. It is good practice to close your file. Use the `close()` method.

Here's another way to read data in from a file. A `for` loop can be used to iterate through the file one line at a time.

```
#!/usr/bin/env python3

seq_file_obj = open("seq.nt.txt","r")
for line in seq_file_obj: # Python magic: reads in a line from file
  print(line)
```

Output:

```
$ python3 file_for.py
ACAAAATACGTTTTGTAAATGTTGTGCTGTTAACACTGCAAATAAACTTGGTAGCAAACACTTCCAAAAG

ACCGGTTTCCAAAGACAGTCTTCTAATTCCTCATTAGTAATAAGTAAAATGTTTATTGTTGTAGCTCTGG
```

> Notice the blank line at after each line we print. `print()` adds a newline and we have a newline at the end of each line in our file. Use `rstrip()` method to remove the newline from each line.

Let's use `rstrip()` method to remove the newline from our file input.

```
$ cat file_for_rstrip.py
#!/usr/bin/env python3

seq_file_obj = open("seq.nt.txt","r")
for line in seq_file_obj:
  line = line.rstrip()
  print(line)
```

> `rstrip()` without any parameters returns a string with whitespace removed from the end.

Output:

```
$ python3 file_for_rstrip.py
ACAAAATACGTTTTGTAAATGTTGTGCTGTTAACACTGCAAATAAACTTGGTAGCAAACACTTCCAAAAG
ACCGGTTTCCAAAGACAGTCTTCTAATTCCTCATTAGTAATAAGTAAAATGTTTATTGTTGTAGCTCTGG
```

> Where do the newlines in the above output come from?

Opening a file with `with open() as fh:`

Many people add this, because it closes the file for you automatically. Good programming practice. Your code will clean up as it runs. For more advanced coding, `with ... as ...` saves limited resources like filehandles and database connections. For now, we just need to know that the `with ... as ...:` does the same as `fh = open(...) ... fh.close()`. So here's what the adapted code looks like

```
#!/usr/bin/env python3

with open("seq.nt.txt","r") as seq_file_obj: #cleans up after exiting
                                             # the 'with' block
  for line in seq_file_obj:
    line = line.rstrip()
    print(line)
#file gets closed for you here.
```

Writing to a File

Writing to a file just required opening a file for writing then using the `write()` method.

The `write()` method is like the `print()` function. The biggest difference is that it writes to your file object instead of the screen. Unlike `print()`, it does not add a newline by default. `write()` takes a single string argument.

Let's write a few lines to a file named "writing.txt".

```
#!/usr/bin/env python3

fo = open("writing.txt" , "w")  # note that we are writing so the mode is "w"
fo.write("One line.\n")
fo.write("2nd line.\n")
fo.write("3rd line" + " has extra text\n")
some_var = 5
fo.write("4th line has " + str(some_var) + " words\n")  # the write() method does not
convert ints for you
fo.close()
print("Wrote to file 'writing.txt'") # it's nice to tell the user you wrote a file
```

Output:

```
$ python3 file_write.py
Wrote to file 'writing.txt'
$ cat writing.txt
One line.
2nd line.
3rd line has extra text
4th line has 5 words
```

Now, let's get crazy! Lets read from one file a line at a time. Do something to each line and write the results to a new file.

```
#!/usr/bin/env python3

total_nts = 0
# open two file objects, one for reading, one for writing
with open("seq.nt.txt","r") as seq_read, open("nt.counts.txt","w") as seq_write:
  for line in seq_read:
    line = line.rstrip()
    nt_count = len(line)
    total_nts += nt_count
    seq_write.write(f"{nt_count}\n")

  seq_write.write(f"Total: {total_nts}\n") # better than concatenation + str()
```

```
print("Wrote 'nt.counts.txt'")
```

Output:

```
$ python3 file_read_write.py
Wrote 'nt.counts.txt'
$ cat nt.counts.txt
71
71
Total: 142
```

> The file we are reading from is named, "seq.nt.txt"
> The file we are writing to is named, "nt.counts.txt"
> We read each line, calculate the length of each line, and print the length
> We also create a variable to keep track of the total nt count
> At the end, we print out the total count of nts
> Finally, we close each of the files

Building a Dictionary from a File

This is a very common task. It will use a loop, file I/O, and a dictionary.

Assume we have a file called "sequence_data.txt" that contains tab-delimited gene names and sequences that looks something like this

```
TP53    GATGGGATTGGGGTTTTCCCCTCCCATGTGCTCAAGACTGGCGCTAAAAGTTTTGAGCTTCTCAAAAGTC
BRCA1   GTACCTTGATTTCGTATTCTGAGAGGCTGCTGCTTAGCGGTAGCCCCTTGGTTTCCGTGGCAACGGAAAA
```

How can we read this whole file in to a dictionary?

```
#!/usr/bin/env python3

genes = {}
with open("sequence_data.txt","r") as seq_read:
  for line in seq_read:
    line = line.rstrip()
    gene_id,seq = line.split() #split on whitespace
    genes[gene_id] = seq
print(genes)
```

Output:

```
{'TP53': 'GATGGGATTGGGGTTTTCCCCTCCCATGTGCTCAAGACTGGCGCTAAAAGTTTTGAGCTTCTCAAAAGTC',
'BRCA1': 'GTACCTTGATTTCGTATTCTGAGAGGCTGCTGCTTAGCGGTAGCCCCTTGGTTTCCGTGGCAACGGAAAA'}
```