Welcome!

- Sign in to your computer and open IDLE.
- Download the workshop materials:
  - **https://tinyurl.com/2018-bcb-adv-python**
- Ask a volunteer for help if you need it.

BCBGSO ADVANCED PYTHON WORKSHOP 2018

Welcome to the Advanced Python Workshop

- Organized by the Bioinformatics and Computational Biology Graduate Student Organization.
- The BCB Symposium is next week. More info here: http://www.bcb.iastate.edu/bcb-symposium-march-30-2018
- This year's theme is "The Past, Present, and Future of Bioinformatics and Computational Biology."
- Please register if you're interested!

Acknowledgements

- Thanks to Urminder for organizing these workshops.
- Trish Stauble for helping us with logistics.
- Thanks to Levi Baber and Jennifer Johnson from IT for helping us set up the virtual machines.
- BIG UPS to our volunteers: Sagnik, Akshay, Pranav, Sharmistha, and Avani.

Today’s Plan

- Review Lists
- Review Reading files
- Review Functions
- Designing Basic Programs

Lists

Lists (and dictionaries) are probably the most used data structure in Python.

They are mutable, easy to work with, and pretty efficient.

They can hold any combination of data types, unlike, say, Java.

We will be working a lot with lists today, so it’s important for you to be comfortable with them.

Some common list operations:

```
names = [‘Alice’, ‘Adrianna’, ‘Gia’]
names[-1]  # ‘Gia’
names[:2]  # [‘Alice’, ‘Adrianna’]
names2 = [‘Juliet’, ‘Rami’]
names.extend(names2)
len(names) # 5
names.remove(“Gia”)
len(names) # 4
```

A powerful aspect of lists is list comprehensions. There is a decent outline of them here.

List comprehensions are expressions that create lists. It is a lot more useful than it sounds.

For example, instead of doing:

```
num_list = [1, 2, 3, . . ., 100]
```

We can type:

```
num_list = [i + 1 for i in range(100)]
```

Open `list_review.py` and work through the examples.

Reading Files

It is very easy to read from and write to files in Python.

In most cases, code involving files will be built off of the following:

```
with open(file_name) as fin:
    for line in fin:
        < do stuff >
```

This opens up the file, reads it line by line, does whatever to each of the lines, and closes it when you're finished.

Open up `file_io.py` and work through it.

Functions

Functions are modules of code that take in arguments and do something with them.

For instance, the `len()` function takes in an object and returns it's length.

The `str.replace(target, replace)` is a function performed on a string that replaces all instances of `target` with `replace`.

The syntax to define a function is:

```
def function_name(input_arguments):
    < do stuff >
```

Why use functions?

- Portability – by writing your code as a function, you can export it to your other programs or share it with others
- Readability – it's easier to understand what a function is doing by it's name than by looking at it's code (assuming it's named well)
- Debugging code – if you don't use functions and instead just copy-paste your code everywhere, you have to make corrections to each of those code chunks each time you want to change something. If you have a function, you only need to make one change.

Open up `functions.py` and work through the examples.

Designing Basic Programs

Exercise 1

We will now walk through the process of designing a simple program.

Go to the `exercise_1` folder.

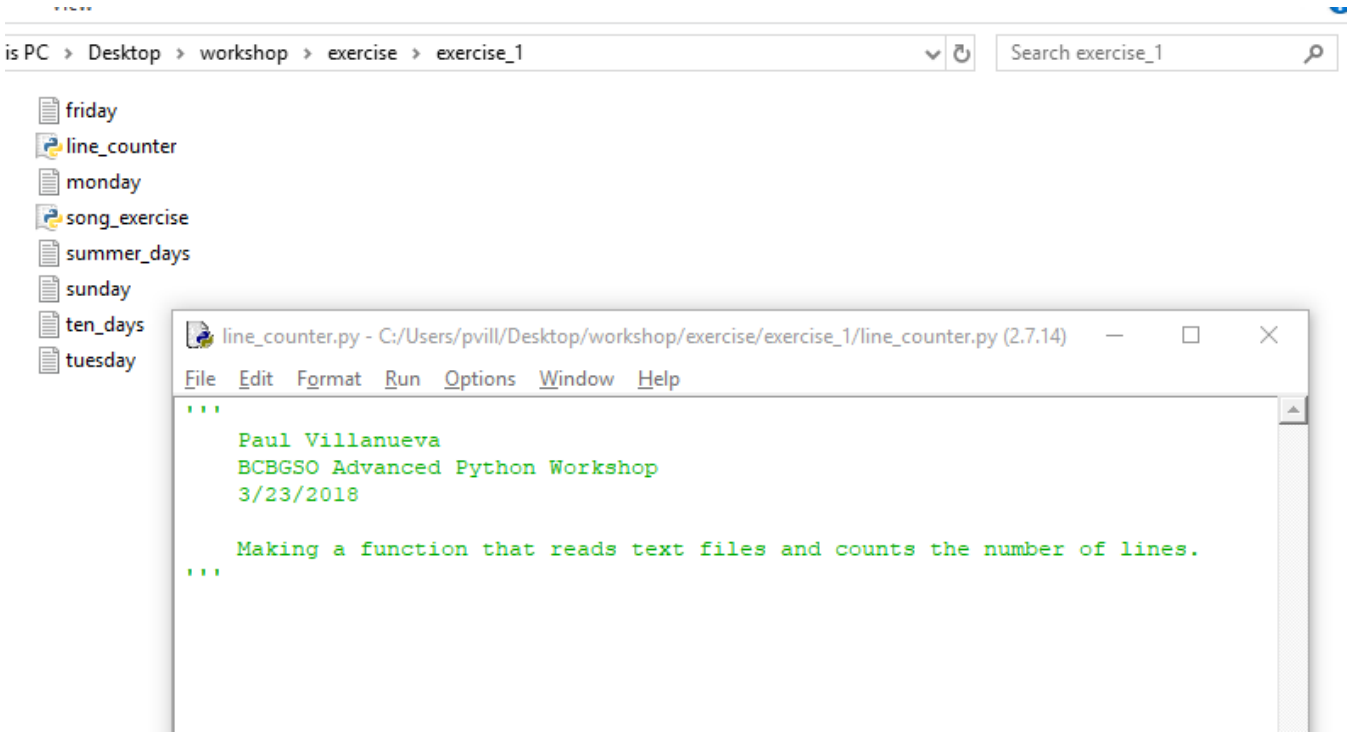
The folder contains several files containing song lyrics.

We want to find the song with the most lines, as well as the song title and artist.

In IDLE, do File > New File.

Add a header containing your name and today's date.

Save it in the `exercise_1` folder as `line_counter`.



Begin by writing code to open `sunday.txt` and print out each line.

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/line_counter.py (2.7.14)

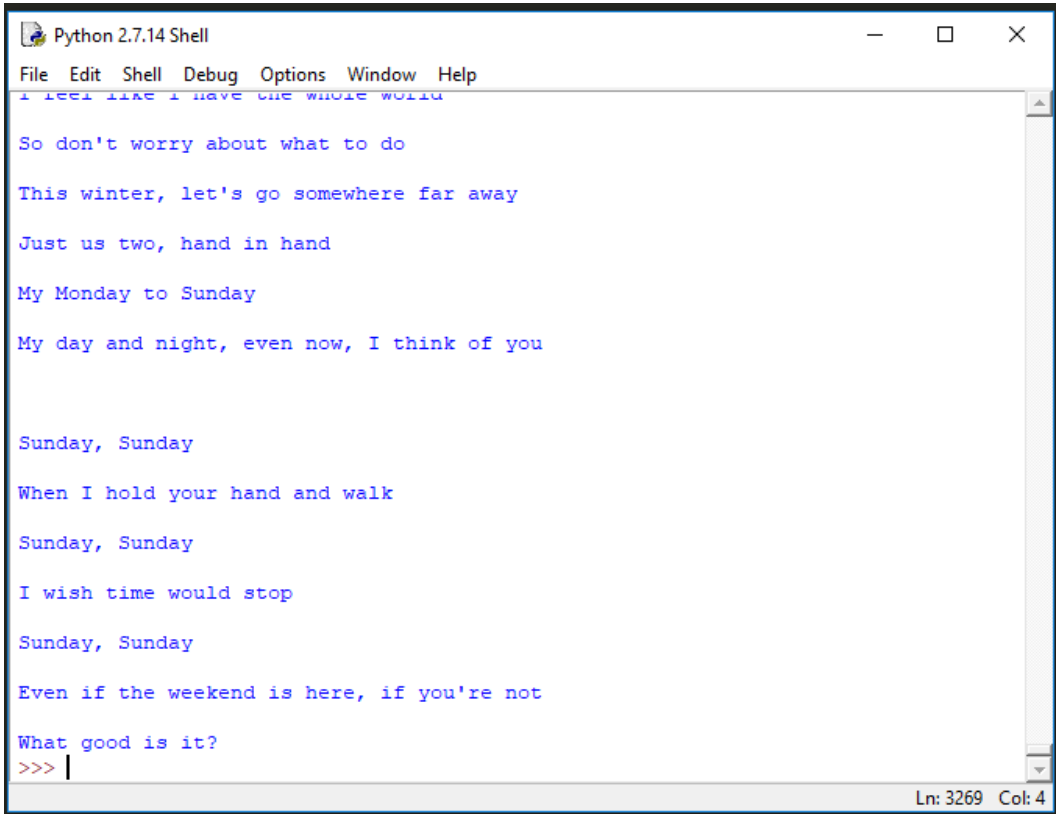
File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and counts the number of lines.
'''

with open('sunday.txt') as fin:
    for line in fin:
        print line
```

Run the code. What happened?



Why are extra lines being printed out?

Modify your code to remove trailing newline characters from each line.

(Hint)

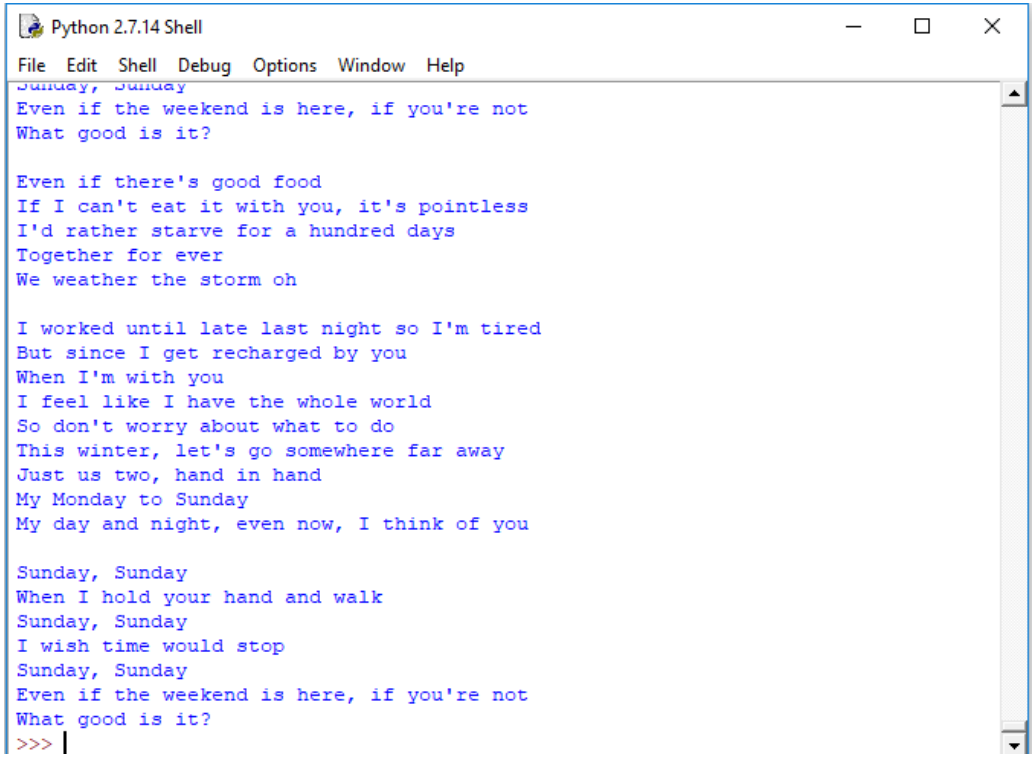
line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/line_counter.py (2.7.14)

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and counts the number of lines.
'''

with open('sunday.txt') as fin:
    for line in fin:
        line = line.rstrip()
        print line
```



N I C E

Now, let’s add some code to start counting lines.

Hint: there are two ways to do this listed here.

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/l

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and co
'''

with open('sunday.txt') as fin:
    line_num = 0
    for line_num, line in enumerate(fin):
        line = line.rstrip()
        line_num += 1
        print line
```

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and cc
'''

with open('sunday.txt') as fin:
    for line_num, line in enumerate(fin):
        line = line.rstrip()
        print line
```

Both work equally well. I will use the method on the right that uses `enumerate()`.

See documentation here to learn more about `enumerate()`.

Let’s output the line number with the lines.

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and
'''

with open('sunday.txt') as fin:
    for line_num, line in enumerate(fin):
        line = line.rstrip()
        print line_num, line
```

Python 2.7.14 Shell

File Edit Shell Debug Options Window Help

```
48 Sunday, Sunday
49 Even if the weekend is here, if you're not
50 What good is it?
51
52 Even if there's good food
53 If I can't eat it with you, it's pointless
54 I'd rather starve for a hundred days
55 Together for ever
56 We weather the storm oh
57
58 I worked until late last night so I'm tired
59 But since I get recharged by you
60 When I'm with you
61 I feel like I have the whole world
62 So don't worry about what to do
63 This winter, let's go somewhere far away
64 Just us two, hand in hand
65 My Monday to Sunday
66 My day and night, even now, I think of you
67
68 Sunday, Sunday
69 When I hold your hand and walk
70 Sunday, Sunday
71 I wish time would stop
72 Sunday, Sunday
73 Even if the weekend is here, if you're not
74 What good is it?
>>>
```

Ln: 3577 Col: 4

Output the total number of lines at the end.

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exe

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files a
'''

with open('sunday.txt') as fin:
    for line_num, line in enumerate(fin):
        line = line.rstrip()
        print line_num, line

print "Total number of lines: ", line_num
```

```
62 So don't worry about what to do
63 This winter, let's go somewhere far away
64 Just us two, hand in hand
65 My Monday to Sunday
66 My day and night, even now, I think of you
67
68 Sunday, Sunday
69 When I hold your hand and walk
70 Sunday, Sunday
71 I wish time would stop
72 Sunday, Sunday
73 Even if the weekend is here, if you're not
74 What good is it?
Total number of lines:  74
>>>
```

Let’s turn this into a function now.

Using the code you just wrote, define a new function `line_counter()` that takes as input the name of a file and outputs the total number of lines.

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/line_counter.py (2.7.14)

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and counts the number of lines.
'''

def line_counter(in_file):
    with open('sunday.txt') as fin:
        for line_num, line in enumerate(fin):
            line = line.rstrip()
            print line_num, line

    print "Total number of lines: ", line_num
```

What happens when you run this code? Why?

Nothing happens because this file no longer contains any commands. It only contains a function definition.

To use the function, we have to call it:

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercis

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and
'''

def line_counter(in_file):
    with open('sunday.txt') as fin:
        for line_num, line in enumerate(fin):
            line = line.rstrip()
            print line_num, line

    print "Total number of lines: ", line_num


line_counter('sunday.txt')
```

Python 2.7.14 Shell

File Edit Shell Debug Options Window Help

```
50 What good is it?
51
52 Even if there's good food
53 If I can't eat it with you, it's pointless
54 I'd rather starve for a hundred days
55 Together for ever
56 We weather the storm oh
57
58 I worked until late last night so I'm tired
59 But since I get recharged by you
60 When I'm with you
61 I feel like I have the whole world
62 So don't worry about what to do
63 This winter, let's go somewhere far away
64 Just us two, hand in hand
65 My Monday to Sunday
66 My day and night, even now, I think of you
67
68 Sunday, Sunday
69 When I hold your hand and walk
70 Sunday, Sunday
71 I wish time would stop
72 Sunday, Sunday
73 Even if the weekend is here, if you're not
74 What good is it?
Total number of lines:  74
>>>
```

And we're now getting the same output as before.

Try it on some of the other files by changing the file name in the `line_counter()` call.

What did you expect? What did you get? What's wrong?

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and
'''

def line_counter(in_file):
    with open('sunday.txt') as fin:
        for line_num, line in enumerate(fin):
            line = line.rstrip()
            print line_num, line

    print "Total number of lines: ", line_num


line_counter('monday.txt')
```

Python 2.7.14 Shell

File Edit Shell Debug Options Window Help

```
49 Even if the weekend is here, if you're not
50 What good is it?
51
52 Even if there's good food
53 If I can't eat it with you, it's pointless
54 I'd rather starve for a hundred days
55 Together for ever
56 We weather the storm oh
57
58 I worked until late last night so I'm tired
59 But since I get recharged by you
60 When I'm with you
61 I feel like I have the whole world
62 So don't worry about what to do
63 This winter, let's go somewhere far away
64 Just us two, hand in hand
65 My Monday to Sunday
66 My day and night, even now, I think of you
67
68 Sunday, Sunday
69 When I hold your hand and walk
70 Sunday, Sunday
71 I wish time would stop
72 Sunday, Sunday
73 Even if the weekend is here, if you're not
74 What good is it?
Total number of lines:  74
>>>
```

The problem is that we never changed the `sunday.txt` file name to the input variable.

Let's change it now and run it again.

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and c
'''

def line_counter(in_file):
    with open(in_file) as fin:
        for line_num, line in enumerate(fin):
            line = line.rstrip()
            print line_num, line

    print "Total number of lines: ", line_num


line_counter('monday.txt')
```

Python 2.7.14 Shell

File Edit Shell Debug Options Window Help

```
31 Damn your mood swings
32
33 I'm calling out
34 I don't really care for your city anymore
35 I spend the night
36 I lay awake and miss you when you go
37 Without me
38 What's wrong with you?
39
40 Monday Monday Monday
41 Monday Monday Monday
42 Monday Monday Monday
43 Monday Monday Monday
44
45 Oh, and I
46 I say damn your mood swings
47 Oh, and I
48 I say damn your mood swings
49 Damn your mood swings
50 Oh, and I
51 I say damn your mood swings
52 Damn your mood swings
53 Oh, and I
54 I say damn your mood swings
55 Damn your mood swings
Total number of lines:  55
>>>
```

Great! Print out some other songs.

Open `song_exercise.py` in the `exercise_1` folder.

We can use functions from outside our code by importing it.

Add the following line just beneath the header of `song_exercise.py`:

```
import line_counter
```

and run the code. What happens?

song_exercise.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py (2....

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018


'''

import line_counter

songs = ['days_go_by.txt', 'friday.txt', 'last_friday_night.txt',
         'monday.txt', 'summer_days.txt', 'sunday.txt', 'ten_days.txt'
         'tuesday.txt']
```

When we import the `line_counter.py` file, Python executes all of its code.

This will import the `line_counter` function so that we can use it in `song_exercise.py`. However, Python will also run the call to `line_counter(file_name)` that was added at the bottom of the file.

To fix this, make the following changes to `line_counter.py`:

*line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and c
'''

def line_counter(in_file):
    with open(in_file) as fin:
        for line_num, line in enumerate(fin):
            line = line.rstrip()
            print line_num, line

    print "Total number of lines: ", line_num

line_counter('monday.txt')
```

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and c
'''

def line_counter(in_file):
    with open(in_file) as fin:
        for line_num, line in enumerate(fin):
            line = line.rstrip()
            print line_num, line

    print "Total number of lines: ", line_num

if __name__ == '__main__':
    line_counter('monday.txt')
```

Rerun `line_counter.py`. It will still run just like we had it before because it was being run as a main. This is equivalent to calling python `line_counter.py` from the command line.

Rerun `song_exercise.py`. Now the function call isn't being performed because we're only importing `line_counter.py`, not running it as a main.

```
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py
>>>
```

Ln: 4288 Co

You can read a more detailed explanation of what's going on here.

So how do we call the `line_counter` function?

In IDLE, type import `line_counter`. This will import the function into the IDLE terminal.

Well, not exactly. It imports `line_counter` as a *module* into the environment.

In order to run the `line_counter()` function, we have to call it like this:

```
line_counter.line_counter(file_name)
```

This is like saying "Use the `line_counter` function from the `line_counter` module."

Call it the `line_counter` function a few times.

There are two annoying things that we are going to fix now.

First, it is annoying to type `line_counter.line_counter()` every time we want to use the `line_counter` function.

To fix this, we can modify our import statement to only import a specific function:

```
from line_counter import line_counter
```

This will only load the `line_counter` function into the environment. We will now be able to access the function by typing just `line_counter(file_name)`.

```
>>> from line_counter import line_counter
>>> line_counter('sunday.txt')
0 Sunday (feat. Heize, Jay Park) - GroovyRoom
1 I'll never let you go
2 Even if you yawn
3 Even if you half-heartedly listen to me
4 You never never know
5 I know everything
6 About how you went out last night after I fell asleep
7
8 You keep pulling that oppa card
9 The smell of alcohol covered by Jo Malone
10 But I don't care
11 Cuz we have something more than just rhythm between us
12 So when I saw your flustered smile
13 I grew more calm
```

We can also create an alias for a function as we import it by doing

```
from line_counter import line_counter as lc
```

This will enable us to call the `line_counter` function by typing `lc(file_name)`.

```
>>> from line_counter import line_counter as lc
>>> lc('sunday.txt')
0 Sunday (feat. Heize, Jay Park) - GroovyRoom
1 I'll never let you go
2 Even if you yawn
3 Even if you half-heartedly listen to me
4 You never never know
5 I know everything
6 About how you went out last night after I fell asleep
7
```

Doing this saves time/keystrokes, but often comes at the expense of readability.

What's easier to understand?

`line_counter('sunday.txt')` `lc('sunday.txt')`

Let's change the import line in `song_exercise.py`.

For this workshop, I won't be creating an alias for `line_counter`.

Your `song_exercise.py` should look like one of the following:

*song_exercise.py - C:/Users/pvill/Desktop/workshop/exercise/ex

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018


'''

from line_counter import line_counter

songs = ['days_go_by.txt', 'friday.txt', 'last
         'monday.txt', 'summer_days.txt', 'sur
         'tuesday.txt']
```

song_exercise.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018


'''

from line_counter import line_counter as lc

songs = ['days_go_by.txt', 'friday.txt', 'last_frid
         'monday.txt', 'summer_days.txt', 'sunday.t
         'tuesday.txt']
```

We can now access the `line_counter` function in `song_exercise.py` in fewer keystrokes

The second annoying thing is that our `line_counter` function is printing out the whole file when all we really want is the number of lines in the song.

Let's change that by

- commenting out the code inside the for loop,
- adding a pass statement in its place, and
- changing the final print statement to a return statement.

Your code should look like this.

We're only commenting out the liens inside of the for loop instead of deleting them because we may want them later for debugging purposes.

Notice the new print under main. Why is that there?

What about the `line = line.rstrip()` line? It's not printing anything, so why bother commenting it out?

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/l

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and co
'''

def line_counter(in_file):
    with open(in_file) as fin:
        for line_num, line in enumerate(fin):
##              line = line.rstrip()
##              print line_num, line
            pass


    return line_num

if __name__ == '__main__':
    print line_counter('monday.txt')
```

Running the code with `monday.txt` provides the following output.

```
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/line_counter.py
54
>>>
```

Ln: 4728 Col: 4

Much nicer to deal with than the giant block of text.

Now that we've dealt with those two issues, let's return to `song_exercise.py`.

Write a for loop to print out the number of lines in each song.

Answer:

```
for song in songs:
    print line_counter(song)
```

Run the code. What do you expect? What happened? Why?

```
>>>
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py
37
59
108
54
59
74

Traceback (most recent call last):
  File "C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py", l
ine 16, in <module>
    print line_counter(song)
  File "C:/Users/pvill/Desktop/workshop/exercise/exercise_1\line_counter.py", li
ne 10, in line_counter
    with open(in_file) as fin:
IOError: [Errno 2] No such file or directory: 'ten_days.txttuesday.txt'
>>>
```

The error we're getting says that "there is no such file or directory `ten_days.txttuesday.txt`".

Where is this error from? Find the error, fix it, then run the code again.

song_exercise.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py (2....

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018


'''

from line_counter import line_counter

songs = ['days_go_by.txt', 'friday.txt', 'last_friday_night.txt',
         'monday.txt', 'summer_days.txt', 'sunday.txt', 'ten_days.txt',
         'tuesday.txt']

for song in songs:
    print line_counter(song)
```

```
 RESTART: C:/Users/pvill/Desktop/workshop/exerc
37
59
108
54
59
74
213
66
>>>
```

Your code should be similar to the code above, and your output should be the same.

So far so good. But what was our original problem? We wanted to find the song that had the most lines in it.

How can we track of which song had the most lines in it?

Two possible solutions are:

```
from line_counter import line_counter

songs = ['days_go_by.txt', 'friday.txt', '
         'monday.txt', 'summer_days.txt',
         'tuesday.txt']

longest_song = ""
most_lines = 0
for song in songs:
    print line_counter(song)
    if line_counter(song) > most_lines:
        most_lines = line_counter(song)
        longest_song = song

print longest_song, most_lines
```

```
from line_counter import line_counter

songs = ['days_go_by.txt', 'friday.txt', 'last_friday_night.txt',
         'monday.txt', 'summer_days.txt', 'sunday.txt', 'ten_days
         'tuesday.txt']

song_line_counts = {}
for song in songs:
    print line_counter(song)
    song_line_counts[song] = line_counter(song)


longest_song = max(song_line_counts, key = song_line_counts.get)
most_lines = song_line_counts[longest_song]
print longest_song, most_lines
```

They both provide the same answer.

```
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py
37
59
108
54
59
74
213
66
ten_days.txt 213
>>>
```

Ln: 108 Col:

So when should you use one approach over the other?

What kind of tradeoffs are there?

For this workshop, we'll use the approach that *doesn't* use the dictionary.

We’re almost done! We found the song with the most lines. Now we just need to pull the song title and artist information and print it out a little nicer.

This information is on the first line of each of the song files in the format

Song name – Artist

The problem is that when Python reads a line in a file, it reads the entire line as one long string.

How can we separate this line into the format we want? Hint here.

We're going to use the split function (documentation). Every string object comes with the `split` function which takes as input a delimiting character (ie, the character to split the string on). By default, `split` splits a string on whitespace.

The function returns the list of strings you get when you break the original string into chunks on the delimiting character.

For example:

```
>>> test = "Sorry you couldn't reach me, I ran out of battery"
>>> test.split()
['Sorry', 'you', "couldn't", 'reach', 'me,', 'I', 'ran', 'out', 'of', 'battery']
>>> |
```

This doesn't transform the original object, so you have to save it to another variable if you want to keep this list.

So how can we use this to get the artist and song information from the files?

Since we know that the song and the artist are separated by a -, we can use `split` to separate the two into a list, then assign them both to new variables.

```
>>> info = "Sunday (feat. Heize, Jay Park) - GroovyRoom "
>>> song, artist = info.split("-")
>>> song
'Sunday (feat. Heize, Jay Park) '
>>> artist
' GroovyRoom '
>>>
```

Implement this code in `song_exercise.py`. A hint for one possible solution is here.

```
from line_counter import line_counter

songs = ['days_go_by.txt', 'friday.txt', 'l
         'monday.txt', 'summer_days.txt', '
         'tuesday.txt']

longest_song = ""
most_lines = 0

for song in songs:
##    print line_counter(song)
    if line_counter(song) > most_lines:
        most_lines = line_counter(song)
        longest_song = song

with open(longest_song) as fin:
    line = fin.readline()
    song, artist = line.split("-")

print song, artist, most_lines
```

```
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py
Ten Days of Falling (Baauer Remix)   Shlohmo
213
>>>
```

Ln: 151 Col: 4

We forgot to strip the newline character from the line before we saved it.

We'll do that now with strip.

```
with open(longest_song) as fin:
    line = fin.readline().rstrip()
    song, artist = line.split("-")


print song, artist, most_lines
```

```
>>>
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py
Ten Days of Falling (Baauer Remix)   Shlohmo 213
>>>
```

We almost have it the way we want, but there are some weird spaces between the artist and song.

We *could* modify the individual song and artist variables to get our desired result.

Instead of doing that, we're going to take advantage of list comprehensions to take care of all the formatting at once.

What is this line doing?

```
song, artist = [item.strip() for item in fin.readline().strip().split("-")]
```

- `fin.readline()` reads the first line of fin
- `.strip()` removes leading and trailing whitespace from that line
- `split("-")` separates the line into two chunks delimited by '–'
- Then, for each item in this list:
  - `item.strip()` removes leading and trailing whitespace
- Puts these items into a new list
- Finally, assigns the two values to song and artist.

```
with open(longest_song) as fin:
    song, artist = [item.strip() for item in fin.readline().strip().split("-")]


print song, artist, most_lines
```

```
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py
Ten Days of Falling (Baauer Remix) Shlohmo 213
>>>
```

The last thing we're going to do is make the print out a little more informative.

```
print "The song that had the most lines was {} by {}.  It was {} lines long.".format(
    song, artist, most_lines)
```

```
>>>
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py
The song that had the most lines was Ten Days of Falling (Baauer Remix) by Shloh
mo.  It was 213 lines long.
```

This is what I had, but try out different formats for yourself.

The code in the previous slide was an example of string formatting via the `string.format` function.

It's value wasn't apparent in such a simple example, but it's very powerful and allows you to programmatically format strings.

Some examples are on the next slide.

Order 1 transition matrix for test.fa

	B	C	E	G	H	I	S	T
B	0.02366	0.58573	0.03531	0.02184	0.05643	0.00000	0.13724	0.13979
C	0.03200	0.50021	0.11104	0.03023	0.08109	0.00002	0.14934	0.09606
E	0.00358	0.10419	0.81803	0.00407	0.00518	0.00000	0.03054	0.03442
G	0.00931	0.08848	0.02545	0.70935	0.03063	0.00000	0.06256	0.07422
H	0.00068	0.01482	0.00069	0.00266	0.91464	0.00004	0.01233	0.05414
I	0.00000	0.06061	0.00000	0.00000	0.03030	0.81818	0.00000	0.09091
S	0.02549	0.37001	0.08974	0.01829	0.05487	0.00005	0.36933	0.07223
T	0.01650	0.21240	0.06395	0.01291	0.04512	0.00004	0.12190	0.52718

Order 2 transition matrix for test.fa.

	B	C	E	G	H	I	S	T
BB	0.03125	0.62500	0.00000	0.00000	0.06250	0.00000	0.09375	0.18750
BC	0.02797	0.48788	0.05283	0.03108	0.09136	0.00000	0.14357	0.16532
BE	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
BG	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
BH	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
BS	0.01061	0.25729	0.08753	0.02653	0.02387	0.00000	0.54377	0.05040
BT	0.01302	0.04688	0.01562	0.01823	0.01042	0.00000	0.02344	0.87240
CB	0.02859	0.59604	0.03519	0.02419	0.05792	0.00000	0.12610	0.13196
CC	0.03148	0.52549	0.10726	0.02867	0.08247	0.00000	0.14211	0.08251
CE	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
CG	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
CH	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
CI	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000
CS	0.02524	0.24521	0.11980	0.01496	0.04845	0.00000	0.48902	0.05733
CT	0.00000	0.00000	0.00048	0.00360	0.02712	0.00000	0.00000	0.96880
EB	0.00000	0.44970	0.00000	0.00592	0.02367	0.00000	0.31361	0.20710
EC	0.01417	0.47267	0.01736	0.03452	0.05048	0.00000	0.23085	0.17997
EE	0.00440	0.12829	0.77593	0.00502	0.00637	0.00000	0.03760	0.04239
EG	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
EH	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
ES	0.01715	0.30041	0.03704	0.01235	0.02469	0.00000	0.55075	0.05761
ET	0.00060	0.01510	0.00121	0.00242	0.01087	0.00000	0.00483	0.96498
GB	0.01299	0.62338	0.03896	0.00000	0.05195	0.00000	0.10390	0.16883
GC	0.04128	0.46605	0.16511	0.02929	0.05859	0.00000	0.15446	0.08522
GE	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
GG	0.01325	0.12592	0.03622	0.58635	0.04359	0.00000	0.08903	0.10563
GH	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
GS	0.04589	0.47036	0.05927	0.02868	0.05927	0.00000	0.20650	0.13002
GT	0.02866	0.23408	0.10669	0.00637	0.05255	0.00000	0.08599	0.48567
HB	0.00000	0.56863	0.00000	0.05882	0.00000	0.00000	0.11765	0.25490
HC	0.02203	0.44141	0.04405	0.05463	0.09604	0.00000	0.18502	0.15683
HE	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
HG	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
HH	0.00074	0.01625	0.00076	0.00292	0.90642	0.00004	0.01351	0.05935
HI	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000
HS	0.01303	0.45603	0.02172	0.01629	0.05863	0.00000	0.30510	0.12921
HT	0.00564	0.19613	0.00735	0.01716	0.05492	0.00000	0.11375	0.60505
IC	0.00000	0.50000	0.50000	0.00000	0.00000	0.00000	0.00000	0.00000
IH	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
II	0.00000	0.07407	0.00000	0.00000	0.03704	0.77778	0.00000	0.11111
IT	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000
SB	0.01533	0.54598	0.04789	0.02874	0.08046	0.00000	0.12452	0.15709
SC	0.04259	0.45090	0.16643	0.03457	0.10858	0.00013	0.11739	0.07940
SE	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
SG	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
SH	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
SI	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000
SS	0.02570	0.42636	0.08988	0.01957	0.06445	0.00000	0.30639	0.06764
ST	0.00269	0.00000	0.00067	0.00269	0.03232	0.00000	0.00000	0.96162
TB	0.03088	0.62233	0.03563	0.01425	0.04038	0.00000	0.14727	0.10926
TC	0.03759	0.51403	0.15755	0.02140	0.06277	0.00000	0.14155	0.06511
TE	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
TG	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
TH	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
TI	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000
TS	0.03187	0.49258	0.07352	0.02335	0.06153	0.00000	0.21931	0.09782
TT	0.02791	0.33472	0.11470	0.01707	0.05361	0.00007	0.19454	0.25738

Both of the preceding matrices were printed out using the same code.

```
def print_transition_matrix(d):
    '''
        input:
            d - a transition matrix, represented as a dictionary of dictionaries

        output:
            print out of matrix representation of d
    '''
    out_format = "{:>10}{:^10}{:^10}{:^10}{:^10}{:^10}{:^10}{:^10}{:^10}"
    states = ['B', 'C', 'E', 'G', 'H', 'I', 'S', 'T']
    print out_format.format(" ", *states)

    out_format = "{:>10}{:10.5f}{:10.5f}{:10.5f}{:10.5f}{:10.5f}{:10.5f}{:10.5f}{:10.5f}"

    for key in sorted(d):
        print out_format.format(key, *[d[key][state] for state in states])
```

Here is a good source to learn more about `string.format()`.

The basic idea is that the {} in the string are placeholders that you supply as arguments to `string.format()`.

Back to `song_exercise.py`. Let's take a look at our longest song. Open up `ten_days.txt`. What's wrong?

It's full of blank lines! Then why is line counter still counting them?

Because they're not technically blank – they contain a newline character.

How can we test if a line is blank?

In our `line_counter` function, the lines that only contain a newline character should become empty strings when we do the `line = line.rstrip()` line.

Empty strings (and most other empty objects) in Python evaluate to False when in a Boolean statement.

```
>>> test = "\n"
>>> if test: print "Found something!"

Found something!
>>> test = test.strip()
>>> if test: print "Found something!"

>>> |
```

Let's implement this in `line_counter()`. Open up `line_counter.py` and fix the function.

Hint: You'll need to add a counter for the number of blank lines. See the alternate line counting solution on slide BLAH for how to do this.

Your code should look something like this.

line_counter.py - C:/Users/pvill/Desktop/workshop/exercise/exercise_1/line_counter.py (2

File Edit Format Run Options Window Help

```
'''
    Paul Villanueva
    BCBGSO Advanced Python Workshop
    3/23/2018

    Making a function that reads text files and counts the nu
'''

def line_counter(in_file):
    with open(in_file) as fin:
        blank_lines = 0
        for line_num, line in enumerate(fin):
            line = line.rstrip()
            if not line:
                blank_lines += 1
##            print line_num, line
            pass


    return line_num - blank_lines

if __name__ == '__main__':
    print line_counter('monday.txt')
```

When we run `song_exercise.py`, we get:

```
 RESTART: C:/Users/pvill/Desktop/workshop/exercise/exercise_1/song_exercise.py
The song that had the most lines was Last Friday Night by Katy Perry.  It was 88
 lines long.
>>>
```

There we go. Now our `line_counter` is correctly counting the number of lines in the song and correctly identifies the song with the longest number of lines.

How could we improve this program?

Exercise 2

Now it's your turn to implement something similar.

Go to the `exercise_2` folder.

There are several files containing ratings for stuff.

Your job is to find the highest rated item in each one.

I've supplied some commented starter code to guide you through this task.

You can implement much of the same code as we did in Exercise 1.

Work with your neighbor or search the internet if you get stuck.

Here's one solution for the `get_rating_info` function.

```
def get_rating_info(in_file):
    '''
        in - a file of ratings in the format "ITEM - RATING" separated by newlines.
        out - returns the name and rating for the highest rated item.
    '''
    with open(in_file) as fin:
        highest_item = ''
        highest_rating = 0
        for line_num, line in enumerate(fin, 1):
            current_item, current_rating = [item.strip() for item in line.strip().split("-")]
            current_rating = int(current_rating)
            if current_rating > highest_rating:
                highest_rating = current_rating
                highest_item = current_item

        return highest_item, highest_rating
```

Here’s one solution for the main section.

```
if __name__ == "__main__":
    # Create a list of the file names.  Call this list ratings_list.
    ratings_lists = ['books.txt', 'food.txt', 'games.txt', 'kdramas.txt', 'movies.txt']

    # For each of the items in this list, print out the highest rated item and its rating on
    # separate lines.  Make the print out nice.
    for rating_file in ratings_lists:
        highest_item, highest_rating = get_rating_info(rating_file)
        print "The highest rated item in {} was {} with a score of {}".format(rating_file, highest_item, highest_rating)
```

Exercise 3

More likely than not, someone has worked on a problem similar to the one you are working on. In these cases, it can save a lot of time and heartache to take other people's code and adapt it to your purposes.

In this exercise, we're going to work with someone else's code to solve some bioinformatics problems.

Go to the `exercise_3` folder. It contains:

- Two fasta files
- `bioinformatics_utilities.py`
- `bioinformatics_exercises.py`

Fasta files are text files containing genes and their nucleotide sequences.

`short.inti1.97.fasta` contains the sequences in `inti1.97.fasta` of length less than 200 nucleotides.

`bionformatics_utilities.py` contains functions and classes that we will be working with. We will look at `bioinformatics_exercises.py` later.

For now, open up `bioinformatics_utilities.py` in IDLE.

Spend a few minutes reading through the `main` section. Run the code.

```
if __name__ == '__main__':
    genes = read_fasta_file('short.intil.97.fasta')
    goi = genes[-1]
    best_score = 0
    best_gene = ''
    for gene in genes[:8]:
        current_score = get_alignment_score(goi, gene)
        print "Alignment score between {} and {}: {}".format(goi[0], gene[0], current_score)
        if current_score > best_score:
            best_score = current_score
            best_gene = gene
    print "Gene best aligned with {} was {} with alignment score of {}.\n".format(goi[0], best_gene[0], best_score)
    get_alignment_score(goi, best_gene, output = True)
```

What's being printed out? Can you trace what's happening?

Python 2.7.14 Shell

File Edit Shell Debug Options Window Help

```
 RESTART: C:\Users\pvill\Desktop\workshop\exercise\exercise_3\bioinformatics_uti
lities.py
Alignment score between JQ838000|idbi2921 and AJ319747|idbi777: 180
Alignment score between JQ838000|idbi2921 and AM231806|idbi1134: 64
Alignment score between JQ838000|idbi2921 and AM991327|idbi1006: 45
Alignment score between JQ838000|idbi2921 and AM991329|idbi1194: 60
Alignment score between JQ838000|idbi2921 and AY970968|idbi673: 62
Gene best aligned with JQ838000|idbi2921 was AJ319747|idbi777 with alignment sco
re of 180.

  Match Score     Mismatch Score     Gap Open Penalty     Gap Extension Penalty

      15               -20                 40                     2


          Sequence A: JQ838000|idbi2921
              Length: 15
          Sequence B: AJ319747|idbi777
              Length: 12

     Alignment Score: 180
              Length: 12
          Start in A: 4
          Start in B: 1
            End in A: 15
            End in B: 12

   Number of matches: 12
Number of mismatches: 0
    Match percentage: 1.000000

 Number of deletions: 0
Number of insertions: 0
Total length of gaps: 0

           .    :
    4 GGCGGTTTTCAT
      ||||||||||||
    1 GGCGGTTTTCAT
```

When run as `main`, `bioinformatics_utilities.py` does the following:

- Reads in all the sequences from `short.inti1.97.fasta` into the list genes.
- Chooses the last gene in `genes` as the gene of interest.
- Initialize highest alignment score to 0 and best gene to an empty string.
- Then, for the first 5 genes in `genes`:
  - Calculates the alignment score between the current gene and the gene of interest
  - If the alignment score is better than the highest alignment score:
    - Update highest score, best gene
- After going through all five genes, prints out the full alignment information between the gene of interest and the gene with the best alignment.

It’s not necessary to know local sequence alignment is done for this exercise, but you can learn more about what it is and how it’s calculated from these sources:

- MIT Algorithms for Computational Biology
- Globin Gene Server
- Wikipedia: Sequence Alignment
- https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Open up `bioinformatics_exercises.py` and work through the code.

Work with your neighbors and Google stuff.

Exercise 4

We did a little bit of file output in the last exercise. We're going to practice more of it in this exercise.

In this exercise, we will be working on data representing mathematical knots. If you'd like to learn more about knot theory, these are good places to start:

- Wikipedia: Knot Theory
- Knot Atlas
- Cornell: Introduction to Knot Theory

Go to the `exercise_4` folder. It contains:

- `gauss_codes.txt`, a file contain Gauss codes for some knots,
- `calc_wirt.py`, a module that calculates the Wirtinger number of a knot diagram, and
- `knot_calcs.py`, the exercise file.

Open up `calc_wirt.py` and run it.

```
Knot dictionary:

    STRAND      SUBSEQUENCE                    CROSSINGS OVER
       A        (-1, 2, -3)                    ('B', 'C')
       C        (-3, 1, -2)                    ('A', 'B')
       B        (-2, 3, -1)                    ('C', 'A')

Seed strand set: ('A', 'C')
Wirtinger number: 2
```

When run as `main`, `calc_wirt.py` prints out all of this information here.

Our goal will be to make a new file with the name of the knot, the Gauss code, the seed strand set, and the Wirtinger number.

You can read more about how the program works here.

Open up `knot_calcs.py` and work through the comments.

When finished, you should have a file that looks something like this:

knot_info - Notepad

File Edit Format View Help

```
3_1     ('A', 'C')      2
4_1     ('A', 'C')      2
5_1     ('A', 'E')      2
5_2     ('A', 'C')      2
6_1     ('A', 'C')      2
6_2     ('A', 'D')      2
6_3     ('A', 'B')      2
7_1     ('A', 'C')      2
7_2     ('A', 'C')      2
7_3     ('A', 'C')      2
7_4     ('A', 'C')      2
7_5     ('A', 'C')      2
7_6     ('A', 'B')      2
7_7     ('A', 'C')      2
8_1     ('A', 'D')      2
8_2     ('A', 'B')      2
8_3     ('A', 'B')      2
8_4     ('A', 'B')      2
8_5     ('A', 'C', 'E') 3
8_6     ('A', 'E')      2
8_7     ('A', 'C')      2
8_8     ('A', 'G')      2
8_9     ('A', 'F')      2
8_10    ('A', 'C', 'B') 3
8_11    ('A', 'C')      2
8_12    ('A', 'C')      2
8_13    ('A', 'G')      2
8_14    ('A', 'C')      2
8_15    ('A', 'C', 'G') 3
8_16    ('A', 'C', 'B') 3
8_17    ('A', 'C', 'E') 3
8_18    ('A', 'C', 'B') 3
8_19    ('A', 'C', 'H') 3
8_20    ('A', 'B', 'G') 3
8_21    ('A', 'C', 'F') 3
```

Conclusion

Some learning resources:

- Automate the Boring Stuff With Python
- Google's Python Class
- Stack Overflow

Online courses are great because they offer a structured learning plan.

However, the best way to learn Python (or any coding language) is to work on something you're personally interested in, such as an element of your research that can be automated.

Bye! Have a nice day!