

A Python programming primer for biologists

**(Named after *Monty Python's Flying Circus* &
designed to be fun to use)**

**Systems Biology/Bioinformatics
Edward Marcotte, Univ of Texas at Austin**

In bioinformatics, you often want to do completely new analyses. Having the ability to program a computer opens all sorts of research opportunities. Plus, it's fun!

Most bioinformatics researchers use a scripting language, such as Python, Perl, or R, rather than a compiled language like C++

These languages are not the fastest, not the slowest, nor best, nor worst languages, but they're easy to learn and write, and for many reasons, are well-suited to bioinformatics.

We'll spend the next 2 lectures introducing Python to give you a sense for the language and help introduce the basics of algorithms.

Python documentation: <http://www.python.org/doc/>
& tips: <http://www.tutorialspoint.com/python>

Good introductory Python books:

- *Learning Python*, Mark Lutz & David Ascher, O'Reilly Media
- *Bioinformatics Programming Using Python: Practical Programming for Biological Data*, Mitchell Model, O'Reilly

Good intro video (from a 2 day intro class at Google):

- <https://www.youtube.com/playlist?list=PLC8825D0450647509>

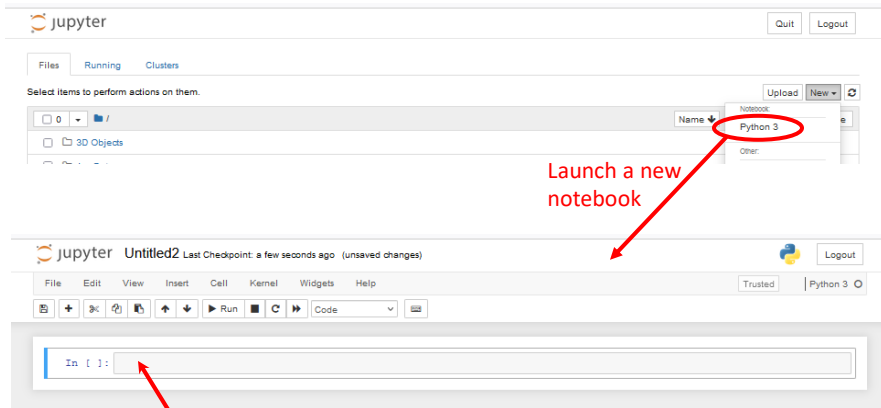
Practical Python, a self-paced online intro course:

- <https://dabeaz-course.github.io/practical-python/>

An online Python tutor with a nice interactive code viewer:

- <http://www.pythontutor.com/>

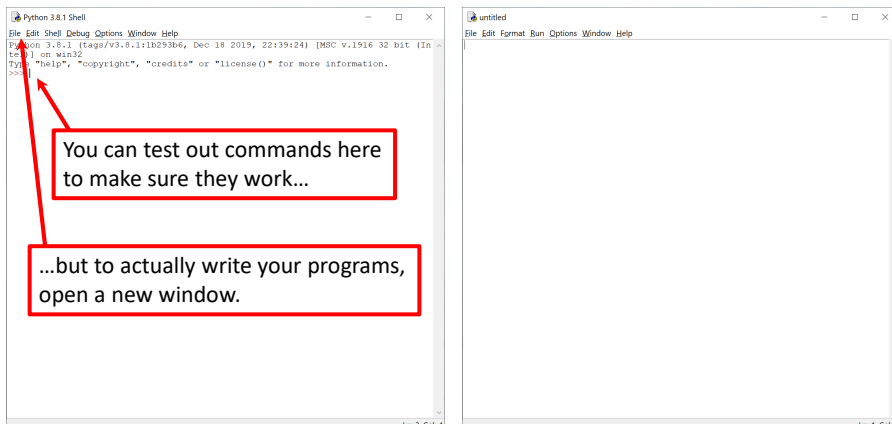
**By now, you should have installed Python on your computer.
If you're using Anaconda/Jupyter, it runs in a web browser:**



You can write your commands and programs here and they will be evaluated when you press Shift-Enter (or other options from the Cell pulldown menu)

Or if you installed IDLE by following the instructions in Rosalind Homework problem #1:

Launch IDLE:



This window will serve as a command line interface & display your program output.

This window will serve as a text editor for programming.

Let's start with some simple programs in Python:

A very simple example is:

```
print("Hello, future bioinformatician!") # print out the greeting
```

Run the program. In Jupyter, you can just type Shift-Enter & the output will appear below this cell of the notebook.

The output looks like this:

Hello, future bioinformatician!

FYI: This is version agnostic. Python 3 takes print("X"). Python 2 also takes print "X" as in Rosalind

A slightly more sophisticated version:

```
name = input("What is your name? ") # asks a question and saves the answer
# in the variable "name"
print("Hello, future bioinformatician " + name + "!") # print out the greeting
```

When you run it this time, the output looks like:

What is your name?

If you type in your name, followed by the enter key, the program will print:

Hello, future bioinformatician Alice!

FYI: Python 2.x uses raw_input() instead of input()


GENERAL CONCEPTS

Names, numbers, words, etc. are stored as *variables*.

Variables in Python can be named essentially anything except words Python uses as command.

For example:

```
BobsSocialSecurityNumber = 456249685  
mole = 6.022e-23  
password = "7 infinite fields of blue"
```



Note that strings of letters and/or numbers are in quotes, unlike numerical values.

LISTS

Groups of variables can be stored as lists.

A list is a numbered series of values,
like a vector, an array, or a matrix.

Lists are variables, so you can name them just as you would name any other variable.

Individual elements of the list can be referred to using [] notation:

```
The list nucleotides might contain the elements  
nucleotides[0] = "A"  
nucleotides[1] = "C"  
nucleotides[2] = "G"  
nucleotides[3] = "T"
```

(Notice the numbering starts from zero. This is standard in Python.)

DICTIONARIES

A VERY useful variation on lists is called a **dictionary** or *dict* (sometimes also called a *hash*).

→ Groups of values indexed not with numbers (although they could be) but with other values.

Individual hash elements are accessed like array elements:

For example, we could store the genetic code in a hash named *codons*, which might contain 64 entries, one for each codon, e.g.

```
codons["ATG"] = "Methionine"  
codons["TAG"] = "Stop codon"  
etc...
```

Now, for some control over what happens in programs.

There are two very important ways to control the logical flow of your programs:

if statements

and

for loops

There are some other ways too, but this will get you going for now.

if statements

```
if dnaTriplet == "ATG":  
    # Start translating here. We're not going to write this part  
    # since we're really just learning about IF statements  
else:  
    # Read another codon
```

Python cares about the white space (tabs & spaces) you use!
This is how it knows where the conditional actions that follow begin and end. **These conditional steps must *always* be indented by the same number of spaces (e.g., 4).**

I recommend using a tab (rather than spaces) so you're always consistent.

Note: in the sense of performing a comparison, not as in setting a value.

| | |
|----|-----------------------------|
| == | equals |
| != | is not equal to |
| < | is less than |
| > | is greater than |
| <= | is less than or equal to |
| >= | is greater than or equal to |

Can nest these using parentheses and Boolean operations, such as *and*, *not*, or *or*, e.g.:

```
if dnaTriplet == "TAA" or dnaTriplet == "TAG" or dnaTriplet == "TGA":  
    print("Reached stop codon")
```

for loops

Often, we'd like to perform the same command repeatedly or with slight variations.

For example, to calculate the mean value of the number in an array, we might try:

- Take each value in the array in turn.
- Add each value to a running sum.
- Divide the total by the number of values.

In Python, you could write this as:

```
grades = [93, 95, 87, 63, 75] # create a list of grades
sum = 0.0 # variable to store the sum
for grade in grades:
    sum = sum + grade # In general, Python cares whether numbers are
                        # integers or floating point (also long integers
                        # and complex numbers).
                        # You can tell Python you want floating point by
                        # defining your variables accordingly
                        # (e.g., X = 1.0 versus X = 1)
mean = sum / 5 # now calculate the average grade
print ("The average grade is ",mean) # print the results
```

| Python 2 | Python 3 |
|----------------|-----------------------|
| >>> 2 / 3 0 | >>> 2 / 3 0.666666 |

Python 2.x: print ("The average grade is "),mean

In general, Python will perform most mathematical operations, e.g.

| | |
|-----------------------|-----------------|
| multiplication | (A * B) |
| division | (A / B) |
| exponentiation | (A ** B) |

etc.

There are lots of advanced mathematical capabilities you can explore later on.

READING FILES

You can use a *for* loop to read text files line by line:

```
count = 0
file = open("mygenomefile", "r")
for raw_line in file:
    line = raw_line.rstrip("\r\n")
    words = line.split(" ")

# Print the appropriate word:
print ("The first word of line {0} of the file is {1}".format(count, words[0]))
count += 1

file.close()
print ("Read in {0} lines\n".format(count))
```

Stands for "read"

Declare a variable to count lines

Open a file for reading (r)

Loop through each line in the file

\r = carriage return
\n = newline

split the line into a list of words

shorthand for count = count + 1

Increment counter by 1

Last, close the file.

Placeholders (e.g., {0}) in the print statement indicate variables listed at the end of the line after the format command

Note: Python expects the file to be in your working directory or that you give it a full path.

WRITING FILES

Same as reading files, but use "w" for 'write':

```
file = open("test_file", "w")
file.write("Hello!\n")
file.write("Goodbye!\n")
file.close() # close the file as you did before
```

Unless you specify otherwise, you can find the new text file you created (test_file) in the default Python directory on your computer. In Jupyter, you should see now it appear in the Jupyter home page directory.

PUTTING IT ALL TOGETHER

```
seq_filename = "Ecoli_genome.txt"
total_length = 0
nucleotide = {} # create an empty dictionary

seq_file = open(seq_filename, "r")
for raw_line in seq_file:
    line = raw_line.rstrip("\r\n")
    length = len(line) # Python function to calculate the length of a string
    for nuc in line:
        if nuc not in nucleotide:
            nucleotide[nuc] = 1
        else:
            nucleotide[nuc] += 1
    total_length += length

seq_file.close()

for n in nucleotide.keys():
    fraction = 100.0 * nucleotide[n] / total_length
    print ("The nucleotide {0} occurs {1} times, or {2} %".format(n, nucleotide[n], fraction))
```

Let's choose the input DNA sequence in the file to be the genome of *E. coli*, available the class web site (& originally from the **Entrez genomes** web site)

The format of the file is ~77,000 lines of A's, C's, G's and T's:
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAGAGTGTG
TGATAGCAGCTTCTGAACCTGCTTACCTGCGTGAGTAAATTTAAATTTTATTGACTTAGG
TCACTAAATACTTTAACCAATATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTAC
ACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATCACCATTACCACAGGT
etc...

Running the program produces the output:

The nucleotide A occurs 1142136 times, or 24.619133255346103 %
The nucleotide G occurs 1176775 times, or 25.365788782211496 %
The nucleotide C occurs 1179433 times, or 25.42308288395832 %
The nucleotide T occurs 1140877 times, or 24.591995078484082 %

So, now we know that the four nucleotides are present in roughly equal numbers in the *E. coli* genome.

Finally, let's give you a new programming super-power with ChatGPT

ChatGPT is (1) truly amazing and powerful, and (2) a pathological liar. Caveat emptor.

Try it out, but don't trust it implicitly. It will give you an astonishing leg up with your programming, with the caveat that ***you have to check every single piece of code or fact supplied by it.*** It's like getting programming help from a gifted psychopath.



API RESEARCH BLOG ABOUT

ChatGPT: Optimizing Language Models for Dialogue

We've trained a model called ChatGPT which interacts in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests. ChatGPT is a sibling model to InstructGPT, which is trained to follow an instruction in a prompt and provide a detailed response.

[TRY CHATGPT](#)



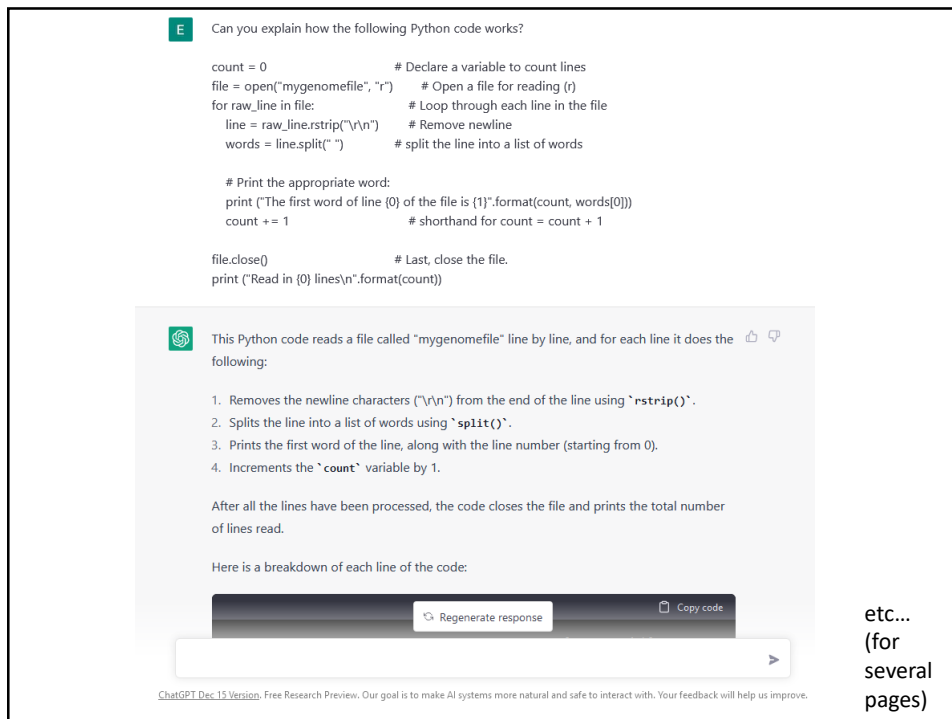
<https://openai.com/blog/chatgpt/>

At this stage, I don't want you to rely on it, especially if you're just getting started and don't know enough to recognize when it's wrong. Please don't ask it write full programs for you or answer homework problems.

However, a few things you might find helpful at this stage:

- (1) Ask it to explain a line of code to you
- (2) Ask it to explain programming syntax, suggest an alternative syntax, or a more compact way to perform the same task
- (3) Debugging, debugging, debugging. Give it your code (if there's not too much to it) and ask it to debug.

Just as an example, here's what chatGPT offers if we give it the example code for reading a file from 5 slides back.



The screenshot shows a ChatGPT chat window. The user asks: "Can you explain how the following Python code works?". The code is as follows:

```
count = 0 # Declare a variable to count lines
file = open("mygenomefile", "r") # Open a file for reading (r)
for raw_line in file: # Loop through each line in the file
    line = raw_line.rstrip("\n") # Remove newline
    words = line.split(" ") # split the line into a list of words

    # Print the appropriate word:
    print ("The first word of line {0} of the file is {1}".format(count, words[0]))
    count += 1 # shorthand for count = count + 1

file.close() # Last, close the file.
print ("Read in {0} lines\n".format(count))
```

The AI response explains the code line by line:

- 1. Removes the newline characters (" \backslash n") from the end of the line using `rstrip()`.
- 2. Splits the line into a list of words using `split()`.
- 3. Prints the first word of the line, along with the line number (starting from 0).
- 4. Increments the `count` variable by 1.

After all the lines have been processed, the code closes the file and prints the total number of lines read.


Here is a breakdown of each line of the code:

At the bottom of the chat window, there are buttons for "Regenerate response" and "Copy code".

etc...
(for several pages)

ChatGPT Dec 15 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

E Can you write a more compact version of the program?

Certainly! Here is a more compact version of the program that achieves the same result: 

```
Copy code
count = 0
with open("mygenomefile", "r") as f:
    for line in f:
        words = line.split()
        print(f"The first word of line {count} of the file is {words[0]}")
        count += 1
print(f"Read in {count} lines")
```

Here is a breakdown of the changes made:

- The `open()` function is called using the `with` statement, which automatically closes the file after the block of code is executed.
- The `raw_line` variable is not needed, so it was removed and the `line` variable was used directly.
- The `rstrip()` method is not needed, because the `for` loop already removes newline characters.
- The `format()` method is replaced with f-strings, which provide a more concise and readable way to insert values into strings.

I hope this helps! Let me know if you have any questions.