**A Python programming primer CH364C/CH391L Bioinformatics** Spring 2013

Python: named after *Monty Python’s Flying Circus*

 (designed to be fun to use)

**Starting with some example programs in Python:**

**A very simple program is:**

#!/usr/bin/python

# That was the only mandatory line (and really, you can even leave it out!)

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

Let’s call it hello.py

chmod +x hello.py

./hello.py

The output looks like this:

Hello, future bioinformatician!

**A slightly more sophisticated version:**

#!/usr/bin/python

# That was the only mandatory line (and really, you can even leave it out!)

name = raw\_input("What is your name? ") # acts a question and saves the answer

 # in the variable "name"

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

Call this one hello2.py

chmod +x hello2.py

and run it:

./hello2.py

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!

**A note about versions:**

Most bioinformaticians use Python 2.7. Check yours with:

python --version

**Some general concepts:**

Names, numbers, words, etc. are stored as ***variables***. Variables in Python can be named essentially anything except words Python uses as command.

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", and nucleotides[3] = "T".

Notice, the numbering starts from zero.

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:

We could store the genetic code in a hash named codons, which might include 64 entries, one for each codon, such as codons["ATG"] = "Methionine" and codons["TAG"] = "Stop codon".

**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

== 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 parenthesis and Boolean operators, such as and*,* not, or or, e.g.:

if dnaTriplet == "TAA" or dnaTriplet == "TAG" or dnaTriplet == "TGA":

 print("Reached stop codon")

**for *loops***

#!/usr/bin/python

grades = [93, 95, 87, 63, 75] # create a list of grades

sum = 0 # variable to store the sum

for grade in grades: # iterate over the list called grades

 sum = sum + grade # indented commands are executed on

 # each cycle of the loop.

mean = sum / 5 # now calculate the average grade

print ("The average grade is "),mean # print the results

Mathematical operations:

Python will perform most mathematical operations, e.g.

multiplication (A \* B)

division (A / B)

exponentiation (A \*\* B)

and so on. There are extensive mathematical capabilities you can explore later.

**Reading and writing files**

**In general, use a *for* loop to read files:**

#!/usr/bin/python

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("\r\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)

**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

**Putting it all together**

#!/usr/bin/python

seq\_filename = "DNA1"

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)

 for nuc in line:

 if nucleotide.has\_key(nuc):

 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 named DNA1 to be the genome of *E. coli*, which we can download from the **Entrez genomes** web site http://www.ncbi.nlm.nih.gov/nuccore/49175990?report=fasta (also on our class web site)

The format of the file DNA1 will be line after line of A’s, C’s, G’s and T’s, such as:

agcttttcattctgactgcaacgggcaatatgtctctgtgtggattaaaaaaagagtgtc... and so on.

Running the program produces the output:

The nucleotide A occurs 1142136 times, or 24.6191332553 %

The nucleotide C occurs 1179433 times, or 25.423082884 %

The nucleotide T occurs 1140877 times, or 24.5919950785 %

The nucleotide G occurs 1176775 times, or 25.3657887822 %