Introduction to Python Programming for Bioinformatics

Rob Harbert

4/27/2018

Python Workshop

This is part of the AMNH Sackler Institute for Comparative Genomics workshop series

This workshop will cover:

- Insalling and running Python and Anaconda for package management
- Data and variable types and structures
- Reading and writing common file types (e.g., csv)
- Basic programming in Python through interactive and scripted sessions
- Using some common libraries in Bioinformatics

To install Python

MacOS https://conda.io/docs/user-guide/install/macos.html

Windows https://conda.io/docs/user-guide/install/windows.html

Python Variables

Much like R, Python has distinct integer and numeric classes. In Python these are:

Integers

```
x=2
print(x)
```

```
## 2
```

```
print(type(x))
```

```
## <type 'int'>
```

Floating Points Numbers

```
y=2.2
print(y)
```

```
## 2.2
```

```
print(type(y))
```

```
## <type 'float'>
```

In Python integers and floating points interface pretty well

```
z = x*y
print(z)
```

```
## 4.4
```

```
type(z)
```

Character Strings

Text is stored in the string variable type. Each character is indexed in an ordered list.

```
str = 'The quick brown fox jumps over the lazy log';
print(str[0])
```

```
## T
```

```
print(str[1])
```

```
## h
```

```
print(str[0:8])
```

```
## The quic
```

Explicit variable calling

```
n = float(1)
print(n)
```

```
## 1.0
```

```
print(type(n))
```

```
## <type 'float'>
```

Array

Stores multiple data objects of type integer, float, or string. Can be mixed. Denoted by brackets "[]"

```
arr = [0,1,2,3,4]
print(arr)
```

```
## [0, 1, 2, 3, 4]
```

```
arr2 = [0, 1, "two", "three", 4.4]
print(arr2)
#values can be changed
```

```
## [0, 1, 'two', 'three', 4.4]
```

```
arr[0] = 2
```

Arrays can be arrays of arrays, which is kind of like a data.frame or matrix.

```
twodarr = [arr, arr, arr, arr]
print(twodarr[1])
```

```
## [2, 1, 2, 3, 4]
```

```
print(twodarr[1][1]) #row 1 column 1
```

```
## 1
```

This is funky so if you want R data.frame like objects us Pandas in Python. http://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html

Tuples

Imutable array-like objects. Denoted by parentheses insead of brackets. Try setting a value in an existing tuple.

```
tup = (0,1,2,3)
print(tup)
```

```
## (0, 1, 2, 3)
```

```
print(tup[0])
```

```
## 0
```

```
tup_b = ('the', 'quick', 'brown', 'fox')
print(tup_b)
```

```
## ('the', 'quick', 'brown', 'fox')
```

```
print(tup_b[3])
```

```
## fox
```

What happens if we try and set a value in a tuple?

Dictionaries

These are lists of key/value pairs. In Perl these are Hash objects. R does not have an equivalent to this but it is kind of like row/column names. The key is set to look-up the values. These can be useful for translation, or information lookup (e.g., zipcodes)

```
translation = {'one': 1, 'two': 2}
print(translation['one'])
#Values can also be set
```

```
## 1
```

```
translation['one'] = 5
print(translation['one'])
#Keys are added like:
```

```
## 5
```

```
translation['five'] = 1
print(translation['five'])
#Values can be anything
```

```
## 1
```

```
translation['six'] = [0,1,2,3,4]
print(translation['six'])
```

```
## [0, 1, 2, 3, 4]
```

Operators

Tutorial: https://www.programiz.com/python-programming/operators

Math

Mostly like R. But compare "**" and "^. In python" is a bitwise operator (bit "OR") which we will ignore for now.

```
print(x+7)
```

```
## 9
```

```
print(x*2)
```

```
## 4
```

```
print(x**3)
```

```
## 8
```

```
print(x^3)
```

```
## 1
```

Logic

```
print(x==y)
```

```
## False
```

```
print(x>y)
```

```
## False
```

```
print(x<=y)
```

```
## True
```

Operators on strings

"+" concatenates strings. "==" compares exact string identity Other operations don't behave like you might expect. ">" and "<" perform byte comparisons of the character codes and are usually not useful.

```
str1 = "hello"
str2 = "world"
print(str1+str2)
```

```
## helloworld
```

```
print(str1+" "+str2+"!")
```

```
## hello world!
```

```
print(str1==str2)
```

```
## False
```

```
print(str1=='hello')
```

```
## True
```

```
print(str1>str2)
```

```
## False
```

```
print(str2>str1)
```

```
## True
```

Python Functions, Libraries, and Loops

Functions

We have already seen a few functions. For example print(). All functions have the same format: name (object)

Some useful functions: Range - cast a sequence of numbers from i to n by x

```
r=range(0,10,1)
print(r)
```

```
## [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
r2=range(0,20,2)
print(r2)
```

```
## [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
r3=range(10,0,-1)
print(r3)
```

```
## [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Join - Concatenate strings b

```
bases=["A","C","G","T"]
#join
print(''.join(bases))
```

```
## ACGT
```

```
print(' '.join(bases))
```

```
## A C G T
```

```
print('_'.join(bases))
```

```
## A_C_G_T
```

```
print("\t".join(bases))
```

```
## A    C   G   T
```

```
print(' '.join(str1))
```

```
## h e l l o
```

Length - get length of array or string

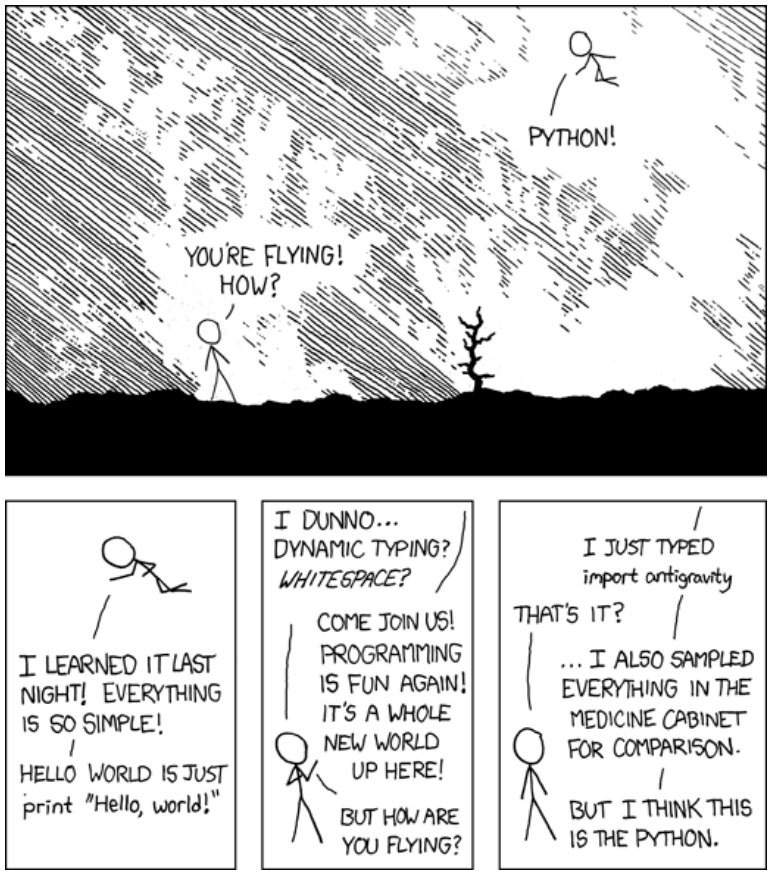
```
print(len(str))
```

```
## 43
```

```
print(len(arr))
```

```
## 5
```

Libraries/Import



Like R, the power of python comes from code libraries. Libraries are called bye the 'import' statement and libraries can be aliased using "import lib as alias".

```
import random
print(random.choice(bases))
```

```
## G
```

```
print(random.choice(bases))
```

```
## G
```

```
import random as r
print(r.choice(bases))
```

```
## C
```

Loops

"For" loops are programming structures that are useful for repeating things. There are often better, more efficient ways to do this like the "apply" family of functions in R. For now we will stick with basic for loops because they are highly useful.

```
for i in [1,2,3,4]:
  print(i)
```

```
## 1
## 2
## 3
## 4
```

Instead of declaring the list of values to iterate through we could also call range and give the number:

```
for i in range(10):
  print(i**3)
```

```
## 0
## 1
## 8
## 27
## 64
## 125
## 216
## 343
## 512
## 729
```

Random character strings

```
import random as r
n=1999
seq=range(n)
for i in seq:
  seq[i] = r.choice(bases)

seqstr=''.join(seq)
print(seqstr[1:50])
```

```
## CCCTTCATTTAGTGCCAATAGCCCCGTCGTCCCGAAGCGCGTAGACCCG
```

String matching with Regular Expressions

The 're' library

```
import re
m=re.findall("TTT",seqstr[0:100])
print(m)
```

```
## ['TTT']
```

```
print(len(m)) #the number of matches to TTT in seq
#re.sub(pattern, replace, string)
```

```
## 1
```

```
s=re.sub('A', 'N', seqstr)
print(s[0:50])
```

```
## NCCCTTCNTTTNGTGCCNNTNGCCCCGTCGTCCCGNNGCGCGTNGNCCCG
```

Reading and parsing text files

To open text files we want to open a filehandle (a variable pointing to a file to read) with the open function. 'with open("file.txt") as filehandle:' is a statement that reads Open file.txt and refer to it with the variable filehandle and then do (":") something. In the case below we will just read the whole file into a character string or into an array of character strings.

```
##read text file
with open ("./data/98.txt", "r") as myfile:
    data=myfile.read()
print(data[0])
#Try also:
```

```
## T
```

```
with open ("./data/98.txt", "r") as myfile:
    data2=myfile.readlines()
print(data2[0])
```

```
## The Project Gutenberg EBook of A Tale of Two Cities, by Charles Dickens
```

With the file read into a single character string we will use 're' to parse the text in any way you might use 'grep'.

```
lines=re.findall("\n", data)
print(len(lines))
```

```
## 16273
```

```
it=re.findall("It", data)
itwas=re.findall("It was", data)
itwasthe=re.findall("It was the ", data)
itwasthebest=re.findall("It was the best", data)
```

Challenge:

- How many words are there in this file?
- How many lines?
- How many chapters?

Searching Nucleotide strings

For object seqstr, find all start codons AUG.

```
transl = re.sub('T', 'U', seqstr)
startc = re.findall("AUG", transl)
#get position of first match, use 'search' instead of 'findall'.
m = re.search("AUG", transl)
first = m.span()
print(first) #will give start and end of first match..
#get position of all matches
```

```
## (231, 234)
```

```
m2 = re.finditer("AUG", transl) #returns an iterator object, must loop
for n in m2:
    coord=n.span()
    print(coord[0])
    print(transl[coord[0]:coord[0]+20])
```

```
## 231
## AUGAUGCCAUCGUUUACUCG
## 234
## AUGCCAUCGUUUACUCGGGU
## 273
## AUGACCACAGCUACCCUGUU
## 296
## AUGGCCCGAUCUGUCUUAGG
## 459
## AUGUUUGGAAGUAGAGGUCA
## 546
## AUGUACCGAUGCCUUUCAUG
## 554
## AUGCCUUUCAUGCUCAUAAA
## 563
## AUGCUCAUAAAACCAAUACC
## 599
## AUGGUUAAACGCAUUACAUG
## 616
## AUGAGCAAAGAUGUGAUAUU
## 626
## AUGUGAUAUUGAUGAUAAGG
## 637
## AUGAUAAGGAAAUCGUCAGC
## 686
## AUGUAUACGACUACCGUACC
## 991
## AUGUCCCACUGCCAUGAUGU
## 1004
## AUGAUGUAGGAUAUCCCUGA
## 1007
## AUGUAGGAUAUCCCUGAGGC
## 1028
```

```
## AUGCUGUGCAAUUUGGGCAC
## 1082
## AUGUAACGCCCGGUCUUCAC
## 1118
## AUGGCUGUGUUACCUACAGU
## 1266
## AUGCAAUCCAAGUCCCUGUA
## 1398
## AUGAGGCUCACUUCUGGCAA
## 1419
## AUGACUUUAGCCAUCGCCAC
## 1477
## AUGGAUGACGCUCGACUCCG
## 1481
## AUGACGCUCGACUCCGUUCC
## 1562
## AUGUAACGGAGCCCGCUGCC
## 1694
## AUGAGAAUGAGAUAGGUAAU
## 1700
## AUGAGAUAGGUAAUUGGGAC
## 1745
## AUGGUGUGAUACACUUUGUC
## 1838
## AUGAUCGUUUGACUUAUAUA
## 1911
## AUGCCGAGCAGUUGACUCGA
## 1957
## AUGGACGGCUCAUUACGACA
```

Challenge

What would we need to get all possible open reading frames and return those as an array?

How could we use a Python dictionary object to translate these ORFs.

Reading CSV files to Pandas

How to install pandas

"conda install pandas"

Writing Python Functions

```
def add(x, y):
  return x + y

add(1,4)
```