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

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

```
## 4.4
```

```python
type(z)
```

Character Strings

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

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

```
## T
```

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

```
## h
```

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

```
## The quic
```

Explicit variable calling

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

```
## 1.0
```

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

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

Array

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

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

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

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

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

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

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

```python
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

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
```

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

```
## True
```

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

```
## False
```

```python
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

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

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

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

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

Join - Concatenate strings b

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

```
## ACGT
```

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

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

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

```
## A_C_G_T
```

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

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

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

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

Length - get length of array or string

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

```
## 43
```

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

```
## 5
```

Libraries

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
```

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])
```

```
## CATAGTCGTGGCGTCACGAATCGGGAGCGGGCCATGATAACGTCCACGT
```