Python for Data Processing

A class for people who want to do science faster

Why Python for Science?

- A **readable** language for people without a lot of production time (i.e. Grad students)
- Pre-installed on Mac OS and Linux
- Tons of **free** tools (i.e. Scipy for stats, Matplotlib for graphing, and Numpy for matrix and array manipulation)
- More general, can do more things (more useful to learn in the long run) than R or MATLAB

Overall Objectives

- Slides and sample files:

  http://sacklerpython.wordpress.com

- Teaching the basics of Python with a focus on data processing, bioinformatics, statistics, and plotting
- Each meeting should be in the following format:
  - Recap (except today)
  - Lesson
  - Deduction
  - Problem solving in code

Installing Python

- **Mac OS and Linux:**
  - Go to Finder or Dash Home
  - Search for 'Terminal'
  - When it loads, type 'python', press Enter
  - That should be it (**Cntl+D** to exit)
- **Windows:**
  - We should have already done this for you, but if not, one of us will do it now!

Don't Get It Confused

- Terminal: uses UNIX
- Use Terminal to get to Python interpreter by writing 'python' and Enter
- Once there, the rules change: now it uses Python until you hit Cntl+D

```
thefourtheye@thefourtheye-Inspiron-5520:~$ echo 'Hello World'
Hello World
thefourtheye@thefourtheye-Inspiron-5520:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Hello World'
Hello World
```

Changing Directories in Terminal

- Terminal starts in your home directory, so you need to create somewhere to keep your files, and then you need to point Terminal to that location before running Python
- Go ahead and create a folder on your Desktop called 'myPython'
- Changing directories is the command 'cd'
  - 'cd $HOME/Desktop/myPython' will take you to your new folder in Terminal
  - 'cd ..' will go up a folder
  - 'cd' by itself will always take to your home folder

```
thefourtheye@thefourtheye-Inspiron-5520:~$ cd Desktop/myPython
thefourtheye@thefourtheye-Inspiron-5520:~/Desktop/myPython$ cd ..
thefourtheye@thefourtheye-Inspiron-5520:~/Desktop$ cd
thefourtheye@thefourtheye-Inspiron-5520:~$
```

Getting Started in Python

- Now type 'python' in Terminal (for realsies)

```
thefourtheye@thefourtheye-Inspiron-5520:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> #This is a comment and does nothing
>>> seq = 'GATTACA'        #This is a string
>>> num = 56       #This is a number
```

- '#' followed by text makes a comment, which is for annotating your code for future reference
- Anything after a '#' is a comment, no matter where it goes or what it says

Variables and Assignments

```
>>> seq = 'GATTACA'
>>> num = 56
>>> print seq
GATTACA
>>> print num
56
```

- Variables like 'num' and 'seq' are just stand-in names for data, like numbers, strings, lists, etc.
- It's a way of storing large amounts of data in a simple reference (like how a person's name can bring to mind an entire individual)

What can be stored in a variable?

- Any Python data type can be stored

Object type	Category	Mutable?
Numbers	Numeric	No
Strings	Sequence	No
Files	Extension	N/A
Lists	Sequence	Yes
Dictionaries	Mapping	Yes
Tuples	Sequence	No

Numbers in Python

- All you need to know about numbers, you learned by typing equations into the Google search bar (+ - * / **)
- 8 is an **integer**, and 8.0 is a **float**
- Note: integer division in Python 2.X is weird with fractions less than 1, **so always divide floats for now**. Fixed in Python 3.0
- For logarithms and trigonometry, etc. **import math**

```
>>> n = 8
>>> n/2
4
>>> n = 8
>>> n/16
0
>>> n = 8.0
>>> n/16
0.5
```

```
>>> n = 2
>>> n**5
32
>>> n**5.0
32.0
```

```
>>> import math
>>> math.log(1000/10, 10)
2.0
>>> math.log(1000/10)
4.605170185988092
>>> math.exp(2)
7.38905609893065
```

A brief note on importing modules

- The basic Python package has a lot of tools, but to do some things, e.g. make an array, perform logarithms, you need to import a module
- To find out if what you want to do needs importing and what the module is called, just Google it
- To import it, **just write 'import modulename'**, and **add it to the beginning of every tool in the module with a period**

```
>>> import math
>>> math.log(1000/10, 10)
2.0
>>> math.log(1000/10)
4.605170185988092
>>> math.exp(2)
7.38905609893065
>>>
>>> import numpy as np
>>> np.array([1,2,3,4])
array([1, 2, 3, 4])
```

Strings in Python

- Strings are just sequences of characters and spaces in quotes ('string' or “string”)
- DNA and protein sequences are almost always represented as strings
  - Parts of strings are substrings, i.e. motifs
- You can slice them, transform them, and get information, just like you would want to for sequences

```
>>> dna = 'CGGTTAATAGGGACTCTC'
>>> protein = 'KMLVVPRF'
```

Slicing Strings in Python

- You can get parts of strings by **slicing**
- This is when you specify the beginning and the end of the part of the string you want, and create a new string as a result
  - **newstring = string[beg:end:step]**
- Slicing actually creates a copy of the string that you can save in another variable
  - It does not modify the string itself, so your original string is intact unless you save over it

```
>>> dna = 'CGGTTAATAGGGACTCTC'
>>> motif = dna[4:10:1]
>>> motif
'TAATAG'
>>> dna
'CGGTTAATAGGGACTCTC'
```

Slicing Strings in Python

- Python has a 0-based numbering system, so counting goes: 0, 1, 2, 3, 4, etc.
- Indexing **[beg:end:step]**
  - Beginning: count to the letter you want starting by 0
  - End: (n-1) by the 0-based system
  - Step: 1 is every letter, 2 is every second letter, 3 is every third, etc.
- If **beg** or **end** is negative, it will count backwards from the end of the string
- If **beg** or **end** is left out, it defaults to 0 and the end, respectively
- You can **reverse** a string by adding [::-1] to the slice or string

```
>>> dna = 'CGGTTAATAGGGACTCTC'
>>> dna[4:10:2]
'TAA'
>>> dna[-2:-1]
'CT'
```

```
>>> dna[2:]
'GTTAATAGGGACTCTC'
>>> dna[2:][::-1]
'CTCTCAGGGATAATTG'
```

Strings in Python: Exercises

```
>>> dna = 'CGGTTAATAGGGACTCTC'
```

```
>>> dna[0]
>>> dna[0:3]
>>> dna[-1]
>>> dna[-1:-3]
>>> dna[-3:-1]
```

```
>>> dna[0:5]
>>> dna[0:5:2]
>>> dna[0:5][::-1]
>>> dna[0:5][::-2]
>>> dna
```

Try to guess the answer using what you've learned first.

Strings in Python: Answers

```
>>> dna = 'CGGTTAATAGGGACTCTC'
```

```
>>> dna[0]
'C'
>>> dna[0:3]
'CGG'
>>> dna[-1]
'C'
>>> dna[-1:-3]
''
>>> dna[-3:-1]
'CT'
```

```
>>> dna[0:5]
'CGGTT'
>>> dna[0:5:2]
'CGT'
>>> dna[0:5][::-1]
'TTGGC'
>>> dna[0:5][::-2]
'TGC'
>>> dna
'CGGTTAATAGGGAC
TCTC'
```

String Methods and Functions

- This is a method: **string.method()**
  - dna.count('A') # counts the number of adenines in your sequence
  - dna.find('GGGA') # finds exact motif and returns index (or -1)
  - dna.replace('GG', 'CC') # replaces all occurrences of 'GG' with 'CC'
  - And many, many more! (The real power of strings comes with **regular expressions** that deal with inexact matches, which will be covered later)
- This is a function: **function(string)**
  - len(dna) # gives you the length of the sequence
  - str(anything) # turns anything (like numbers) into a string
  - And many more! (I rarely use functions on strings, TBH)

Strings in Python: Exercises

```
>>> dna = 'CGGTTAATAGGGACTCTC'
>>> dna.count('A')

>>> dna.find('GGGA')

>>> dna.replace('GG', 'CC')

>>> dna[0:9].count('T')

>>> dna[0:len(dna)/2].find('T')
```

Try to guess the answer using what you've learned first.

Strings in Python: Answers

```
>>> dna = 'CGGTTAATAGGGACTCTC'
>>> dna.count('A')
4
>>> dna.find('GGGA')
9
>>> dna.replace('GG', 'CC')
'CCCTTAATACCGACTCTC'
>>> dna[0:9].count('T')
3
>>> dna[0:len(dna)/2].find('T')
3
```

Problem Solving

- Go to **sacklerpython.wordpress.com** and download the file '**sample1.fa**' into your **myPython** folder (your active folder in Python should still be this folder)
- Now in your Python interpreter, type the following to open the file:
  - After typing the colon, hit Enter and then Tab before typing 'dna' to indent the second line (it's a long story, I'll explain later)

```
>>> with open('sample1.fa', 'r') as file:
        dna = file.read()
```

- Now your FASTA file is stored as a string in the variable **dna**. (Note that our previous string, also called **dna**, is now erased from memory)
- Take a look at **dna**.

Problem Solving

```
>>> dna
'TAGCGTCAATACCGGGTAGCAGGCTGAGAATCCCTTGTAACAGTCTGTTA
CCCCCCCGAACAGAAGCCTTTGACTCTGAACGCACTGTAACGAAGTATGT
TACAGCGTTTACAGCCAACGAAATAGCTTTTGCAGCGGCGTCAAGGAGAC
GGACCACTGGGGAGTAAAGGACTGGCATCAACGTACGTCCACGCAATCA
CGGCTGTGAGCCCATGACAGTGACTTTCACTGGCCATCTTTCTCAGTCAA
CAGATAGTCTCCTATCCAGGCGCTAATTCATCCTCACACGAATGGATATTAG
TATAGCCCTTACAACGGGCCATCGGACCAACTTTAAGCAAACGGACAAGG
AAACGGATACCTTTCCTACACAAAGGTTCACCGCGCCCGATGAGAGGTCA
AGGTGACGCATATCCTTCTCGTAAGTCACTTGCGGCGGGGGGTATAACCT
TCAGTGTGACCAATCGTAGGAAATAGAACGGCAGCCCCAAAGCGCGTGC
AGGTGCGTCATTACGTCTTAGGTGATAGCAGGTATTAGGCCTAGAGAACAT
CATCAATGCCGCGATGCACCTTCTGCTGGCGATGACAGCTCATTCTCTTT
GAGCCAATTGAGTGACTACCGACCTAGCGGCCTCCAGTACGAAGATAATA
GGCGCCAAGAGTGGCGTGTCATTCCGTTAGCAGCATGCGCCAAGATGGG
TTCCGGGATAGGAGCAGAGGAAGCACAAGGCAAACGCGACTAGGCATGT
TGGAAGTTCAGAAGGACTCCTCCGCATCTTACCTACAATCGCACCCACATA
GCTAGCATATGTCAAGATTCGCATGCTTAACGAAAGGTGGGGATACTTGAA
CATTTGAGTTTCCTCTTCGGGCCGGCGATCTCGTC\n'
```

Problem Solving

- Problem 1: See the '\n' at the end? We don't want that, but getting rid of it automatically is for another lesson. For now, use string slicing to cut it out.
- Problem 2: Locate the motif 'TGAGCCCATG' in the sequence. Hint: what is the index?
- Problem 3: What is the GC content of the DNA to 2 decimal spaces?
- Problem 4: If we put the above subsequence into a variable called '**motif**', reproduce it only by slicing **dna**. ***Bonus:*** Make it so that if I put another sequence in **motif**, the code would still work unmodified and reproduce the second motif by just slicing **dna** (hint: use methods and functions in the slice indexes and refer to the subsequence by its variable name).

```
>>> motif = 'TGAGCCCATG'
>>> dna = [?:?]
>>> motif = 'GGAAATAGAACGG'
>>> # will your code still work? (Bonus)
```

Problem Solving

- Problem 1: dna[:-1]
- Problem 2: 205
- Problem 3: 51.18%
- Problem 3: I will accept
  - dna[205:215]
  - dna[dna.index(motif):dna.index(motif)+len(motif)]

```
>>> motif = 'TGAGCCCATG'
>>> dna[dna.index(motif):dna.index(motif)+len(motif)]
'TGAGCCCATG'
>>> motif = 'GGAAATAGAACGG'
>>> dna[dna.index(motif):dna.index(motif)+len(motif)]
'GGAAATAGAACGG'
```