An introduction to Data and Computing for Scientists and Engineers

Outline:

- Objectives and big picture
- Introductions
- Overall syllabus and schedule
- Grading - homeworks and projects
- Next three weeks --- Python in the Jupyter notebook using Anaconda

Heads up

Please be sure to use all of the class resources available

- We've tried to schedule the instructor/TA office hours and (informal) recitation to cover most time zones, and failing that you can make an appointment with any of us, or email us directly.
- Use the BlackBoard discussion group liberally, and don't be shy about either asking or answering questions.

Learning to program for the first time can (and should) be lots of fun, but it can also be pretty intense. You are literally learning a new language and a new way of algorithmic thinking (though, apparently, when programming you don't use the same parts of your brain as you use for human language).

If you are finding keeping up in class or keeping up with the homework is taking more time than you think is reasonable please reach out to us. Everyone progresses at a different pace and the point of this class is to enable you to employ Python and associated tools in your own research and future career, so it's worth taking our time.

Class materials:

- All class materials including assignments, grades etc. will be in [Black Board (https://blackboard.stonybrook.edu/webapps/login/)

For the next three weeks, as we learn and start to use Python, you **need** to be

- reading ahead
- playing/practicing with code
- reading the documentation
- exploring online resources
- interacting with each other (team programming is fun)
- asking lots of questions in class, in office hours, on the discusison board, and of each other

Installing Anaconda

- Go to the Anaconda download page
    - Pick the right version for your operating system if it is not automatically correctly selected.
    - You do not need to sign in or make an account (unless you want to)
- Download and then install it --- you should **not** need administrator (or root) access
    - More detail on installation is here
- Make a new folder (somewhere sensible you can find again!) to hold your notebooks

We will be using the Jupyter notebook for most of class, but there are lots and lots of other things that Anacoda provides. To see (nearly) everything, start Anacoda Navigator

- Getting started instructions

You can start the notebook from within the navigator, but you can also start it more directly

- Under Linux or MacOS, open a terminal, change into the folder (or a parent of it) with your notebooks, then type the command `jupyter notebook`
- Under Windows, under the `Start` menu follow `All programs` , `Anaconda (32 or 64 bit)` , `Jupyter Notebook`

Python Documentation

- Anaconda now provides nearly all the documentation you need under its `Learning` tab, but it is valuable to find the resources you need more directly.
- https://docs.python.org/3/index.html
- Key documents are the
    - Tutorial: https://docs.python.org/3/tutorial/
    - Language reference: https://docs.python.org/3/reference/index.html
    - Library reference: https://docs.python.org/3/library/index.html

Beginning the python tutorial

Go to https://docs.python.org/3/tutorial/

We will start at section 3 and will use the below to augment the presentation in the tutorial.

```
In [ ]:
```

Explore printing *'Hello world!'* and other things in different ways

```
In [ ]:
```

Explore using Python as a simple calculator, and comments

```
In [ ]:
```

Creating, saving, renaming and reverting notebooks

?, help function, tab word completion

Numerical data types: integers and floats (also complex, decimal, rational, ...)

In [ ]:

Variables --- names you associate with values you want to remember for future use

simple examples for you to code (from https://www.programiz.com/python-programming/examples)

- area of a triangle (`base*height/2`)
- area (`pi*r*r`) and circumference (`2*pi*r`) of a circle
- convert units `celsius = (fahrenheit - 32)/1.8 or f = c*1.8+32`
- what happens when you refer to a variable that does not exist
    - what might be common causes of this?
- what happens when you do `1/0` ? or `1.0/0.0 or 10.0**1000.0` ?
- what happens when you do `10**1000` ?

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Operators, precedence, and order of evaluation:

What operations are there? Go look in the language reference:

- if you haven't bookmarked the link just google for python language reference, select 6.5 power operator, also relevant 6.6 thru 6.11)
- evaluation order and operator precedence discussed in 6.16 and 6.17 , respectively

Examples:

- evaluating a quintic polynomial using `**`
- manually computing min, max, mean and std-var of 3 numbers (1.0, 1.5, 2.3), and evaluate -1.0**0.5

In [ ]:

In [ ]:

In [ ]:

Strings --- interacting with data and humans!

- how to specify a string? `""` , `''` , `""" """"` , `''' '''` , quoting, raw
- printing

Operators on strings

- `+` and `*` operators on strings

Examples:

- given a variable `name` (that presumably contains someone's name) write 2 (or more) ways to print out `"The winner is <name>!"` (with the actual name inserted where `<name>` appears)
- print `"hello "` 100 times

```
In [ ]:
```

Indexing and slicing into strings

Examples:

- set the variable `x` to `"The winner is Mary Jennings!"`
- what is the value of `x[0]` ?
- what is the value of `x[5]` ?
- what is the value `x[27]` ?
- what about `x[-1]` or `x[-2]` ?
- what about `x[28]` or `x[1000]` ?
- print out the winner's initials
- print out the winner's full name
- print out the winner's full name reversed
- How many characters are in the string?
- What happens when you do `x[2]='a'` ?
    - strings are **immutable** --- you cannot change them --- you have to make a new string if you want to make a change
- Using slices of `x` and the `+` operation make a new string with Fernando Perez as the winner instead of Mary Jennings
    - Fernando is super cool and you are using something that he created!

```
In [ ]:
```

Lists --- we need lists/arrays of data other than characters!

- operations on a list --- a lot like strings --- indeed all sequences are very similar in Python by design
- Where to find the documentation on strings, lists, etc.? It is in the standard library reference under sequence types. Go look!
- Lists are **mutable** so that they can be efficiently grown and operated on (working with millions of things is easy and routine)

Examples:

- make a list with `'a'` in it 10 times
- make a list with `'a'` in it 100 times --- lesson here! U don't want to print big things!
    - avoid printing in Juypyter (iPython) by assigning to a variable or appending a semicolon to the statement
- assign to variable `y` a list with `'a'` in it 1,000,000 times

    - you've finally done something you could not by hand!

In [ ]:

While loops simple example --- iteration and automation are central to programming

Essence of a while loop: while not done keep working

```
initialize the condition
while (the condition is true) do (work and update the condition)
```

Example:

- print out all positive integers less than 7
    - manual solution is `print(0,1,2,3,4,5,6)`
    - but what if I had asked for integers less than 30,000 --- automation is essential
- do the above with a while loop --- logically we need to do the following

```
set n to zero
while (n is less than 7) do (print n and add 1 to n)
```

  Let's translate this into python:

  (we will see much easier ways to do simple things like this, but the point of this example is to understand iteration and associated parts of Python)

In [ ]:

Key points:

1. translate the natural-language statement of the problem (print numbers less than 7) into an algorithm
2. translate the algorithm into Python code
3. don't forget to initialize the condition
4. logical test (and other tests)
5. `:` is Python's way of indicating a compound statement
6. indentation indicates scope
7. don't forget to increment your counter! What would happen if we did not?

Examples:

- print out all positive integers the square of which is less than 100
- print out all positive integers n such that their sum is less than 100
- the fibbonaci example from the tutorial

In [ ]:

Reading before the next class

- very briefly peek at and bookmark these three valuable resources to be sure you can find them since you will be referring extensively to them

  - tutorial: https://docs.python.org/3/tutorial/index.html
  - language reference: https://docs.python.org/3/reference/index.html#reference-index
  - standard libary: https://docs.python.org/3/library/index.html
- **actually explore** the notebook documentation under help
  - you won't understand everything yet (most people never need to)
  - do the "User Interface Tour" --- it's really short
  - look at the "Keyboard Shortcuts"
  - look at the "Notebook Help"
  - look at http://jupyter.readthedocs.io/en/latest
- **actually read** jupyter notebook definitive guide
  - https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook
- **review** tutorial section 3 (we just did this)
- **actually read** tutorial section 4 (skip 4.8), section 5, and sections 8.1 and 8.2
- **reading means** actually typing in and trying to understand at least some of the code examples --- we will go through examples in class and will focus on your questions and problems
- very briefly explore one or two of these online resources --- you can come back to them in more detail later, as needed.
  - http://python.org
  - http://jupyter.readthedocs.io/en/latest
  - http://www.datacamp.com
  - http://python-textbok.readthedocs.io/en/1.0
  - http://python.swaroopch.com
  - http://learnpython.org
  - http://www.tutorialspoint.com/python/index.htm
  - http://anaconda.org/ijstokes/python4science/notebook
  - http://thepythonguru.com/

```
In [ ]:
```