Lecture 3

Pythons.

PSAs

- Please use the etherpad if you have questions

- Lectures may be changed at the last minute

- Feedback (especially negative) is very much appreciated.

FAQ

- Terminal color change
  - Mac: Terminal -> Preferences->Text. The window that pops up allows you to create and save a custom scheme
  - Linux: Edit -> Profile -> New profile -> colors

Topics

- Python tools and self study
- Object orientation
  - What is an object?
  - How is this different from other 'things'?
  - What are some types of objects?
- Operators
- Looping
  - What is a loop?
  - Introducing: our friend, the for loop
- Input/ouput
  - How can you handle files with Python?

Tools for learning Python

- Code Academy (www.codeacademy.com)
  - Nice interactive tutorials

- Software Carpentry
  - (software-carpentry.org)
  - Recommended lectures

Tools for learning Python

- The python interpreter
  - type 'python' at the command line
  - A Unix-like python environment will start
  - Good for learning and testing little bits of code
  - Log out with Ctrl+D

- Interpreter prompt looks like >>>
  - We'll use this notation for examples

An Introduction to Objects

- Object
  - The "nouns" of python programming

An Introduction to Objects

- Object
  - The "nouns" of python programming
  - A way of abstracting and storing data

An Introduction to Objects

- Object
  - The "nouns" of python programming
  - A way of abstracting and storing data

- An object has three attributes
  - Identity - Constant. Like a computer address.
  - Type - Constant. Defines the operations that can be performed with this object.
  - Value - Usually mutable. Defined by user.

Types of Objects

- There are many built-in types
  - We'll discuss string, integer and list today

Types of Objects

- There are many built-in types
  - We'll discuss string, integer and list today

- Types are arranged in a hierarchical manner in Python.
  - We have provided a boiled-down version of the type hierarchy in this week's cheat sheet.

Operators

- Operators are fundamentally different than objects
    - Like verbs, operators do actions to objects

Operators

+	Addition	3+4=7
-	Subtraction	3-4=1
*	Multiplication	4*3=12
/	Division	12/4 = 3
%	Modulus	4%3 = 1
**	Exponent	4**3 = 64

Operators

==	Equals	>>>3==4 False
!=	Not equals	>>>3!=4 True
>	Greater	>>>3>4 False
<	Less	>>>3<4 True
>=	Greater than or equal to	>>>3>=4 False
<=	Less than or equal to	>>>3<=4 True

Variables

- Variables store data in shorthand for quick access
    - Variables reserve space in memory to store information

- A fundamental kind of object.

- The *type* of an object changes what you can do with an object stored in a variable
        - Today, we'll talk about integers, floats and strings

Integer

- Like an integer in math: a whole number.

- The possible sizes depend on memory
  - Usually between -2147483648 and +2147483648

- For example, if you wanted to store the number of observations in your data set, 1078, you would do it like this:

  ```
  >>> num_obs = 1078
  >>> num_obs
  1078
  ```

Float

- Floats are superficially similar to integers, in that they store numbers
- But floats can store decimals

Float

- Floats are superficially similar to integers, in that they store numbers
- But floats can store decimals
- If you assign a number that contains a decimal to a variable, Python will automatically float it
  - However, if you perform an operation on an integer that turns it into a float (such as dividing 5 by 2), Python will not automatically convert to a float.

Float

- Assignments work the same way for floats as ints
    - >>> b = 5.5
    - >>> b
    - 5.5
    - >>> type(b)
    - <type 'float'>

Type conversion

- Types determine behavior!

  ```
  >>> a = 5
  >>> b = 2
  >>> c = a/b
  >>> c
  2
  ```

  - Whaaaaa?

Type conversion

- Types determine behavior!

  ```
  >>> a = 5
  >>> b = 2
  >>> c = a/b
  >>> c
  2
  ```

  - Whaaaaa?

- Python will not automatically convert between types

Type Conversion

- Luckily, it's easy to convert between types
        - >>> a = float(a)
        - >>> a
        - 5.0
        - >>> a/b
        - 2.5
- To do type conversion, simply put the type of variable to which you'd like to convert in front of the number
    - float(a)
    - int(b)

String

- A string is a series of characters
  - They are ***ordered***.
  - They are ***immutable***.

String

- A string is a series of characters
  - They are ***ordered***.
  - They are ***immutable***.
- Strings are declared with quotes
  - Can be single or double, but be consistent
  - Example: You need to store a sequence of DNA bases in memory.
    - >>> seq1 = 'agatcagtcatgact'
    - >>> seq1
    - 'agatcagtcatgact'
    - >>> seq1 = ' agatcagtcatgact '

String

- A string is a series of characters
  - They are ***ordered***.
  - They are ***immutable***.
- Strings are declared with quotes
  - Can be single or double, but be consistent
  - Example: You need to store a sequence of DNA bases in memory.
    - >>> seq1 = 'agatcagtcatgact'
    - >>> seq1
    - 'agatcagtcatgact'
    - >>> seq1 = ' agatcagtcatgact '
- Concatenate strings with "+" operator
  - >>> new_seq = seq1 + 'acatg'

Strings

- When do we use strings?
  - Very common in DNA sequence analysis

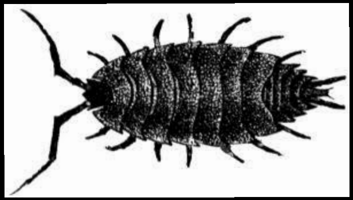
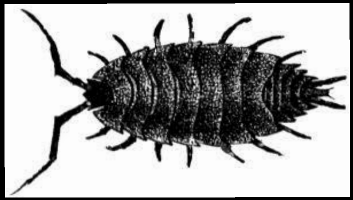
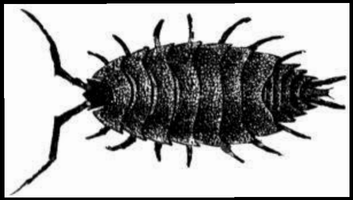
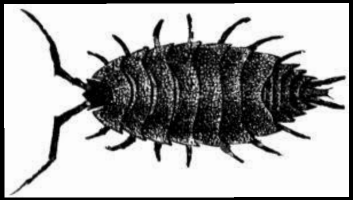
Common String Methods

- **Methods** are special procedures associated with types of objects
    - The object's type will determine the methods available to handle the object
    - Python has a simple method notation

- One common string method: .upper()
    - >>> seq1 = 'agatca'
    - >>>seq1.upper()
    - >>>seq1
    - 'AGATCA'

A slight digression: White space

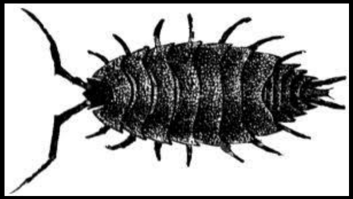
- White space refers to the space between words and characters
  - In python, white space is generally not important
  - But there are two main things to be aware of:

A slight digression: White space

- White space refers to the space between words and characters
  - In python, white space is generally not important
  - But there are two main things to be aware of:

1. Whitespace characters may be hidden in your text, but they're there
   a. Common whitespace characters:
      \t, \s, \n, \r



2. Whitespace matters for indented code
   a. More on this later...

Common String Methods

- Strip Methods - almost as fun as they sound
  - Remove whitespace from the ends and/or beginning of a string
    - >>> seq1 = ' agatcagtcatgact '
    - >>>seq1.strip()
    - 'agatcagtcatgact'
  - seq1.lstrip(), seq1.rstrip() - left and right strip.
    - Strips leading and trailing characters
  - seq1.strip('\n') - You'll use this every day.
    - Will only strip newline characters
    - Other characters can also be used

Common String Methods

- Split
  - Returns a list of words (or other items) in the string. These words (or items) are usually separated by white space
  - >>> names = "Ben April"
  - >>> nameslist = names.split()
  - >>> nameslist
  - ['Ben','April']

- 'nameslist' is a new kind of type, called a list.

Lists

- A list is an ordered, mutable grouping of objects
  - The list itself is mutable. You can add, remove and reorder the list

- Lists are declared by square brackets
  - Contained objects can be (almost?) anything
  - Objects are delimited by commas
  - `>>> list1 = [1,2.0,"three"]`

Lists

- Some terminology...

- Objects can be *declared* and then *populated*.
    - >>> list1 = []   #declaration
    - >>> list 1
    - []

Lists

- Lists are mutable
  - Need to add something?
    - >>> list2 = [] #declaration
    - >>> list2.append('eagle') #population
    - >>> list2
    - ['eagle']

Lists

- Lists are mutable
  - Need to add something?
    - >>> list2 = [] #declaration
    - >>> list2.append('coyote') #population
    - >>> list2
    - ['coyote']
  - Need to remove something?
    - >>>list2.remove('eagle')
    - >>>list2
    - ['eagle']

Lists

- Great for many things
  - But possibly the best thing about lists is using them as a tool for *iteration*, our next subject

Looping

- In its most basic form, the act of doing a task many times

Looping

- In its most basic form, the act of doing a task many times

- Loops, along with other statements we'll cover, give your program *control flow*

The "for" loop

- For loops interact really nicely with lists
  - So we'll start our discussion of looping here!

The "for" loop

- For loops interact really nicely with lists
    - So we'll start our discussion of looping here!
- For loops are used to perform a task *n* times.
    - General format

      ```
      for item in collection:
          do something with item
      ```

    - Loop will execute each statement in the indented block from top to bottom

The "for" loop

- For loops interact really nicely with lists
  - So we'll start our discussion of looping here!
- For loops are used to perform a task *n* times.

  ```
  >>> list1 = ['possum','raccoon','bobcat','eagle']
  >>> for x in list1:
  ...       print x
  possum
  raccoon
  bobcat
  eagle
  ```

The "for" loop

```
>>> for x in list1:
...       print x
```

- What are the features of this loop?
  - Two variables.
    - X: declared automatically.
    - list: container over which the loop operates. *Python knows how big list1 is!*
  - A colon. This ends the conditional.
  - An indented second line. Indentation must be the same within the whole body of the loop

Input/Output

- You don't always want to type input into the terminal.

Input/Output

- You don't always want to type input into the terminal.
- Instead, you might have a data file that you would like to open and use as input
  - Is the whole file the input?
  - Do you want to read some next-gen data line-by-line?

Input

- open() is one of the most common ways of doing this
    - f = open('filename', 'mode')
    - the 'filename' will be the file you want to open
    - 'mode' will be what you would like to do with this file
        - r for read will be assumed if no mode is provided
        - Read-only means you cannot write to the file
        - w will allow you to write to the file
        - r+ will allow reading and writing
        - Default is 'r'

Input example

- I have some data in a file. I'd like to open it, read it and write some lines to it, as well
        - >>> f = open('myfile.txt' , 'r+')
    - f is now a file object
    - This simply opens the file in a way that will allow reading and writing

Input

- Now what?
  - >>> f.read()
    - Returns your whole file as one big string. It will not be nicely formatted and will show whitespace characters.
  - >>> f.readlines()
    - This will create a list of all the lines in a file

Input

- Now what?
    - - >>> f.read()
    - This will show your whole file. It will not be nicely formatted, but will show characters, such as end-of-line characters
        - f.readlines()
    - This will create a list of all the lines in a file
    - Or, you can do a little looping
        - >>> for line in f:
        - ...   print line
    - Capture these to variables
        - >>> myfile = file.read()
        - >>> myfilelist = file.readlines()

Output

- Pretty similar to input!
    - But you need different permissions...
    - >>> outfile = open('outfile.txt','w') #writing permission
    - >>> outfile.write(my_data_object)
    - >>> outfile.close()

Exercise

- We've provided a data file
- Open this file
- Make a list where each entry is a line
    - Using a for loop
    - Using .readlines()
- Print the list
- Add one to each item in the list

- Challenges to think about:
    - Are there any white space or other difficult characters?
    - Are the characters the right type to do addition?