Lecture 2

Pythons.

Today

- Quick Homework review
- Basic Python
  - Lots of terminology - keep that pen out!
  - The interpreter
  - Operators, statements, and variables
  - Integers, Floats, String and Lists
- Control flow with loops
  - for and while
- Terminology and Philosophy
  - Objects and Types
- Python tools and self study

Today

- 3 concepts to know by 3:00pm
  - Control Flow
    - Three kinds: sequential, selection, repetition
  - Objects
    - “Object oriented language.”
    - What objects are
    - Why you should care
  - Types
    - Type hierarchy
    - What types are
    - Why you should care

Homework

- What we received looked good.
- These two are equivalent:

```
cat tree1/tree1.txt tree2...etc.
cat tree*/tree*.txt >> all_trees
```

Note on self-study

- Most learning will be done on your own
  - This class will not be enough
  - Some literature is listed at the end
  - No one learns without trial and error (and error)
  - This means you must be proactive and program a lot

- Force yourself to use Unix and Python
  - Do things you could do more quickly by hand
  - At “some point” it becomes fun, I promise.

Note on in-class time

- Don’t feel like you have to “type along”
  - We recommend note taking during lecture
  - There will be plenty of time to practice

Note on in-class time

- Don’t feel like you have to “type along”
  - We recommend note taking during lecture
  - There will be plenty of time to practice

- Why not type along?

Note on in-class time

- Don’t feel like you have to “type along”
  - We recommend note taking during lecture
  - There will be plenty of time to practice

- Why not type along?
  - Examples will speed by
  - You know how to type - that’s not why we’re here
  - Easy to forget what you have typed and why

Quick note on programming

- Most of you have articulated why you want to learn

- `ls` example

Quick note on programming

- Most of you have articulated why you want to learn

- `ls` example
  - Programming turns a large series of commands into just one
  - The program is now as infallible as your code is
  - It is repeatable and documented
  - Python is an excellent multi-purpose language
    - Huge and growing documentation
    - Easy on the eyes

Basic 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

Basic Python

- The obligatory “hello world”

```
>>> print “hello world”
hello world
```

Basic Python

- The obligatory “hello world”

Statement **Data**

```
>>> print “hello world”
hello world
```

Basic Python

- The obligatory “hello world”

Statement **Data**

```
>>> print “hello world”
hello world
```

- Statement: “do something”, call a procedure on your data

Operators

- Operators also do something to data

Operators

- Operators also do something to data

+	Addition	3+4
-	Subtraction	3-4
*	Multiplication	4*3
/	Division	5/2 # -> 2
%	Modulus	4%3 # -> 1
**	Exponent	4**3

Operators (Logical)

==	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
  - Binds a *name* to *data*
  - Assign a value to a variable with =
    - Note that only one ‘=’ is used

```
>>> a = 1
>>> a
1
```

Variables

- Variables store data
  - Binds a *name* to *data*
  - Assign a value to a variable with =
    - Note that only one ‘=’ is used

Name **Value**

```
>>> a = 1
>>> a
1
```

<-- Value is "returned"

Integer

- As in the last example, any whole number.

Integer

- As in the last example, any whole number.

- Note problem with division

```
>>> 5/2
2
```

Float

- Decimals

```
>>> 5.0/2.0
2.5
```

Float

- Decimals

```
>>> 5.0/2.0
2.5
>>> 5.0/2 #you only need one!
2.5
```

Float

- Decimals

```
>>> 5.0/2.0
2.5
>>> 5.0/2 #you only need one!
2.5
```

Comment

Syntax

- How does Python “know” what’s a float and what’s an int?
  - Syntax!

- When you type a whole number, it’s an int
  - You “declared” it to be so
  - You must take control, or face catastrophe

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

```
>>> 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

```
>>> seq1 = '1'
>>> seq1
'1'
>>> print seq1
1
```

Why no quotes?

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

```
>>> seq1, seq2 = 'atc','gta'
>>> seq1 + seq2
'atcgta'
```

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

Multiple assignment

```
>>> seq1, seq2 = 'atc','gta'
>>> seq1 + seq2
'atcgta'
```

Concatentation

Lists

- Just what they sound like
  - ***Ordered***
  - ***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

- Lists are mutable
  - Need to add something?

```
>>> list2 = [] #declaration
>>> list2.append('eagle')#population
>>> list2
['eagle']
```

Lists

- Lists are mutable
  - Need to remove something?

```
>>>list2.remove('eagle')
>>>list2
[]
```

List and string commonalities

- Both are *ordered*
  - Python knows where the elements are in each collection.
  - How do we use this information?

List and string commonalities

- Both are *ordered*
  - Python knows where the elements are in each collection.
  - How do we use this information?

- Indexing

```
>>> L = ['a','b','c']
>>> L[0] #First item is 0!
'a'
```

List and string commonalities

- Both are *ordered*
  - Python knows where the elements are in each collection.
  - How do we use this information?

- Slicing (can be tricky)

```
>>> L[1:2]#colon give range
['b'] #[inclusive:exclusive]
```

List and string commonalities

- Both are *ordered*
  - Python knows where the elements are in each collection.
  - How do we use this information?

- Slicing (can be tricky)

```
>>> L[1:2]#colon give range
['b'] #[inclusive:exclusive]
>>> L[0:3:2] #[from:to:step]
['a','c']
```

List and string commonalities

- Both are *ordered*
  - Python knows where the elements are in each collection.
  - How do we use this information?

- Slicing (can be tricky) 2

```
>>> L[-1] #negative indexing!
'c'
>>> L[-3:-1]
['a', 'b'] #still exclusive
```

List and string commonalities

- Both are *ordered*
  - Python knows where the elements are in each collection.
  - How do we use this information?

- Slicing (can be tricky) 3

```
>>> L[:] #everything
['a', 'b', 'c']
>>> L[:3]
['a', 'b', 'c'] #inclusive! aaaah!
```

Control Flow

- Three types
  - Sequential
  - Selective
  - Repetitive

Control Flow

- Three types
  - Sequential (Do something top to bottom)
  - Selective (Do something if...)
  - Repetitive (Do something many times)

Control Flow

- Three types
  - Sequential (Do something top to bottom)
  - Selective (Do something if...)
  - Repetitive (Do something many times)

  - Sequential (Default)
  - Selective (`If/else` clause)
  - Repetitive (Loops - `for and while`)

Control Flow

- Three types
  - Sequential (Do something top to bottom)
  - Selective (Do something if...)
  - Repetitive (Do something many times)

- Sequential (Default)
- Selective (`If/else` clause)
- Repetitive (Loops - **`for`** `and while`)

The `for` loop

- General format

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

- Loop will execute each statement in the indented block from top to bottom until the end of the collection is reached.

The `for` loop

- What's a collection?
  - Strings and Lists are collections

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

The `for` loop

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

- Additional features of this loop
  - Two variables.
    - Easy to understand: `list1`
    - Hard to understand: `x`
      - Declared automatically, name doesn't matter (except for normal naming conventions)
  - An indented second line.
    - Must be indented manually (use `tab`)
    - Indentation must be the same within the whole body of the loop

Practice (please work in pairs)

1. Declare a list of integers 1 - 5
   a. Name it “+”
      i. What happens? Why is this a good idea?
   b. Now name it “1”
      i. Read the answer section here (later) http://stackoverflow.com/questions/18716564/python-cant-assign-to-literal
   c. Now give the list an actual name
   d. Remove the even numbers, then add them back
2. Declare an empty list
   a. Write a `for` loop that creates a new list where each element corresponds to 1 + the matching element in your first list
   b. Find the code that makes a new list the same as the

An Introduction to Objects

- Object
  - A way of abstracting and storing data
  - What’s an object in Python?

An Introduction to Objects

- Object
  - A way of abstracting and storing data
  - What's an object in Python?
  - Pretty much everything

An Introduction to Objects

- Object
  - A way of abstracting and storing data
  - What's an object in Python?
  - Pretty much everything

- An object has three attributes
  - Identity - Constant, once it's stored in a variable.
  - 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've discussed String, Integer, Float, and List.

Types of Objects

- There are many built-in types
  - We've discussed String, Integer, Float, and List.

- 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.

Types of Objects

- There are many built-in types
  - We've discussed String, Integer, Float, and List.

- 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.

- Why care?
  - An object’s type determines what you can do with it
  - This will instantly clarify syntax x100

Types of Objects - object *behavior*

```
>>> a = 1          INTs
>>> a
1

>>> a = '1'        STRs
>>> a
'1'
```

Types of Objects - object *behavior*

```
>>> a,b = 1,2        INTs
>>> a+b
3

>>> a,b = '1','2'   STRs
>>> a+b
'12'
```

Types of Objects - object *methods*

```
>>> a = 'a'
>>> a.upper()
'A'

>>> a = 1
>>> a.upper()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no
attribute 'upper'
```

Types of Objects - object *methods*

```
>>> a = 'a'
>>> a.upper()   #Methods accessed by dot
'A'             #notation called on the
                #variable, which is an
                #object instance
```

Types of Objects - object *methods*

```
>>> a = 'a'
>>> a.upper()
'A'

>>> a = 1
>>> a.upper() #Always read your tracebacks!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no
attribute 'upper'
```

Types of Objects

- How do I learn an object's type
  - type type()

```
>>> a,b = 1,'b'
>>> type(a)
<type 'int'>
>>> type(b)
<type 'str'>
```

Types of Objects

- How do I learn an object's type
  - type type()

```
>>> type(type(a))
<type 'type'>
```

- Turtles all the way down.

Types of Objects

- Turtles all the way down.
  - This is what it means for Python to be object oriented
  - It has lots and lots of objects built in

- Pros and Cons
  - Pro: you don't have to design your own object
    - you have to in C
  - Con: you have to learn a bunch of Python objects
    - These range in complexity from integers, to custom packages for almost any kind of data.

Types of Objects

- Type conversion
  - `str()`
  - `list()`
  - `int()`
  - `float()`

```
>>> a = '1'
>>> int(a)
1
```

Homework

- Read the Type Hierarchy
- Read Wk2 cheat sheet
- Learn additional string methods
  - *str*.strip()
  - *str*.split()
  - *str*.join()
  - *str*.rjust()
  - *str*.ljust()
- And a quick excercise

Next Time

- Selective control flow (`if/else`)
- File input and output
- More types
  - Dictionaries
  - Files (streams)
- Nested Statements
- Comprehensions

Tools for learning Python

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

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

Important string methods

str.strip([*chars*])
 	
#remove characters, default: whitespace
	`>>> a = ' a '`
	`>>> a.strip()`
	`'a'`

str.split(*sep*)
 	
#returns list, with elements separated by *sep*. Default: whitespace
	`>>> a = 'a b'`
	`>>> a.split()`
	`['a', 'b']`

More Concatenation

+ (string)	“atc” + “gta”	“atcgta”
+ (list)	>>> a = ['a'] >>> b = ['b']	>>> a+b ['a', 'b']
+= (string)	>>> a = 'atc' >>> a += 'gta'	>>> a 'atcgta'
+= (list)	>>> a = ['a'] >>> a += ['b']	>>> a ['a', 'b']