Python Basics

Whitespace matters! Your code will not run correctly if you use improper indentation.
`#this is a comment`

Basic Python Logic

if:
```
  if test:
    #do stuff if test is true
  elif test 2:
    #do stuff if test2 is true
  else:
    #do stuff if both tests are false
```

while:
```
 while test:
   #keep doing stuff until
   #test is false
```

for:
```
   for x in aSequence:
     #do stuff for each member of aSequence
     #for example, each item in a list, each
     #character in a string, etc.

   for x in range(10):
     #do stuff 10 times (0 through 9)

   for x in range(5,10):
     #do stuff 5 times (5 through 9)
```

Python Strings

A string is a sequence of characters, usually used to store text.

creation:	the_string = “Hello World!”	
	the_string = ‘Hello World!’	
accessing:	the_string[4]	returns ‘o’
splitting:	the_string.split(‘ ‘)	returns [‘Hello’, ‘World!’]
	the_string.split(‘r”)	returns [‘Hello Wo’, ‘ld!’]

To join a list of strings together, call `join()` as a method of the string you want to separate the values in the list (`‘’` if none), and pass the list as an argument. Yes, it’s weird.

```
words = [”this”, ‘is’, ‘a’, ‘list’, ‘of’, “strings”]
```

`‘ ‘.join(words)`	returns “This is a list of strings”
`‘ZOOL’.join(words)`	returns “ThisZOOLisZOOLaZOOLlistZOOLofZOOLstrings”
`‘’.join(words)`	returns “Thisisalistofstrings”

String Formatting: similar to `printf()` in C, uses the `%` operator to add elements of a tuple into a string

```
this_string = “there”
print “Hello %s!”%this_string
```
returns “Hello there!”

Python Tuples

A tuple consists of a number of values separated by commas. They are useful for ordered pairs and returning several values from a function.

creation:
```
emptyTuple = ()
singleItemTuple = (“spam”,)
thistuple = 12, 89, ‘a’
thistuple = (12, 89, ‘a’)
```
← note the comma!

accessing: `thistuple[0]` returns 12

Python Dictionaries

A dictionary is a set of key:value pairs. All keys in a dictionary must be unique.

creation:	`emptyDict = {}`	
	`thisdict = {‘a’:1, ‘b’:23, ‘c’:”eggs”}`	
accessing:	`thisdict[‘a’]`	returns 1
deleting:	`del thisdict[‘b’]`	
finding:	`thisdict.has_key(‘e’)`	returns False
	`thisdict.keys()`	returns [‘a’, ‘c’]
	`thisdict.items()`	returns [(‘a’, 1), (‘c’, ‘eggs’)]
	`‘c’ in thisdict`	returns True
	`‘paradimethylaminobenzaldehyde’ in thisdict`	returns False

Python List Manipulation

One of the most important data structures in Python is the list. Lists are very flexible and have many built-in control functions.

creation:	`thelist = [5,3,‘p’,9,‘e’]`		`[5,3,’p’,9,’e’]`
accessing:	`thelist[0]`	returns 5	`[5,3,’p’,9,’e’]`
slicing:	`thelist[1:3]`	returns [3, ‘p’]	`[5,3,’p’,9,’e’]`
	`thelist[2:]`	returns [‘p’, 9, ‘e’]	`[5,3,’p’,9,’e’]`
	`thelist[:2]`	returns [5, 3]	`[5,3,’p’,9,’e’]`
	`thelist[2:-1]`	returns [‘p’, 9]	`[5,3,’p’,9,’e’]`
length:	`len(thelist)`	returns 5	`[5,3,’p’,9,’e’]`
sort:	`thelist.sort()`	no return value	`[3,5,9,’e’,’p’]`
add:	`thelist.append(37)`		`[3,5,9,’e’,’p’,37]`
return &	`thelist.pop()`	returns 37	`[3,5,9,’e’,’p’]`
remove:	`thelist.pop(1)`	returns 5	`[3,9,’e’,’p’]`
insert:	`thelist.insert(2, ‘z’)`		`[3,’z’,9,’e’,’p’]`
remove:	`thelist.remove(‘e’)`		`[3,’z’,9,’p’]`
	`del thelist[0]`		`[‘z’,9,’p’]`
concatenation:	`thelist + [0]`	returns [‘z’,9,’p’,0]	`[‘z’,9,’p’]`
finding:	`9 in thelist`	returns True	`[‘z’,9,’p’]`

List Comprehension

A special expression enclosed in square brackets that returns a new list. The expression is of the form:
`[`*`expression`*` for `*`expr`*` in `*`sequence`*` if `*`condition`*`]`   The condition is optional.

```
>>>[x*5 for x in range(5)]
[0, 5, 10, 15, 20]
>>>[x for x in range(5) if x%2 == 0]
[0, 2, 4]
```

Python Class and Function Definition

function:
```
def myFunc(param1, param2):
    """By putting this initial sentence in triple quotes, you can
    access it by calling myFunc.__doc___"""
    #indented code block goes here
    spam = param1 + param2
    return spam
```

class:
```
class Eggs(ClassWeAreOptionallyInheriting):
     def __init__(self):
       ClassWeAreOptionallyInheriting.__init__(self)
       #initialization (constructor) code goes here
       self.cookingStyle = 'scrambled'
     def anotherFunction(self, argument):
       if argument == "just contradiction":
          return False
       else:
          return True

theseEggsInMyProgram = Eggs()
```

Files

open:

`thisfile = open("datadirectory/file.txt")` note: forward slash, unlike Windows! This function defaults to read-only

accessing:

`thisfile.read()`	reads entire file into one string
`thisfile.readline()`	reads one line of a file
`thisfile.readlines()`	reads entire file into a list of strings, one per line
`for eachline in thisfile:`	steps through lines in a file