# Python Basics

Whitespace matters! Your code will not run correctly if you use improper indentation.

```
      # this is a comment
```

## 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",)         # note the comma!
            thistuple = 12, 89, 'a'
            thistuple = (12, 89, 'a')

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
            ‘thisisnotthere’ 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.

| Operation | Syntax | Return | New Value |
|---|---|---|---|
| Create | `thelist = [5,3,‘p’,9,‘e’]` | No return value | `[5,3,‘p’,9,‘e’]` |
| Accessing | `thelist[0]` | `5` | Unchanged |
| Slicing | `thelist[1:3]`<br>`thelist[2:]`<br>`thelist[:2]`<br>`thelist[2:-1]` | `[3, ‘p’]`<br>`[‘p’, 9, ‘e’]`<br>`[5, 3]`<br>`[‘p’, 9]` | Unchanged<br>Unchanged<br>Unchanged<br>Unchanged |
| Length | `len(thelist)` | `5` | Unchanged |
| Sort | `thelist.sort()` | No return value | `[3,5,9,’e’,’p’]` |
| Add | `thelist.append(37)` | No return value | `[3,5,9,’e’,’p’,37]` |
| Return and Remove | `thelist.pop()`<br>`thelist.pop(1)` | `37`<br>`5` | `[3,’z’,9,’p’]`<br>`[‘z’,9,’p’]` |
| Insert | `thelist.insert(2, ‘z’)` | No return value | `[3,’z’,9,’e’,’p’]` |
| Remove | `thelist.remove(‘e’)`<br>`del thelist[0]` | No return value<br>No return value | `[3,’z’,9,’p’]`<br>`[’z’,9,’p’]` |
| Concatenate | `thelist + [0]` | `[’z’,9,’p’,0]` | `[’z’,9,’p’]` |
| Finding | `9 in thelist` | `True` | Unchanged |

## 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 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
```