# python™

2/28/2008

# >>> Overview

- Arrays in Python – a.k.a. Lists
- Ranges are Lists
- Strings vs. Lists
- Tuples vs. Lists
- Map-Reduce
- Lambda
- Review: Printing to a file

# >>> Arrays in Python

Python has a data type known as a list. For our purposes, lists are arrays.

**Declaration syntax:** <name> = [<value>, <value>, <value>, ..., <value>]
<name> = [<default value>] * <initial array size>

**Example:** numbers = [12, 49, -2, 26, 5, 17, -6]
zeros = [0] * 10

**Indexing:** Lists have zero based indexing from front
**Negative Indexing:** You can also refer to an element by a negative index representing how far it is from the end.

**Example:** index from front *0 1 2 3 4 5*
numbers = [ 13, 25, 39, 46, 54, 68]
index from back *-6 -5 -4 -3 -2 -1*

python™

# >>> Methods for Lists

**Basic Methods** – directly modify the lists

- list.append(item) – appends the item to the end of the list
- list.insert(index, item) – inserts the item at the specified index
- list.remove(item) – removes the first occurrence of item from the list
- list.extend(second_list) – appends second_list to the list

**Mathematical Operators** – behave as you would expect them to

- (+) Returns a new list by adding two lists. Appends the right-hand list to the left-hand list.
- (+=) Appends the right-hand list to the left-hand list. Modifies left list. Acts like extend()
- (*) Multiplies a list and an integer "n". Returns a new list that has n-1 versions of original list appended to it

**Examples:**

list = [34, 21, 29, 86, 29]

list.append(3) => [34, 21, 29, 86, 29, 3]
list.insert(2, 3) => [34, 21, 3, 29, 86]

list2 = [1, 2, 3, 4]

list.remove(29) => [34, 21, 86, 29]
list.extend(list2) => [34, 21, 86, 29, 1, 2, 3, 4]
[0] * 5 => [0, 0, 0, 0, 0]

# >>> More Methods

**More Methods**

- list.count(element) – returns number of times element occurs in the list
- list.sort – sorts the element in place
- list.reverse – reverses the element in place

**Slicing** – can get a sub list of a list

<name>[<first index inclusive> : <second index not-inclusive>]

```
list = [4, 23, 16, 7, 29, 56, 81]
list[3:6] => [16, 7, 29]
```

**Length of lists**

```
len(list) => 7
```

**Split** – returns a list

“lets try some splitting here”.split(“ “)  =>  ['lets', 'try', 'some', 'splitting', 'here']

# >>> Printing Lists

There are two ways to print lists.

```
list1 = [“elements”, “of”, “our”, “list”]
list2 = [21, 29, 86, 19, 42]
```

**String concatenation and type conversion:**

```
print “This list is ” + str(list1)        =>  This list is [“elements”, “of”, “our”, “list”]
print “This list is ” + str(list2)        =>  This list is [21, 29, 86, 19, 42]
```

**Comma separated arguments in the print method:**

```
print “This list is”, list1               =>  This list is [“elements”, “of”, “our”, “list”]
print “This list is”, list2               =>  This list is [21, 29, 86, 19, 42]
```

# >>> Ranges are Lists

Recall how we used the method range() in for loops.
Calling range returns a list with the patterns specified by range().

**Example:**

```
range(5)          =>    [0, 1, 2, 3, 4]
range(0, 10, 2)   =>    [0, 2, 4, 6, 8]
```

Using a for loop iterates over each element in a list.

**Example:**

```
list = [3, 6, 5, 7, 15]
for i in list:
    print i
```

**Example 2:**

```
list = [3, 6, 5, 7, 15]
for i in range(len(list))
        list[i] = list[i] + 1
```

python™

# >>> Strings vs. Lists

Although Strings are different from lists, Strings can be accessed like lists.

**Example:**

| | |
|---|---|
| s = “Hello!” | |
| s[1] | => ‘e’ |
| s[-1] | => ‘!’ |
| s[1:5] | => “ello” |
| s.count(“l”) | => 2 |

Once a String has been created, it cannot be changed.
Methods that alter a list cannot be called on Strings.

**Note:** Python does not distinguish between characters and strings.
Characters are just Strings of length 1.

# >>> Tuples vs. Lists

Additionally, tuples can be accessed like lists. However, tuples are not list.
Tuples, like strings cannot be changed once they have been created.

**Example:**

```
s = (123, 456, 789, 246, 357)
s[1]            => 456
s[-1]           => 357
s[1:4]          => (456, 789, 246)
```

python™

# >>> Random with lists

```
>>> from random import *
>>> randint(0,9)
1
>>> randint(0,9)
4
>>> choice(range(10))
7
```

random.**randint**(a,b)

returns an int between a and b inclusive

random.**choice**(seq)

returns a random element of the sequence

python™

# >>> Review - Files

Opening files:
open(filename) ~ defaults to read
open(filename, "r") ~ specifies read
open(filename, "w") ~ writes to this file

File objects: (we won't really have to use these)
* .readlines() ~ file as a list of lines
* .read() ~ file as a string
* .readline(e) ~ next line as string

**imdb.py**

```
filename = "imdb.txt“

f1 = open(filename)
for line in f1:
print line.upper()
f1.close()

f2 = open(filename, “w”)
f2.write(“This will over write the file \n”)
f2.close()
```

python™

# >>> Sections Example in Python

Let's solve the Sections problem.  We want to take the following line from a file:

111111010111111101001110110110110001110……

And turn it into:

Sections attended: [9, 6, 7, 4, 3]
Sections scores: [20, 18, 20, 12, 9]
Sections grades: [100.0, 90.0, 100.0, 60.0, 45.0]

Sections attended: [6, 7, 5, 6, 4]
Sections scores: [18, 20, 15, 18, 12]
Sections grades: [90.0, 100.0, 75.0, 90.0, 60.0]

Sections attended: [5, 6, 5, 7, 6]
Sections scores: [15, 18, 15, 20, 18]
Sections grades: [75.0, 90.0, 75.0, 100.0, 90.0]

# >>> Map-Reduce: Map

Python supports functional programming.

Functional programming differs from what we have been doing, by treating programming as the evaluation of a series of mathematical functions.  Map() and reduce() are functional language methods.  They return a new list instead of modifying the one passed.

Map – takes a function and a list and applies the function to each individual element in the list.

```
def add_one(x)
        return x + 1

list = [0, 2, 4, 6, 8]
new_list = map(add_one, list)
```

Looking at our new list:

```
new_list        =>      [1, 3, 5, 7, 9]
```

python™

# >>> Map-Reduce: Reduce

Reduce – takes a function and a list and reduces the list to a single element by combining the element using the given function.

```
def multiply(x, y)
    return x * y

list = [2, 4, 6, 8, 10]
value = reduce(multiply, list)
```

Looking at our value:

value => 3840

# >>> Sections Example - Map

Let’s use the functional method map() to modify our Sections Example.

# >>> Lambda

Lambda is a keyword that designates an “anonymous function”. This is a lot of terminology, but lets see how we can use it.

Instead of defining a method, and then applying it using map():

```
def add_one(x)
    return x + 1

list = [0, 2, 4, 6, 8]
map(add_one, list)
```

We can do it all in one line using lambda and anonymous functions:

```
map(lambda x : x + 1, list)
```

Lets use lambda to further simplify our Sections Example.

python™