Python Lists

What is Not a “Collection”?

Most of our variables have one value in them - when we put a new value in the variable, the old value is overwritten

```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```

A List is a Kind of Collection

- A collection allows us to put many values in a single “variable”
- A collection is nice because we can carry all many values in one convenient package.

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]

carryon = [ 'socks', 'shirt', 'perfume' ]
```

List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow', 'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

We Already Use Lists!

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
```

```
5
4
3
2
1
Blastoff!
```

Looking Inside Lists

- List items are ordered, changeable, and allow duplicate values.
- Just like strings, we can get at any single element in a list using an index specified in square brackets
- Negative indexing possible
- Since lists are indexed, lists can have items with the same value
- Can have different data types


```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print(friends[1])
Glenn
>>>
```

Lists are Mutable

- Strings are “immutable” - we cannot change the contents of a string - we must make a new string to make any change

- Lists are “mutable” - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not support item assignment
>>> x = fruit.lower()
>>> print(x)
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

How Long is a List?

- The len() function takes a list as a parameter and returns the number of elements in the list
- Actually len() tells us the number of elements of any set or sequence (such as a string...)

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
>>>
```

Using the range Function

- The range function returns a list of numbers that range from zero to one less than the parameter
- We can construct an index loop using for and an integer iterator

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(list(range(len(friends))))
[0, 1, 2]
>>>
```

A Tale of Two Loops...

```
friends = ['Joseph', 'Glenn', 'Sally']

for friend in friends :
    print('Happy New Year:',  friend)
```

```
for i in range(len(friends)) :
    friend = friends[i]
    print('Happy New Year:',  friend)
```

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(list(range(len(friends))))
[0, 1, 2]
>>>
```

Happy New Year: Joseph

Happy New Year: Glenn

Happy New Year: Sally

Concatenating Lists Using +

We can create a new list by adding two existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

Replicating the elements

- The * operator

```
>>>List=[10,20]

>>>List1=2*List    // [10,20,10,20]
```

Is Something in a List?

- Python provides two operators that let you check if an item is in a list
- These are logical operators that return True or False
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

Is Operator

- Check if two lists refer to same object

```
>>>A=[‘a’,’b’,’c’]

>>>B=[‘a’,’b’,’c’]

>>>A is B

False    // two lists are identical but they don’t refer to same object
```

del Operator

- Remove the elements from a list based on their index.

```
>>>list=[10,20,30,40]

>>>del list[-1]   // [10,20,30]

>>>del list[1:3] // [10,40]
```

Lists Can Be Sliced Using :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: Just like in strings, the second number is “up to but not including”

List Methods

```
>>> x = list()
>>> type(x) // returns the type of list
<type 'list'>
>>> dir(x) //returns all properties and methods of the specified object, without the values.
[... 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

Building a List from Scratch

- We can create an empty list and then add elements using the append method
- The list stays in order and new elements are added at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

Lists are in Order

- A list can hold many items and keeps those items in the order until we do something to change the order
- A list can be sorted (i.e., change its order)
- The sort method (unlike in strings) means “sort yourself”

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort() //sort alphabetically
>>> print(friends)
['Glenn', 'Joseph', 'Sally']
>>> print(friends[1])
Joseph
>>>
```

- The sort() method sorts the list ascending by default.
- You can also make a function to decide the sorting criteria.
- Syntax: list.sort(reverse=True|False, key=myFunc)
  - Reverse : optional, if reverse=True, perform decending order sorting
  - Key: optional. A function to specify the sorting criteria

Built-in Functions and Lists

- There are a number of functions built into Python that take lists as parameters

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

Python has a set of methods you can use with lists:

- append() Adds an element at the end of the list
- extend() Add the elements of a list (or any iterable), to the end of the current list
- copy() Returns a copy of the list
- count() Returns the number of elements with the specified value
- index() Returns the index of the first element with the specified value

- insert() Adds an element at the specified position
- clear() Removes all the elements from the list
- pop() Removes the element at the specified position
- remove() Removes the first item with the specified value
- reverse() Reverses the order of the list
- sort() Sorts the list

- Concatenating two lists will result in a new list object, but calling extend method will update the original list itself

- Append function in Python adds a single element to the end of the list, whereas the extend function adds multiple elements to the list.

- >>>List1.append(‘A’)// a single value or a list is added to the last as single element

- >>>List1.extend(List2)// List2 may be a single value or a list having multiple values

- >>>List1=[10,20,30]
- >>>List2=[30,40,50]
- >>>List1.append(List2) // [10,20,30,[30,40,50]]
- >>>List1.extend(List2) // [10,20,30,30,40,50]

- List Comprehensions
  - Used to create a new list from the existing sequences.
  - Tool for transforming a given list to new list.
  - Syntax:

    [<expression> for <element> in <sequence> if <conditionals>]

    Means…compute the expression for each element in a sequence, if the condition satisfies.

```
>>>List1 [10,20,30,40,50]
>>>List1
      [10,20,30,40,50]

>>>for i in range(len(List1)):
          List1[i]=List1[i]+10
>>>List1
      [20,30,40,50,60]
```

...

Using list comprehension,

```
>>>List1=[x+10 for x in List1]

>>>List1
          [20,30,40,50,60]
```

```
>>>List1=[1,2,3,4,5]

>>>List1

      [1,2,3,4,5]

>>>List1=[x for x in List1 if x%2==0]       //use of conditionals.

>>>print(List1)

[2,4]

>>>newlist = [x if x%2== 0 else "odd" for x in List1] //if…else conditionals

>>>print(newlist) //[‘odd’,2,’odd’,4]
```

- List comprehensions requires lesser code and runs faster.
- Conatins:
  - An input list
  - A variable referencing the input sequence
  - An optional expression
  - An output expression

...