2/23/19 Class 4: Control Flow (continued) and Lists

- **`while` loop**
- **`for` loop**
- **Lists**

A. While (loop)

```
while(Boolean expression):
     stuff to do until expression becomes false
```

- If the Boolean statement is true, we keep running
- Common mistake for beginners is making an infinite loop!
- Thus it's important to include some change of variable in the while loop so that eventually, the Boolean expression becomes false

E.g.)

```
x=5
while(x>3):
     print 'happy'
     x=x-1
```

Compiler will enter loop because x = 5 >3 → print 'happy' and x becomes 4
Compiler enters loop again because 4>3 → prints 'happy' and x becomes 3
Compiler evaluates 3>3 → no longer True the compiler moves on to line after loop

B. for (loop): logically equivalent to a while loop (simply a matter of convenience)

```
In: l = [4,1,'happy','munday'] # this is a list
In: for item in l:
     print item
Out:
4
1
happy
munday
```

**for vs while is largely a difference in convenience

- These are some general guidelines for those who want to differentiate their use:
    - if your problem definition involves the word UNTIL: use a while loop (keep repeating UNTIL that while loop is false)
    - If you just want to repeat it for everything in a collection or range, use a for loop

** `range` function is useful in `for` loops

```
In: for i in range(10):
          print i
```

```
Out:
0
1
2
3
4
5
6
7
8
9
```

C. LISTS: ordered set of variables (any mix of data types) or it could be an empty list

Datatype	Python class	Code	Variable	Literal
List, or a collection of objects	`list`	`l = [4, 1, 'happy', True]`	`l`	`[4, 1, 'happy', True]`

- **Access elements of a list**

```
In: l[0]
Out: 4
```

- **Modify/Set an element in a list**

```
In: l[3]='munday'
In: l
Out: [4,1,'happy','munday']

In: n=[1,2,3,4,5] #same syntax for splicing a list as a string
In: n[0:2] = [6,7]
In: n
Out: [6, 7, 3, 4, 5]
```

- **Find length**

```
In: l.len()
Out: 4
```

- **Append and Pop (end of list)**

```
In: l.append(5)
In: l
Out: [4,1,'happy', 'munday',5]
In: l.pop()   ## remove element at end of list
In: l
Out: [4,1,'happy', 'munday']
```

```
In: l.pop(1) ## you can remove at a specified index
In: l
Out: [4,'happy','munday']
```

- **Insert and Remove**

```
In: l.insert(2,"sad")   ## insert at index2
In: l
Out: [4,'happy',"sad",'munday']

In: l.remove("sad") ## For remove, you must give the element,
                    ## not index. (it will remove the first
                    ## occurrence)
In: l
Out: [4,'happy','munday']
```

- **Add two lists together**

```
In: l.extend(l2)
In: l
Out: [4,'happy','munday',4,3,'sad',1]
```

- **Find index of element (This finds the first occurrence)**

```
In: l.index(4)
Out: 0
```

- **Sort from smallest to largest**

```
In: l.sort()
In: l
Out: [1, 3, 4, 4, 'happy', 'munday', 'sad']
```

  - ## numbers come before letters in default sorting methods due to ASCII (https://en.wikipedia.org/wiki/ASCII)
  - to avoid unprecedented complications, only use this when all elements are of the same datatype

- **Copy the list**

```
In: l = [4, 1, 'happy', True]
In: l_copy = l.copy()
In: l_copy
Out: [4, 1, 'happy', True]
In: l_copy[2]='sad'
In: l_copy
Out:[4, 1, 'happy', True]
```

  - Think about how this would be different from setting `l_copy = l`

- **Reverse the list**

```
In: l.reverse()
In: l
Out: ['sad', 'munday', 'happy', 4, 4, 3, 1]
```

** FYI: All of these methods modify the list `l`, instead of creating a new list object. This is because lists are **mutable** objects (We will get to this distinction later in the course)

- the `in` operator: checks membership in a collection (lists and strings)

```
In: l = [1,2,3,4,5]
In: 5 in l
Out: True
```

- Using `for` loop to create lists = list comprehension

```
In: [x for x in range(10)]
Out:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

<Additional info: Useful methods for Strings>

Suppose `s` is an object of class `str` (in all of these methods, s itself is unchanged because it is **immutable**)

- `s.upper()` → returns new string that looks just like s with everything capitalized
- `s.lower()`
- `s.isalpha()` → returns True if the string is comprised solely of alphabets
- `s.split()` → split string into individual elements using whitespace and store in a list
- `s.rstrip(`*something*`)`: returns string with element removed from the right end of string
- `s.lstrip(`*something*`)`: same as above from left end