Python

Agenda - 2

- Sequence type Cont.:
  - List
- Looping Statements:
  - For loop
  - While loop
- **Lab 4 – List Lab**

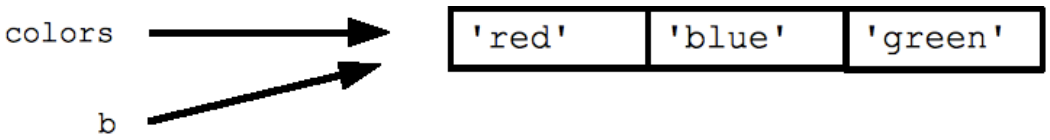
Sequence Types
Lists

- List literals are written within square brackets [ ]. Lists work similarly to strings -- use the len() function and square brackets [ ] to access data, with the first element at index 0.

```
colors = ['red', 'blue', 'green']
print colors[0]    ## red
print colors[2]    ## green
print len(colors)  ## 3
```

- Assignment with an = on lists **does not make a copy**. Instead, assignment makes the two variables point to the one list in memory.

```
b = colors    ## Does not copy the list
```



Sequence Types
Lists

- The "**empty list**" is just an empty pair of brackets [ ]. The '**+**' works to append two lists, so [1, 2] + [3, 4] yields [1, 2, 3, 4].

- To build a list a common pattern is to start an **empty list** [], then use **append**() or **extend**() to add elements to it:

```
mylist = []          ## Start as the empty list
mylist.append('a')   ## Use append() to add elements
mylist.append('b')
```

- **Slices** work on lists just as with strings, and can also be used to change sub-parts of the list.

```
mylist = ['a', 'b', 'c', 'd']
print mylist[1:-1]   ## ['b', 'c']
mylist[0:2] = 'z'    ## replace ['a', 'b'] with ['z']
print mylist         ## ['z', 'c', 'd']
```

Sequence Types
Lists

- **Lists Methods:**
  - mylist.**append**(elem) -- adds a single element to the end of the list. Common error: does not return the new list, just modifies the original.
  - mylist.**insert**(index, elem) -- inserts the element at the given index, shifting elements to the right.
  - mylist.**extend**(list2) adds the elements in list2 to the end of the list. Using + or += on a list is similar to using extend().
  - mylist.**index**(elem) -- searches for the given element from the start of the list and returns its index. Throws a ValueError if the element does not appear.

Sequence Types
Lists

- **Lists Methods:**
  - mylist.**remove**(elem) -- searches for the first instance of the given element and removes it (throws ValueError if not present)
  - mylist.**sort**() -- sorts the list in place (does not return it). (The **sorted**() function is preferred.)
  - mylist.**reverse**() -- reverses the list in place (does not return it)
  - mylist.**pop**(index) -- removes and returns the element at the given index. Returns the rightmost element if index is omitted (roughly the opposite of append()).

Sequence Types
Lists

- **Lists Methods:**

```
mylist = ['larry', 'curly', 'moe']
mylist.append('shemp')         ## append elem at end
mylist.insert(0, 'xxx')        ## insert elem at index 0
mylist.extend(['yyy', 'zzz'])  ## add list of elems at end
print mylist  ## ['xxx', 'larry', 'curly', 'moe', 'shemp', 'yyy', 'zzz']
print mylist.index('curly')    ## 2

mylist.remove('curly')         ## search and remove that element
mylist.pop(1)                  ## removes and returns 'larry'
print mylist  ## ['xxx', 'moe', 'shemp', 'yyy', 'zzz']
```

- **Common error**: note that the above methods do **not** ***return*** the modified list, they just modify the original list.

```
mylist = [1, 2, 3]
print mylist.append(4)   ## NO, does not work, append() returns None
```

Looping Statements for.....in

- Python's *for* and *in* constructs are extremely useful. The *for* construct -- **for var in list** -- is an easy way to look at each element in a list. Remember do not add or remove from the list during iteration.

```
squares = [1, 4, 9, 16]
sum = 0
for num in squares:
  sum += num
print sum  ## 30
```

- The *in* construct on its own is an easy way to test if an element appears in a list -- **value in collection** -- tests if the value is in the collection, returning True/False.

```
list = ['larry', 'curly', 'moe']
if 'curly' in list:
  print 'yay'
```

Looping Statements for…..in

- The **range**(**n**) function yields the numbers 0, 1, ... n-1, and **range**(**a**, **b**) returns a, a+1, ... b-1 -- up to but **not including** the **last number**. The combination of the for-loop and the range() function allow you to build a traditional numeric for loop:

```
## print the numbers from 0 through 99
for i in range(100):
  print i
```

- There is a variant **xrange**() which avoids the cost of building the whole list for a memory-starved machine. The advantage of the xrange type is that an xrange object will always take the **same amount of memory**, **no matter the size** of the range it represents.

```
sys.getsizeof(xrange(0,10)) #20
sys.getsizeof(xrange(0,1000)) #20
```

Check the following link to know the difference between range and xrange https://www.pythoncentral.io/how-to-use-pythons-xrange-and-range/

Looping Statements for…..in

- To determine the increment step in for loop:

```
## print the numbers from 0 through 99 with step=2
for i in range(0,100,2):
  print i,
```

```
mylist = [1,2,3,4,5,6,7,8,9,10]
for i in mylist[::2]:
  print i,  ## prints 1 3 5 7 9
```

List Comprehensions

- **List comprehensions** are a more advanced feature which is nice for some cases. A list comprehension is a compact way to write **an expression** that **expands** to a whole **list**.

- Suppose we have a list nums [1, 2, 3], here is the list comprehension to compute a list of their squares [1, 4, 9]:

```
nums = [1, 2, 3, 4]

squares = [ n * n for n in nums ]   ## [1, 4, 9, 16]
```

List Comprehensions

- The syntax is [***expr*** **for** var **in** list] -- the for var in list looks like a regular for-loop, but **without** the colon (**:**). The ***expr*** to its left is evaluated once for each element to give the values for the new list.

- Here is an example with strings, where each string is changed to **upper case** with '**!!!**' appended:

```
strs = ['hello', 'and', 'goodbye']

shouting = [ s.upper() + '!!!' for s in strs ]
## ['HELLO!!!', 'AND!!!', 'GOODBYE!!!']
```

List Comprehensions

- You can add an **if** test to the right of the for-loop to narrow the result. The if test is evaluated for each element, including only the elements where the test is true.

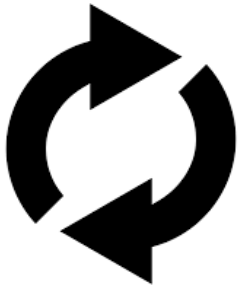
```
## Select values <= 2
nums = [2, 8, 1, 6]
small = [ n for n in nums if n <= 2 ]  ## [2, 1]

## Select fruits containing 'a', change to upper case
fruits = ['apple', 'cherry', 'bannana', 'lemon']
afruits = [ s.upper() for s in fruits if 'a' in s ]
## ['APPLE', 'BANNANA']
```

Looping Statements while loop

- Python also has the standard **while**-**loop**, and the ***break*** and ***continue*** statements.

```
## Access every 3rd element in a list
i = 0
my_str= "ABCABCABCABCABC"
while i < len(my_str):
  print my_str[i]
  i = i + 3
```



LAB – 4
LISTS LAB

Lists Lab

- Please download the lab from the following link:
  - https://drive.google.com/file/d/0B6Hf8UvSSqTXRkRURGd6dFpMaFU/view?usp=sharing
- Complete the script **list_lab.py** in **45** mins and send your solution on the following email:
  - Omar.Soliman@imtSchool.com with the following subject :
    - If you are from ITI-Smart track ES:
      - [ITI_SV_39][PY-list]yourfullname
    - If you are from ITI-Nasr City track ES:
      - [ITI_NC_39][PY-list]yourfullname

LAB – 4
LISTS LAB

What's Next ?

- Get Certified With:
  - https://www.edx.org/course/learn-program-using-python-utarlingtonx-cse1309x
  - https://www.coursera.org/course/interactivepython1
  - https://www.coursera.org/course/interactivepython2

- More Interesting References:
  - Python Cookbook, 2nd Edition
  - https://automatetheboringstuff.com/
  - http://code.activestate.com/recipes/langs/

Eng. Mohammad A.Hekal: embeddedgeek.34@gmail.com
Eng. Omar Soliman: Omar.Soliman@imtSchool.com