Lesson 1 - Thomas Deneuville

September 26, 2017

1 List Nuances

Let's go over some nuances about lists

1.1 Indexing into a list

The general syntax it the following. Assuming `l` is your list:

`m = l[begin:end:step]`

`begin` tells you where you start accessing in the list. `end` tells you where you stop but **does not include** that value (exclusive) and `step` is the increment of what you need to access starting from `begin`.

```
In [2]: l = [1, 2, 3, 4, 5, 1, 2, 3]
        #    0  1  2  3  4  5  6  7
        m = l[2:6:2] # Start from index 2, ends at index 6 (but does not include) and we acces
        # m = [3,5]
        print(m)

        # If we do not include the step, we assume that the increment is 1
        m2 = l[2:6]
        # m2 = [3,4,5,1]
        print(m2)
```

```
[3, 5]
[3, 4, 5, 1]
```

`begin` and `end` can be negative, but you must ensure that the increment is also negative. Negative indices mean that we are approaching the list in reverse order

```
In [ ]: # l = [1, 2, 3,  4, 5,  1, 2, 3]
        #    -8  -7  -6 -5  -4 -3 -2 -1
        m3 = l[-1:-6:-2]
        #m3 = [3, 1, 4]
        print(m3)
```

You can omit the `begin` and `end`. It assumes that they are `0` and the end of the list respectively

```
In [8]: # l = [1, 2, 3,  4, 5,  1, 2, 3]
        #      0  1  2   3  4   5  6  7
        m4 = l[:5] # specifying the begin is omitted which means that it is automatically star
        print(m4)
        m5  = l[5:] # begin is 5, but specifying the end is omitted, so automatically go to th
        print(m5)
        m6 = l[::2] # begin is 0, end is the end of the list, increment is 2 - skips every oth
        print(m6)
        m7 = l[::-1] # A trick to go in reverse order
        print(m7)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3]
[1, 3, 5, 2]
[3, 2, 1, 5, 4, 3, 2, 1]
```

2 Inserting elements in a list

We can insert elements in a list at any position we want. There are two methods to allow us to insert. One method is called `append`. Adds an element to the end of the list. You can use the `insert` method to insert values in the list and shift everything to the right. New element goes where you specified where to insert.

```
In [11]: A = [1, 'string', [0, 1, 2]]
         print(A)

         # We can add another element to this list at the end
         A.append('hello')
         print(A)

         # We can also insert in any arbitrary position
         A.insert(2, 'world')
         print(A)
```

```
[1, 'string', [0, 1, 2]]
[1, 'string', [0, 1, 2], 'hello']
[1, 'string', 'world', [0, 1, 2], 'hello']
```

3 Counting elements in a list

We can also count how many times we encounter an element in a `list`. We can use the `count` method for that.

```
In [14]: # A = [1, 'string', 'world', [0, 1, 2], 'hello']
         c = A.count(1)
```

```
print(c)
d = A.count('world')
print(d)
```

```
1
1
```

4 Extending in a list

To extend a `list` means that you are gluing two lists together

```
In [16]: D = [1, 2, 3, 4]
         E = [5, 6, 7]

         # If we appended E to D, we would get a 5 element list, with the [5, 6, 7] attached a
         D.append(E)
         print(D)

         # In order to make D an EXTENDED version of this list (i.e. 7 elements), you use the
         D = [1, 2, 3, 4]
         D.extend(E)
         print(D)
```

```
[1, 2, 3, 4, [5, 6, 7]]
[1, 2, 3, 4, 5, 6, 7]
```

5 Popping and Removing Items

The pop method allows you to remove elements in a list at the end (which is the default) or at any position you desire

```
In [19]: V = [5, 4, 3, 2, 1]
         print(V)
         # V.pop will:
         # 1. It removes the last element from this list
         # 2. It returns that element to you in a variable
         d = V.pop()
         print(V)
         print(d)
         # You can just do
         # V.pop() which will just remove the last element and you don't save that element var

         # You can also remove at any arbitrary position.  Just specify the index of where you
         # V = [5, 4, 3, 2]
         e = V.pop(1)
         print(V)
         print(e)
```

```
[5, 4, 3, 2, 1]
[5, 4, 3, 2]
1
[5, 3, 2]
4
```

We can use `remove` remove the first occurrence of an element that exists in the list. We don't operate on the indices or locations of the list

```
In [24]: F = [1, 2, 1, 1, 3, 2, 3, 'hello', 'hello']

         F.remove(1)
         print(F)
         F.remove(1)
         print(F)
         F.remove(3)
         print(F)
         F.remove('hello')
         print(F)
         F.remove('world')
```

```
[2, 1, 1, 3, 2, 3, 'hello', 'hello']
[2, 1, 3, 2, 3, 'hello', 'hello']
[2, 1, 2, 3, 'hello', 'hello']
[2, 1, 2, 3, 'hello']
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-24-679eff019f47> in <module>()
  9 F.remove('hello')
 10 print(F)
---> 11 F.remove('world')


ValueError: list.remove(x): x not in list
```

6 Sorting

Last but not least - We can `sort` a list

```
In [25]: A = [1, 5, 4, 3, 3, 2, 10, 20, -1 # Create a list
         A.sort() # The sort method sorts a list - it does in place
         print(A)
```

```
[-1, 1, 2, 3, 3, 4, 5, 10, 20]
```

If you want to return a newly sorted list while maintaining the original one, use the `sorted` method. This is a function built-in to Python:

```
In [26]: A = [1, 5, 4, 3, 3, 2, 10, 20, -1]
         B = sorted(A)
         print(A)
         print(B)
```

```
[1, 5, 4, 3, 3, 2, 10, 20, -1]
[-1, 1, 2, 3, 3, 4, 5, 10, 20]
```

7 Tuples

They are like lists, but once you create them you cannot change their contents. You use tuples to ensure that the information is not changed. To create them, just use parantheses instead of `[]`

```
In [30]: B = (1, 2, 'hello', 'thomas')
         print(B)
         print(B[0])
         print(B[1:])  # Gives you another tuple of just the last three elements
         B[2] = -1
```

```
(1, 2, 'hello', 'thomas')
1
(2, 'hello', 'thomas')
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-30-57c55400b3fd> in <module>()
          3 print(B[0])
          4 print(B[1:])  # Gives you another tuple of just the last three elements
    ----> 5 B[2] = -1


        TypeError: 'tuple' object does not support item assignment
```

8 Looping over lists

We can certainly loop over each element in list and we can do something with each element

```
In [31]: B = ['hello', 'how', 1, 2, 3, 'are', 'you']

         # l will contain one element from the list B and this starts from
         # the beginning, to the end
         for l in B:
             print(l)
```

```
hello
how
1
2
3
are
you
```

9 Using `enumerate`

We can also determine the position of where each element in the list occurred. Use the `enumerate` object in a loop

```
In [32]: for (i, l) in enumerate(B): # i contains the position and l contains the correspondin
             print (i, l)
```

```
0 hello
1 how
2 1
3 2
4 3
5 are
6 you
```

10 Using `zip`

Given a pair of lists that are the same size, we can iterate and access individual elements in the same position of the lists

```
In [34]: A = [0, 8, 5, 4, 3]
         B = ['a', 'c', 'thomas', 'hello', 'ray']

         for (i, j, k ,l) in zip(A, B): # zip returns a tuple that is of the same size for as
             print (i, j)         # Each element i is an element from the list A and j is an e
                                        # The elements correspond to the same positions starti
```

```
0 a 0 a
8 c 8 c
5 thomas 5 thomas
```

```
4 hello 4 hello
3 ray 3 ray
```