NOTES ON LISTS

ERIC MARTIN

1. Operations on lists

Some operations change a list, others create a new list:

```
>>> L1 = [1, 2, 3]
>>> L2 = L1
>>> L2
[1, 2, 3]
>>> L2 += [4, 5]
>>> L2 = L2 + [6, 7]  Add a second list which can be easily removed
>>> L2
[1, 2, 3, 4, 5, 6, 7]
>>> L1
[1, 2, 3, 4, 5]
```

Date: Session 1, 2015.

2. List sizes

First note how many bytes are needed to store 0:

```
>>> import sys
>>> sys.getsizeof(0)
24
```

Creating longer and longer lists of 0's:

```
>>> for i in range(30):
        L = [0] * i
        print('{:} elements take {:} bytes.'.format(i, sys.getsizeof(L)))
```

```
0 elements take 64 bytes.
1 elements take 72 bytes.
2 elements take 80 bytes.
3 elements take 88 bytes.
4 elements take 96 bytes.
5 elements take 104 bytes.
6 elements take 112 bytes.
7 elements take 120 bytes.
8 elements take 128 bytes.
9 elements take 136 bytes.
10 elements take 144 bytes.
11 elements take 152 bytes.
12 elements take 160 bytes.
13 elements take 168 bytes.
14 elements take 176 bytes.
15 elements take 184 bytes.
16 elements take 192 bytes.
17 elements take 200 bytes.
18 elements take 208 bytes.
19 elements take 216 bytes.
20 elements take 224 bytes.
21 elements take 232 bytes.
22 elements take 240 bytes.
23 elements take 248 bytes.
24 elements take 256 bytes.
25 elements take 264 bytes.
26 elements take 272 bytes.
27 elements take 280 bytes.
28 elements take 288 bytes.
29 elements take 296 bytes.
```

Making a list longer and longer:

```
>>> L = []
>>> for i in range(30):
        print('{:} elements take {:} bytes.'.format(i, sys.getsizeof(L)))
        L.append(0)
```

```
0 elements take 64 bytes.
1 elements take 96 bytes.
2 elements take 96 bytes.
3 elements take 96 bytes.
4 elements take 96 bytes.
5 elements take 128 bytes.
6 elements take 128 bytes.
7 elements take 128 bytes.
8 elements take 128 bytes.
9 elements take 192 bytes.
10 elements take 192 bytes.
11 elements take 192 bytes.
12 elements take 192 bytes.
13 elements take 192 bytes.
14 elements take 192 bytes.
15 elements take 192 bytes.
16 elements take 192 bytes.
17 elements take 264 bytes.
18 elements take 264 bytes.
19 elements take 264 bytes.
20 elements take 264 bytes.
21 elements take 264 bytes.
22 elements take 264 bytes.
23 elements take 264 bytes.
24 elements take 264 bytes.
25 elements take 264 bytes.
26 elements take 344 bytes.
27 elements take 344 bytes.
28 elements take 344 bytes.
29 elements take 344 bytes.
```

3. Hidden complexity of operations

Timing the time it takes (in seconds) to append n elements to an empty list:

```
>>> from time import time
>>> def time_append(n):
    L = []
    now = time()
    for i in range(n):
        L.append(0)    Add 0 to end of the list
    return time() - now

>>> time_append(10)
5.9604644775390625e-06
>>> time_append(100)
1.5974044799804688e-05
>>> time_append(1000)
9.679794311523438e-05
>>> time_append(10000)
0.0009160041809082031
>>> time_append(100000)
0.008705854415893555
>>> time_append(1000000)
0.09330201148986816
>>> time_append(10000000)
0.9275541305541992
>>> time_append(100000000)
9.481359958648682
```

Timing the time it takes (in seconds) to insert n elements at the beginning of an empty list:

```
>>> def time_insert_at_beginning(n):
    L = []
    now = time()
    for i in range(n):
        L.insert(0, 0)
    return time() - now

>>> time_insert_at_beginning(10)
7.152557373046875e-06
>>> time_insert_at_beginning(100)
2.7179718017578125e-05
>>> time_insert_at_beginning(100000)
2.103389024734497
>>> time_insert_at_beginning(10)
7.152557373046875e-06
>>> time_insert_at_beginning(100)
2.5033950805664062e-05
>>> time_insert_at_beginning(1000)
0.00038695335388183594
>>> time_insert_at_beginning(10000)
0.022001981735229492
>>> time_insert_at_beginning(100000)
2.104335069656372
>>> time_insert_at_beginning(1000000)
225.46559500694275
```

append puts element at the end, insert needs to shift all the elements to the right first before inserting the element at beginning so not as efficient.