9-27 Lecture:

Review:

- .split() will yield a list of words, like [“this”, “is”, “a”, “test”] by discarding all the inter-word spaces
- QOTD8 ideal solution:

```
Def charCnt(S):
       return(sum(map(len, S.split())))
```

- Map: is a function of the same type, takes a sequence, first argument is the name of the function, which gets applied to each element in sequence and get back a list of values returned
-

Comprehensions:

- The general form of a simple list comprehension is:

```
[ expression(x) for x in sequence ]
```

Where sequence can be any of the types(lists, tuples, strings, ranges) or collections, such as sets. More generally:

```
[expression(x) for x in sequence1 for y in sequence2 ... if condition(x,y)]
```

```
>>>[(x,y) for x in range(3) for y in range(3) if x !=y]
[(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)]
#if x != y ensures only pairs where x and y are not the same are included.
>>>[(x,y) for x in range(3) for y in range(3) if x > y]
[(1,0), (2,0), (2,1)]
```

\
More comprehensions:

- In addition to list comprehensions, there are set and dictionary comprehensions as well, where the outer[] are replaced with{}:

```
>>>[ x+y for x in range(3) for y in range(3)]
{0,1,2,3,4}
>>>[ x:y for x in range(3) for y in range(3) if x != y]
{0: 2, 1: 2, 2: 1} #keys are unique, order matters
```

```
>>>[ x:y for x in range(3) for y in range(2, -1, -1) if x != y]
{0: 1, 1: 0, 2: 0} #last added takes precedence
```

CAREEFUL! Use x:y to get a dictionary, else you might get something you dont expect:

```
>>>{ (x,y) for x in range(3) for y in range(3) if x !=y }
{(0, 1), (1, 2), (2, 1), (2, 0), (0, 2), (1, 0)}
```

You don't get a dictionary, you get tuples which are immutable. Tuples can be elements of sets. You get oprdered set of all tuples(x,Y) where x!= y

```
>>> [ x:y for x in range(3) for y in range(3) if x !=y ]
SyntaxError: invalid syntax
```

Why not String Comprehensions?:

- Strings can be the sequence in a comprehension, but you cannot construct a string in a comprehension

```
[x.upper() for x in "testing" ]
['T', 'E', 'S', 'T', 'I', 'N', 'G']
```

- On the other hand, string methods can be used to convert the output of a list into a string.

```
''.join([ x.upper() for x in "testing" ])
'TESTING'
```

- Where string.join(L) uses string to paste together elements of L.
- Careful! Order matters(beware of sets!):

```
>>> '-+-'.join({ str(x) for x in range(5)})
'0-+-1-+-3-+-2-+-4'
```

Tuple Comprehensions?:

- There is no such thing as a tuple comprehension: instead, a () enclosed “comprehension” is a generator expression, which returns an object much like range(), but with all of comprehensions flexibility
- Lazy evaluations: basically if u say compute something snd it returns huge lisst and then say return 7 is when it starts computing till number 7

```
>>>(x +1 for x in range(1000000))
<generator object <genexpr> at 0x000001C392A2DBE0>
>>>sum((x +1 for x in range(1000000)))
```

```
500000500000
```

- Generators will come later: but kow they behave a little bit like ranges, but cannot be indexed. Instead they are meant to be accessed in order, and can be “exhausted” once evaluated. Best o aavoid

  Examples:

```
>>> mapExponent(list(range(1,6)), 3)
[1, 8, 27, 64, 125]
>>> mapExponent(list(range(1,6)), 0)
[1, 1, 1, 1, 1]
>>> mapExponent(list(range(-6,0)), 2)
[36, 25, 16, 9, 4, 1]

def mapExponent(L, k):
    return([ i**k for i in L ])
```

  This is similar to map() but in some sense more natural and easier to use, since map() is restricted to mapping functions as opposed to expressions over the sequence. Map expects one argument

```
# Specification: collectDivisible(max, k) takes two integers max and k
# and returns a list containing all integers 0 < i <= max such that i
# is evenly divisible by k. Use a comprehension.

>>> collectDivisible(20,3)
[3, 6, 9, 12, 15, 18]
>>> collectDivisible(20,2)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
>>> collectDivisible(20,5)
[5, 10, 15, 20]

def collectDivisible(max, k):
    return([ i for i in range(1, max+1, 1) if i%k == 0 ])
```