# CS1 Lecture 21 Mar. 6, 2017

- HW 5 Q2 and 3 today 5pm.
  - Q1 is more than half the work. You should have started working on it.
- Discussion sections tomorrow/Wed. – no points. Just homework 5 Q1 help.
- HW 4 graded by end of tomorrow

# Today

- Chapter 12 – tuples (and zip and sort and lambda)
- HW 5 discussion/tips

# Ch 12: Tuples

- Another sequence type (you've seen list, string, range so far) is **tuple**
- Tuples are just like lists except they are *immutable*!
- Create by typing a comma-separated list of values
  - Standard practice is to enclose the list in parentheses (but that's not required)

```
>>> myTuple = (1,2,3)
>>> myList = [1,2,3]
>>> len(myTuple)
3
>>> myList[0] = 4
>>> myList
[4, 2, 3]
>>> myTuple[0] = 4
...
TypeError: 'tuple' object does not support item assignment
>>> newList = list(myTuple)
>>> newList
[1, 2, 3]
```

# Ch 12: Tuples

```
>>> myTuple = 10, 11, 12
>>> myTuple
(10, 11, 12)
>>> myTuple = (3)
>>> myTuple
3
>>> myTuple = (3,)
>>> myTuple
(3,)
>>> myTuple = ()
```

NO! This is just 3

Tuple of length 1

Empty tuple – length 0

Adding tuples works:

```
>>> myTuple = (1,2,3)
>>> myTuple + myTuple
(1,2,3,1,2,3)
```

But append and other operations that mutate lists do not

# Ch 12: Tuples

You don't *need* tuples. You could use lists anywhere instead. But the immutability is sometimes desirable.

E.g. sometimes good to pass tuples as arguments to functions. The function can't then (accidentally or purposely) modify your data!

Easy to create a tuple from a list:

```
>>> importantList = [10, 11, 12]
>>> safeTuple = tuple(importantList)
>>> safeTuple
(10, 11, 12)
```

lec21.py

# Ch 12: Tuple assignment

Some of you have learned you can write things like:

```
>>> a, b = 1, 2
```

(or even (a,b) = (1,2))

```
>>> a
1
>>> b
2
```

This is called tuple assignment (even though you don't really need to think about tuples here)

# Ch 12: Tuple assignment

Useful for swapping values

```
>>> a, b = 1, 2
>>> a = b
>>> b = a
```

Does not correctly swap!

```
>>> temp = a
>>> a = b
>>> b = temp
```

Does correctly swap.

But so does

```
>>> a, b = b, a
```

```
          2, 1
    a, b = 2, 1
```

Why does this work?

Remember our old rule:

1. evaluate right hand side
2. update values var names refer t

```
>>> a
2
>>> b
1
```

# Ch 12: Tuple assignment

Also useful for "unpacking" results from functions that return list or tuple of results

```
def twiceAndThriceN(n):
    return (2*n, 3*n)                    [2*n, 3*n] also ok

>>> twoN, threeN = twiceAndThriceN(5)
>>> twoN
10
>>> threeN
15
>>> result = twiceAndThriceN(5)          also works, of course
>>> result
(10, 15)
```

# Ch 12: Zip (in section called Lists and Tuples)

zip is a convenient function for "re-packaging" sequences (for me, was useful for HW5 Q1, so perhaps for you too ☺)

Given two (or more) sequences, zip returns an **iterator** of tuples, where the i-th tuple contains the i-th element from each of the argument sequences.

zip([1,2,3], ['a', 'b', 'c']) is *like* the tuple of tuples:

( (1, 'a'), (2, 'b'), (3, 'c') )

but isn't exactly that…

```
>>> zippedStuff = zip([1,2,3], ['a', 'b', 'c'])
>>> zippedStuff
<zip object at 0x1045c2c48>
```

??? It's an **iterator**

# Quick look at iterators (not required)

zip returns a zip object, which is a kind of **iterator.** Iterators (and generators) are important more advanced Python concepts that you are not required to know.

```
>>> zippedStuff = zip([1,2,3], [‘a’, ‘b’, ‘c’])
>>> zippedStuff
<zip object at 0x1045c2c48>
```

What can you do with that zip object?

1) Turn it into a list:

```
>>> zippedStuffList = list(zippedStuff)
>>> zippedStuffList
[(1, 'a'), (2, 'b'), (3, 'c')]
```

common/useful

# What can you do with iterator object like result of zip?

```
>>> zippedStuff = zip([1,2,3], [‘a’, ‘b’, ‘c’])
>>> zippedStuff
<zip object at 0x1045c2c48>
```

2) use it with for loops

```
>>> for element in zippedStuff:
        print(element)
(1, ‘a’)
(2, ‘b’)
(3, ‘c’)
```

# What can you do with iterator object like result of zip?

```
>>> zippedStuff = zip([1,2,3], [‘a’, ‘b’, ‘c’])
>>> zippedStuff
<zip object at 0x1045c2c48>
```

3) ask for the items one by one

```
>>> next(ZippedStuff)
(1, ‘a’)
>>> next(zippedStuff)
(2, ‘b’)
>>> next(zippedStuff)
(3, ‘c’)
>>> next(zippedStuff)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    next(zippedStuff)
StopIteration
```

next is a special built-in func. for working with iterators

# What can you do with iterator object like result of zip?

```
>>> zippedStuff = zip([1,2,3], [‘a’, ‘b’, ‘c’])
```

BUT BE CAREFUL! You can only “use” iterator once!

```
>>> for element in zippedStuff:
        print(element)
(1, ‘a’)
(2, ‘b’)
(3, ‘c’)
>>> list(zippedStuff)
[]
>>> next(zippedStuff)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    next(zippedStuff)
StopIteration
>>>
```

Nothing’s left!

Nothing’s left!

# Ch 12: zip

Typically, we'll use list(zip(…)) or tuple(zip(…))

zipObj = zip([1, 2, 3], ['uno', 'dos', 'tres'], ['mot', 'hai', 'ba'])

zippedList = list(zipObj)

(or zippedList = tuple(zipObj)

[(1, 'uno', 'mot'), (2, 'dos', 'hai'), (3, 'tres', 'ba')]

Notice: in example above three lists were zipped (rather than two in earlier examples – can zip any number)

What happens when lists/things being zipped aren't the same length? *Try it …*

# For HW5 Q1, sorting is helpful

Why?

We might have dictionary:

{‘free’: 23, ‘you’: 50, ‘go’: 10, ‘zoo’: 1}

How can we extract the words in order starting with most frequent?

sort/sorted “work” on dictionaries but do they do what we want?

```
>>> d = {‘free’: 23, ‘you’: 50, ‘go’: 10, ‘zoo’: 1}
>>> sorted(d)
['free', 'go', 'you', 'zoo']
```

helpful???

# For HW5 Q1, sorting is helpful

But suppose we have a list of tuples instead of a dictionary.

tl = [('free', 23), ('you', 50), ('go', 10), ('zoo', 1)]

>>> sorted(tl)
[('free', 23), ('go', 10), ('you', 50), ('zoo', 1)] Now helpful? Not very

sorts based on whole tuple

sorted (and sort) have two useful optional arguments:

key: you provide a little function that to apply to item to generate key to use to sort

reverse: provide True if you want list from largest to smallest instead of default of smallest to largest

# For HW5 Q1, sorting is helpful

sorted (and sort) have two useful optional arguments:

1) key: a little function that is applied to item to generate key to use to sort

```
>>> tl = [('free', 23), ('you', 50), ('go', 10), ('zoo', 1)]
>>>sorted(tl, key = item1)                          if item1 function exists.
>>>sorted(tl, key = lambda item: item[1])     but don't need to write separate
                                                 function. 'lambda' allows you to
                                                 define (anonymous function)
                                                 anywhere
[('zoo', 1), ('go', 10), ('free', 23), ('you', 50)]   yes - better!
```

Also use other optional argument,

2) reverse: True if you want list from largest to smallest instead of default of smallest to largest

( see lec21.py)

```
>>> sorted(tl, key = lambda item: item[1], reverse = True)
[('you', 50), ('free', 23), ('go', 10), ('zoo', 1)]  That's what we want!
```

# For HW5 Q1, sorting is helpful

How do you use this stuff in HW5?

You create two dictionaries of word counts.
Want to extract items with highest counts.

1. Saw (three slides back) that

    sort(dict)

didn't quite give us what we needed

2. Saw (in last two slides) that we can usefully sort list of tuples

So … can you make a list of tuples [ … (word, count) …] from a dictionary?

There are a couple of ways … (one based on something learned earlier today)

# HW5

It is not hard if you do a little bit at a time. Get it working bit by bit.

1. Read the file, storing all the messages and their labels (spam/ham)
   - [[‘ham’, ‘text me later!’], [‘spam’, ‘call 1412 to win’]]
   - [[‘ham’, [‘text’, ‘me’, ‘later!’]], [‘spam’, [‘call’, ‘1412’, …]]
   - Or use two separate lists, one for spam, one for ham
2. For each message, extract its words, and update spam or ham dictionary of word counts accordingly
   - for ‘text me later!’ increment ‘text’, ‘me’, ‘later’ entries in ham dictionary
3. Use the two dictionaries to compute and print some statistics
   - get total spam/ham word counts and unique word counts
   - extra most common words from dictionaries
   - print some stats (*maybe* eliminating tiny words, numbers, …? Experiment to see what produces interesting output)

# HW5

1. File has some non-Ascii characters.

   use: open(fileName, encoding = 'utf-8')

2. To break line into tokens – individual elements of a line, learn how to use string **split**

   ```
   for line in fileStream:
       lineAsList = line.split()
   ```

   lec20split.py

3. get rid of extra stuff "…cool!?" learn how to use string **strip** (and/or lstrip, rstrip)

# Next Time

- A bit of Ch 19: conditional expressions and list comprehensions (will be topic of HW5 Q2 and Q3)