How to Do Best in This Class

• It’s impossible to learn programming by reading!

• There’s a day between the lecture and the lab, use it:
  a. Read the book chapters we covered,
  b. Practice with the problems in the book.

• Your life will be a lot easier, believe me!

• There will be a homework assignment:
  Tobi will send an email about it out tonight.
  • You can get extra credit by handing it in a day early.
Dictionaries, Text, Tuples
We’re Familiar with Lists

[2, 3, 51, 0]
[[‘a’, 151], [‘z’, 23], [‘i’, -42.3]]

• Lists enable us to store multiple values in one “variable”
• Lists are ordered, and we can iterate through them
• We can easily find and/or change a particular element

x = [2, 3, 51, 0]
x[2] = -23.0
print (x)
A Story of Two Collections..

• List
  - A linear collection of values that stay in order

• Dictionary
  - A “bag” of values, each with its own label
It Would Be Hard to Keep Track of My Jelly Bean Collection with a List

Buttered Popcorn
Chocolate Pudding
Mixed Berry Smoothie
Orange Sherbet
Peach
Raspberry
Red Apple
Sparkling Berry Blue
Sparkling Blueberry
Sparkling Cream Soda
Sparkling Grape Soda
Sparkling Green Apple
Sparkling Island Punch
Sparkling Orange
Sparkling Sour Apple
Sparkling Sour Lemon
Sparkling Very Cherry
Sparkling Wild Blackberry
Strawberry Jam
Watermelon

This is just a small part of it!
Dictionaries

- Dictionaries are Python’s most powerful data collection
- Dictionaries allow us to do fast database-like operations in Python
- Dictionaries have different names in different languages
  - Associative Arrays - Perl / PHP
  - Properties or Map or HashMap - Java
  - Property Bag - C# / .Net

http://en.wikipedia.org/wiki/Associative_array
Dictionaries

- Lists **index** their entries based on the position in the list.

- **Dictionaries** are like bags - no order.

- So we **index** the things we put in the **dictionary** with a "lookup tag".

```python
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```
Comparing Lists and Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values.

```python
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```python
>>> ddd = dict()
>>> ddd["age"] = 21
>>> ddd["course"] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd["age"] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```
If the value in each list position stood for something specific, it would be hard to keep track of it. Not so in a dictionary!

```python
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]

>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```
Dictionary Literals (Constants)

- Dictionary literals use curly braces and have a list of key : value pairs

- You can make an empty dictionary using empty curly braces

```python
>>> jjj = { 'chuck' : 1, 'fred' : 42, 'jan' : 100}
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = {}
>>> print(ooo)
{}
>>> ```
Counting: A Common Use of Dictionaries
Most Common Name?
Most Common Name?

marquard  cwen  cwen
zhen  marquard  zhen
csev  zhen  csev
zhen  csev  zhen
Most Common Name?

marquard  cwen  cwen
zhen      marquard  zhen
Csev      Csev  Csev
Zhen
Many Counters with a Dictionary

One common use of dictionaries is counting how often we “see” something

```python
>>> ccc = dict()
>>> ccc['csev'] = 1
>>> ccc['cwen'] = 1
>>> print(ccc)
{'csev': 1, 'cwen': 1}
>>> ccc['cwen'] = ccc['cwen'] + 1
>>> print(ccc)
{'csev': 1, 'cwen': 2}
```
Dictionary Tracebacks

• It is an **error** to reference a key which is not in the dictionary

• We can use the **in** operator to see if a key is in the dictionary

```python
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in ccc
False
```
When We See a New Name

When we encounter a new name, we need to add a new entry in the dictionary and if this the second or later time we have seen the name, we simply add one to the count in the dictionary under that name.

```python
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names:
    if name not in counts:
        counts[name] = 1
    else:
        counts[name] = counts[name] + 1
print(counts)
```

```python
{'csev': 2, 'zqian': 1, 'cwen': 2}
```
The **get** Method for Dictionaries

The pattern of checking to see if a **key** is already in a dictionary and assuming a default value if the **key** is not there is so common that there is a **method** called **get()** that does this for us.

```python
if name in counts:
    x = counts[name]
else:
    x = 0

x = counts.get(name, 0)

{'csev': 2, 'zqian': 1, 'cwen': 2}
```

Default value if key does not exist (and no Traceback).
Simplified Counting with \texttt{get()} \\

We can use \texttt{get()} and provide a default value of zero when the key is not yet in the dictionary - and then just add one \\

\begin{verbatim}
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names:
    counts[name] = counts.get(name, 0) + 1
print(counts)
\end{verbatim}

Default \hspace{1cm} \{'csev': 2, 'zqian': 1, 'cwen': 2\}
So Let’s Count

https://www.youtube.com/watch?v=voor_2lo9ujg
Counting Words in Text

the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car
The general pattern to count the words in a line of text is to split the line into words, then loop through the words and use a dictionary to track the count of each word independently.

counts = dict()
print('Enter a line of text: ')
line = input(' ')

words = line.split()

print('Words: ', words)

print('Counting...')
for word in words:
    counts[word] = counts.get(word, 0) + 1
print('Counts', counts)
python wordcount.py
Enter a line of text:
the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car

Words: ['the', 'clown', 'ran', 'after', 'the', 'car', 'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and', 'the', 'tent', 'fell', 'down', 'on', 'the', 'clown', 'and', 'the', 'car']
Counting...

Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7, 'tent': 2}

http://www.flickr.com/photos/71502646@N00/2526007974/
counts = dict()
line = input('Enter a line of text: ')
words = line.split()

print('Words:', words)
print('Counting...')

for word in words:
    counts[word] = counts.get(word, 0) + 1
print('Counts', counts)

python wordcount.py
Enter a line of text:
the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car

Words: ['the', 'clown', 'ran', 'after', 'the', 'car', 'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and', 'the', 'tent', 'fell', 'down', 'on', 'the', 'clown', 'and', 'the', 'car']
Counting...

Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'and': 1, 'the': 7, 'down': 1, 'fell': 1, 'the': 3, 'clown': 2, 'into': 1, 'after': 1}
Definite Loops and Dictionaries

Even though dictionaries are not stored in order, we can write a for loop that goes through all the entries in a dictionary - actually it goes through all of the keys in the dictionary and looks up the values.

```python
>>> counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for key in counts:
...    print(key, counts[key])
...
jan 100
chuck 1
fred 42
>>>
Helpful Dictionary Methods Exist!

• You easily can get a list of keys, values, or both (items) from a dictionary

```python
dict.keys()
dict.values()
dict.items()
```

• Check it out!
Retrieving Lists of Keys and Values

You can get a list of **keys**, **values**, or **items (both)** from a dictionary.

```python
>>> jjj = { 'chuck' : 1, 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
```

What is a “tuple”? - coming soon...
Bonus: Two Iteration Variables!

- We loop through the key-value pairs in a dictionary using *two* iteration variables.

- Each iteration, the first variable is the key and the second variable is the corresponding value for the key.

```python
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

```
jan 100
[chuck] 1
[fred] 42
```
What Does This Code Do?

name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
Summary

• What is a collection?
• Lists versus Dictionaries
• Dictionary constants
• The most common word
• Using the get() method
• Hashing, and lack of order
• Writing dictionary loops
• Sneak peek: tuples
• Sorting dictionaries
Tuples
Tuples Are Like Lists

Tuples are another kind of sequence that functions much like a list - they have elements which are indexed starting at 0

```python
>>> x = ('Glenn', 'Sally', 'Joseph')
>>> print(x[2])
Joseph
>>> y = (1, 9, 2)
>>> print(y)
(1, 9, 2)
>>> print(max(y))
9
```
but... Tuples are “immutable”

Unlike a list, once you create a tuple, you cannot alter its contents - similar to a string

```python
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
Traceback: 'str' object does not support item assignment
>>> [9, 8, 6]
>>> y = 'ABC'
>>> y[2] = 'D'
Traceback: 'str' object does not support item assignment
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback: 'tuple' object does not support item assignment
>>> z = (5, 4, 3)
```
Things **not** to do With Tuples

```python
>>> x = (3, 2, 1)
>>> x.sort()
Traceback:
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback:
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback:
AttributeError: 'tuple' object has no attribute 'reverse'
```
A Tale of Two Sequences

```python
>>> l = list()
>>> dir(l)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

>>> t = tuple()
>>> dir(t)
['count', 'index']
```
Tuples are More Efficient

- Since Python does not have to build tuple structures to be modifiable, they are simpler and more efficient in terms of memory use and performance than lists.
- So in our program when we are making “temporary variables” we prefer tuples over lists.
Tuples and Assignment

• We can also put a tuple on the left-hand side of an assignment statement

• We can even omit the parentheses

```python
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> (a, b) = (99, 98)
>>> print(a)
99
```
The `items()` method in dictionaries returns a list of (key, value) tuples.
Tuples are Comparable

The comparison operators work with tuples and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```python
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ( 'Jones', 'Sally' ) < ( 'Jones', 'Sam' )
True
>>> ( 'Jones', 'Sally' ) > ( 'Adams', 'Sam' )
True
```
Once a dictionary is turned into a list of tuples, it can be sorted.

### Sorting Lists of Tuples

- We can take advantage of the ability to sort a list of tuples to get a sorted version of a dictionary.
- First we sort the dictionary by the key using the `items()` method and `sorted()` function.

```python
>>> d = {'a': 10, 'b': 1, 'c': 22}
>>> d.items()
dict_items([('a', 10), ('c', 22), ('b', 1)])
>>> sorted(d.items())
[('a', 10), ('b', 1), ('c', 22)]
```
Using `sorted()`

We can do this even more directly using the built-in function `sorted` that takes a sequence as a parameter and returns a sorted sequence.

```python
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = sorted(d.items())
>>> t
[('a', 10), ('b', 1), ('c', 22)]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
 a 10
 b 1
 c 22
```
‘sort’ vs. ‘sorted’

• ‘sort’ is a method that is defined only for lists, and modifies them in place

• ‘sorted’ is a built-in function that can be applied to any ‘iterable’ structure, and will turn it into a new list.
  E. g.,
  >>> sorted('abdce')
  ['a', 'b', 'c', 'd', 'e']

• ‘sorted’ can take an argument that defines the value to compare for each element — make sense when each element is a more complex structure

• Read more about sorting here:
  • https://wiki.python.org/moin/HowTo/Sorting
Sort by Values Instead of Key

- If we could construct a list of tuples of the form (value, key) we could sort by value.
- We do this with a for loop that creates a list of tuples.

```python
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items():
...     tmp.append( (v, k) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```
fhand = open('romeo.txt')
counts = {}
for line in fhand:
   words = line.split()
   for word in words:
      counts[word] = counts.get(word, 0) + 1

lst = []
for key, val in counts.items():
   newtup = (val, key)
   lst.append(newtup)

lst = sorted(lst, reverse=True)

for val, key in lst[:10]:
   print(key, val)
Even Shorter Version

```python
>>> c = {'a': 10, 'b': 1, 'c': 22}

>>> print(sorted([(v,k) for k,v in c.items()]))
[(1, 'b'), (10, 'a'), (22, 'c')]
```

List comprehension creates a dynamic list. In this case, we make a list of reversed tuples and then sort it.

http://wiki.python.org/moin/HowTo/Sorting
Summary

- Tuple syntax
- Immutability
- Comparability
- Sorting

- Tuples in assignment statements
- Sorting dictionaries by either key or value