CSC 221: Introduction to Programming

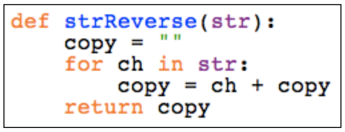
Fall 2013

Big data

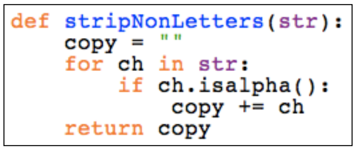
- building lists
  list comprehensions, throwaway comprehensions
  conditional comprehensions
- processing large data files
  example: dictionary, anagram finder
  example: NBA stats, player reports
  example: twitter stats, hashtag reports
  read-store-and-process vs. read-and-process

Review: building strings

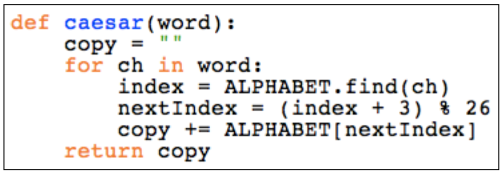
we have seen numerous examples of building strings



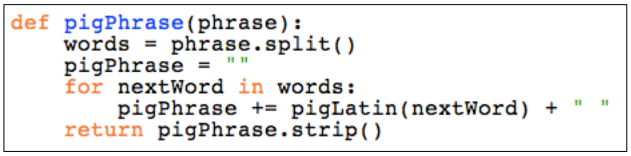
```
def strReverse(str):
    copy = ""
    for ch in str:
        copy = ch + copy
    return copy
```



```
def stripNonLetters(str):
    copy = ""
    for ch in str:
        if ch.isalpha():
            copy += ch
    return copy
```



```
def caesar(word):
    copy = ""
    for ch in word:
        index = ALPHABET.find(ch)
        nextIndex = (index + 3) % 26
        copy += ALPHABET[nextIndex]
    return copy
```



```
def pigPhrase(phrase):
    words = phrase.split()
    pigPhrase = ""
    for nextWord in words:
        pigPhrase += pigLatin(nextWord) + " "
    return pigPhrase.strip()
```

Similarly: building lists

since lists & strings are both sequences, can similarly build lists

```
def squares(numSquares):
    nums = []
    for i in range(1, numSquares+1):
        nums.append(i*i)
    return nums
```

```
def evens(numEvens):
    nums = []
    for i in range(1, numEvens+1):
        nums.append(2*i)
    return nums
```

```
def pigList(words):
    pigs = []
    for nextWord in words:
        pigs.append(pigLatin(nextWord))
    return pigs
```

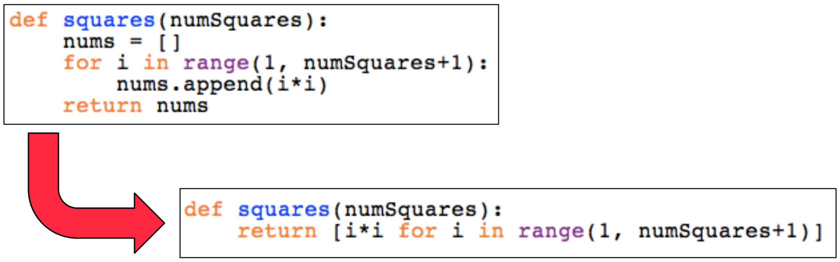
List comprehensions

Python has an alternative mechanism for building lists: comprehensions

- inspired by math notation for defining sets

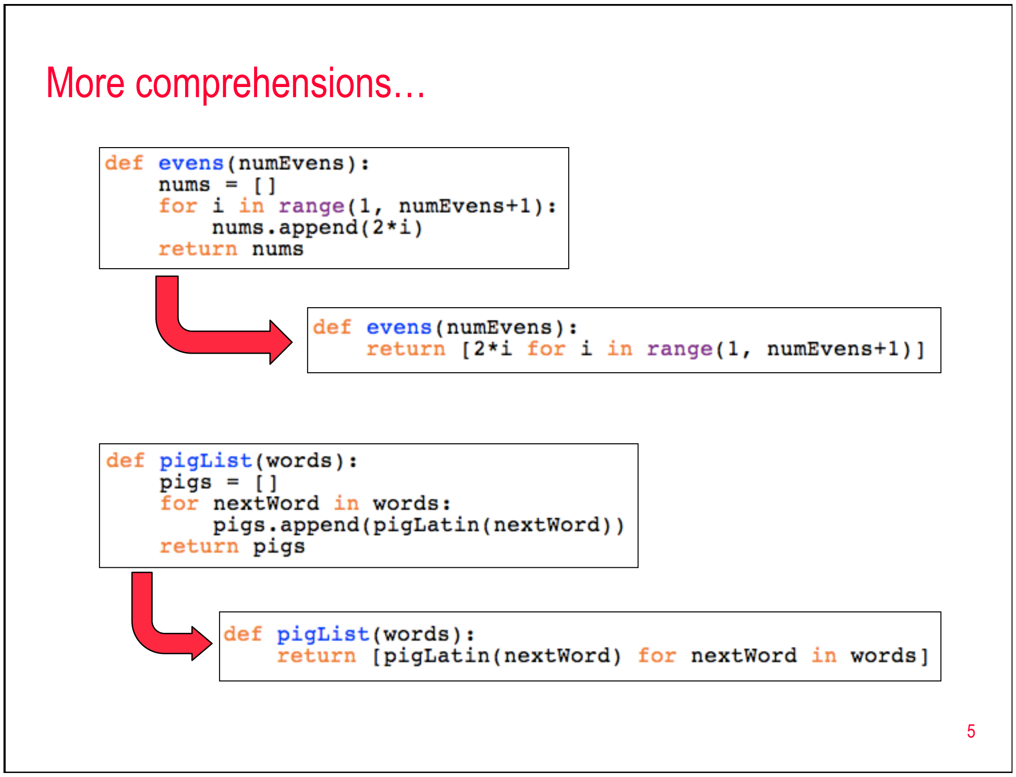
  squares(N) = $\{i^2 \quad \text{for } 1 \le i \le N$

- in Python: `[EXPR_ON_VAR for VAR in LIST]`



```
def squares(numSquares):
    nums = []
    for i in range(1, numSquares+1):
        nums.append(i*i)
    return nums
```

```
def squares(numSquares):
    return [i*i for i in range(1, numSquares+1)]
```



More comprehensions…

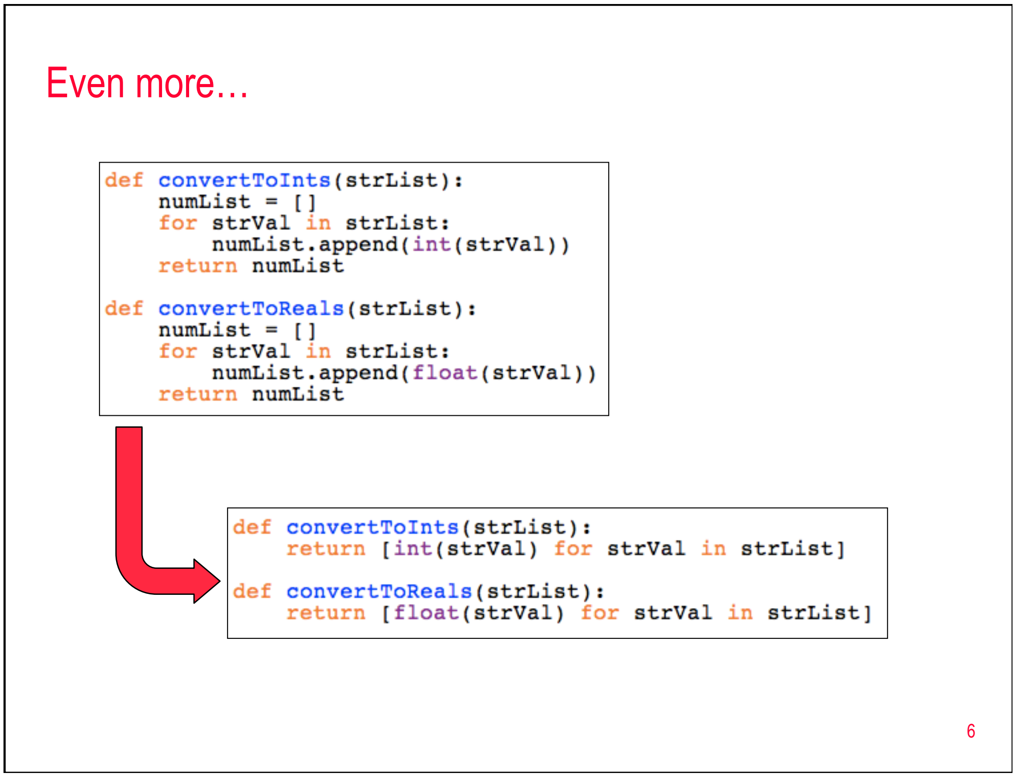
```
def evens(numEvens):
    nums = []
    for i in range(1, numEvens+1):
        nums.append(2*i)
    return nums
```

```
def evens(numEvens):
    return [2*i for i in range(1, numEvens+1)]
```

```
def pigList(words):
    pigs = []
    for nextWord in words:
        pigs.append(pigLatin(nextWord))
    return pigs
```

```
def pigList(words):
    return [pigLatin(nextWord) for nextWord in words]
```

5



Even more…

```
def convertToInts(strList):
    numList = []
    for strVal in strList:
        numList.append(int(strVal))
    return numList

def convertToReals(strList):
    numList = []
    for strVal in strList:
        numList.append(float(strVal))
    return numList
```

```
def convertToInts(strList):
    return [int(strVal) for strVal in strList]

def convertToReals(strList):
    return [float(strVal) for strVal in strList]
```

6

Exercises:

use list comprehensions to build the following lists of numbers:

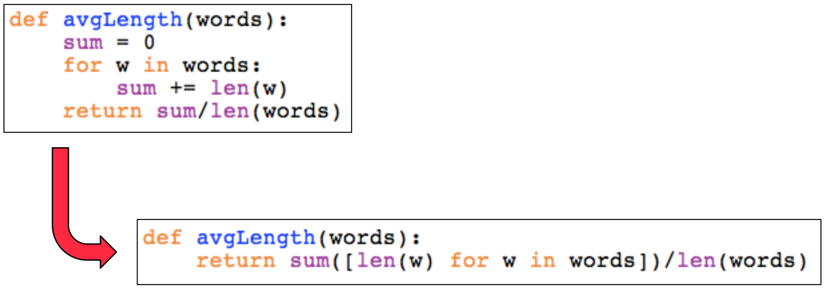
- a list of the first 20 odd numbers
- a list of the first 20 cubes
- a list of the first 20 powers of 2

given a list of words, use list comprehensions to build:

- a list containing all of the same words, but capitalized
- a list containing all of the same words, but each word reversed
- a list containing the lengths of all of the same words

Throwaway comprehensions

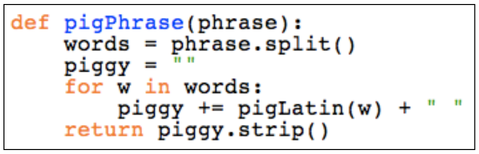
comprehensions can be useful as part of bigger tasks



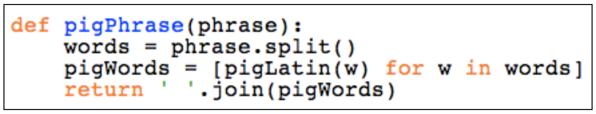
```
def avgLength(words):
    sum = 0
    for w in words:
        sum += len(w)
    return sum/len(words)
```

```
def avgLength(words):
    return sum([len(w) for w in words])/len(words)
```

Another example



```
def pigPhrase(phrase):
    words = phrase.split()
    piggy = ""
    for w in words:
        piggy += pigLatin(w) + " "
    return piggy.strip()
```



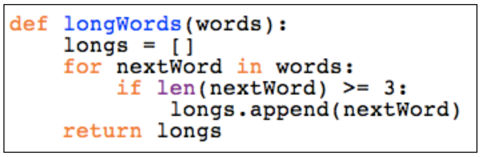
```
def pigPhrase(phrase):
    words = phrase.split()
    pigWords = [pigLatin(w) for w in words]
    return ' '.join(pigWords)
```

note: the string method `join` appends the contents of a list into a single string, using the specified divider

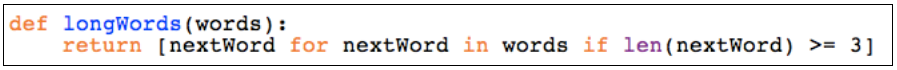
Conditional comprehensions

comprehensions can include conditions

- in general: `[EXPR_ON_VAR for VAR in LIST if CONDITION]`



```
def longWords(words):
    longs = []
    for nextWord in words:
        if len(nextWord) >= 3:
            longs.append(nextWord)
    return longs
```



```
def longWords(words):
    return [nextWord for nextWord in words if len(nextWord) >= 3]
```

Another example

```
def isPrime(num):
    if num < 2:
        return False
    elif num == 2:
        return True
    else:
        for i in range(2, (num/2)+1):
            if num % i == 0:
                return False
        return True

def primesInRange1(low, high):
    primes = []
    for i in range(low, high+1):
        if isPrime(i):
            primes.append(i)
    return primes
```

```
def primesInRange(low, high):
    return [i for i in range(low, high+1) if isPrime(i)]
```

Big data

lists enable us to store and process large amounts of data

- `dictionary.txt` contains 117,633 words
- suppose we want to write a function that finds anagrams

- repeatedly prompt the user for a word
  - traverse the dictionary to find & display all anagrams of that word

- we could start fresh for each word – read from the dictionary file and compare words
  - however, 117,633 words don't take up that much space (!)
  - can read and store them in a list, then repeatedly access – MUCH FASTER

- how can we determine if two words are anagrams?
  - generate the *alphagram* of each word:

    "spear" → "aeprs"   "pares" → "aeprs

  - if the alphagrams are identical, the words are anagrams

Anagram finder

```
import tkinter

def alphagram(word):
    """Returns the alphagram of word, i.e., word with the letters sorted."""
    return "".join(sorted(list(word)))


def anagrams(word, dictionary):
    """Returns a list of all anagrams of word from dictionary."""
    alpha = alphagram(word)
    return [dWord for dWord in dictionary if alphagram(dWord) == alpha]


def findAnagrams():
    """Repeatedly prompts the user for words and displays all anagrams."""
    dictFilename = \
      tkinter.filedialog.askopenfilename(**{"title":"Select the dictionary file"})
    dictFile = open(dictFilename, "r")
    dictionary = dictFile.read().split()
    dictFile.close()

    word = input("Enter a word (hit return to quit):")
    while word != "":
        anagramList = anagrams(word, dictionary)
        print(anagramList, ":", len(anagramList))
        word = input("Enter a word (hit return to quit):")
```

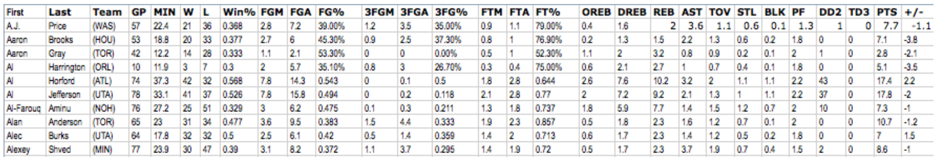
NBA stats

NBA2013.txt contains NBA stats from the 2012-2013 season

- downloaded from NBA.com (61 KB)
- 469 rows (excluding the header)

  if a player played for multiple teams in the year (e.g., as a result of a trade), there is a separate entry for each team
- each row consists of 29 data fields

  line[0] = firstName   line[1] = lastName   line[2] = (team)   line[27] = PPG



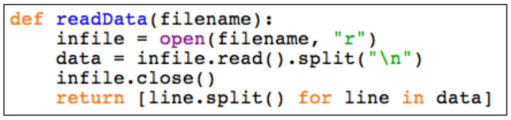
First	Last	Team	GP	MIN	W	L	Win%	FGM	FGA	FG%	3FGM	3FGA	3FG%	FTM	FTA	FT%	OREB	DREB	REB	AST	TOV	STL	BLK	PF	DD2	TD3	PTS	+/-
A.J.	Price	(WAS)	57	22.4	21	36	0.368	2.8	7.2	39.00%	1.2	3.5	35.00%	0.9	1.1	79.00%	0.4	1.6	2	3.6	1.1	0.6	0.1	1.3	1	0	7.7	-1.1
Aaron	Brooks	(HOU)	53	18.8	20	33	0.377	2.7	6	45.30%	0.9	2.5	37.30%	0.8	1	76.90%	0.2	1.3	1.5	2.2	1.3	0.6	0.2	1.8	0	0	7.1	-3.8
Aaron	Gray	(TOR)	42	12.2	14	28	0.333	1.1	2.1	53.30%	0	0	0.00%	0.5	1	52.30%	1.1	2	3.2	0.8	0.9	0.2	0.1	2	1	0	2.8	-2.1
Al	Harrington	(ORL)	10	11.9	3	7	0.3	2	5.7	35.10%	0.8	3	26.70%	0.3	0.4	75.00%	0.6	2.1	2.7	1	0.7	0.4	0.1	1.8	0	0	5.1	-3.5
Al	Horford	(ATL)	74	37.3	42	32	0.568	7.8	14.3	0.543	0	0.1	0.5	1.8	2.8	0.644	2.6	7.6	10.2	3.2	2	1.1	1.1	2.2	43	0	17.4	2.2
Al	Jefferson	(UTA)	78	33.1	41	37	0.526	7.8	15.8	0.494	0	0.2	0.118	2.1	2.8	0.77	2	7.2	9.2	2.1	1.3	1	1.1	2.2	37	0	17.8	-2
Al-Farouq	Aminu	(NOH)	76	27.2	25	51	0.329	3	6.2	0.475	0.1	0.3	0.211	1.3	1.8	0.737	1.8	5.9	7.7	1.4	1.5	1.2	0.7	2	10	0	7.3	-1
Alan	Anderson	(TOR)	65	23	31	34	0.477	3.6	9.5	0.383	1.5	4.4	0.333	1.9	2.3	0.857	0.5	1.8	2.3	1.6	1.2	0.7	0.1	2	0	0	10.7	-1.2
Alec	Burks	(UTA)	64	17.8	32	32	0.5	2.5	6.1	0.42	0.5	1.4	0.359	1.4	2	0.713	0.6	1.7	2.3	1.4	1.2	0.5	0.2	1.8	0	0	7	1.5
Alexey	Shved	(MIN)	77	23.9	30	47	0.39	3.1	8.2	0.372	1.1	3.7	0.295	1.4	1.9	0.72	0.5	1.7	2.3	3.7	1.9	0.7	0.4	1.5	2	0	8.6	-1

Stats list

recall that `infile.read()` reads the entire contents of `infile` into a single string

- repeatedly searching the string for player info would be tedious
- better to break the string into logical divisions, i.e., a list of player data sets

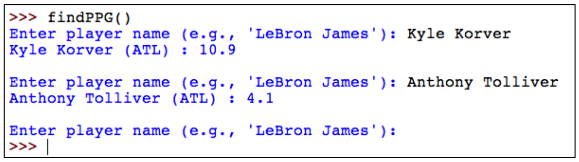
```
[['FIRST', 'LAST', 'TEAM', 'GP', 'MIN', 'W', 'L', 'Win%', 'FGM', 'FGA', 'FG%', '3FGM', '3FGA',
   '3FG%', 'FTM', 'FTA', 'FT%', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL', 'BLK', 'PF',
   'DD2', 'TD3', 'PTS', '+/-'],
 ['A.J.', 'Price', '(WAS)', '57', '22.4', '21', '36', '0.368', '2.8', '7.2', '39.0%', '1.2',
   '3.5', '35.0%', '0.9', '1.1', '79.0%', '0.4', '1.6', '2.0', '3.6', '1.1', '0.6', '0.1',
   '1.3', '1', '0', '7.7', '-1.1'],
 ['Aaron', 'Brooks', '(HOU)', '53', '18.8', '20', '33', '0.377', '2.7', '6.0', '45.3%', '0.9',
   '2.5', '37.3%', '0.8', '1.0', '76.9%', '0.2', '1.3', '1.5', '2.2', '1.3', '0.6', '0.2',
   '1.8', '0', '0', '7.1', '-3.8'],
 ['Aaron', 'Gray', '(TOR)', '42', '12.2', '14', '28', '0.333', '1.1', '2.1', '53.3%', '0.0',
   '0.0', '0.0%', '0.5', '1.0', '52.3%', '1.1', '2.0', '3.2', '0.8', '0.9', '0.2', '0.1',
   '2.0', '1', '0', '2.8', '-2.1'],
 .
 .
 .
]
```



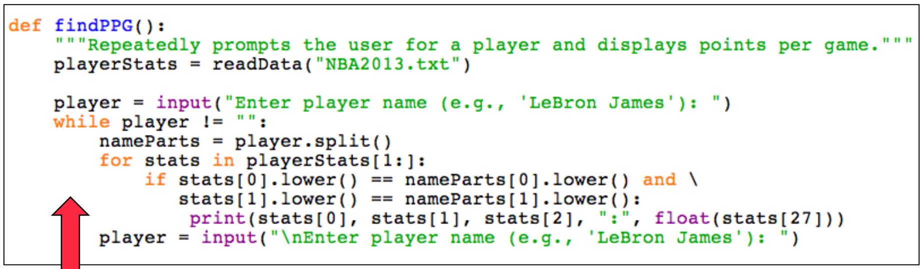
```
def readData(filename):
    infile = open(filename, "r")
    data = infile.read().split("\n")
    infile.close()
    return [line.split() for line in data]
```

QUESTION: should we store the header in the list?

Processing the stats



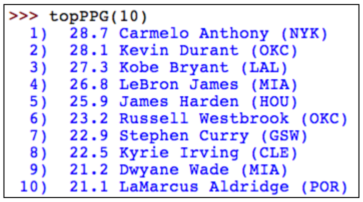
```
>>> findPPG()
Enter player name (e.g., 'LeBron James'): Kyle Korver
Kyle Korver (ATL) : 10.9

Enter player name (e.g., 'LeBron James'): Anthony Tolliver
Anthony Tolliver (ATL) : 4.1

Enter player name (e.g., 'LeBron James'):
>>> |
```



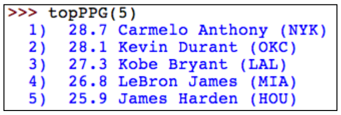
```
def findPPG():
    """Repeatedly prompts the user for a player and displays points per game."""
    playerStats = readData("NBA2013.txt")

    player = input("Enter player name (e.g., 'LeBron James'): ")
    while player != "":
        nameParts = player.split()
        for stats in playerStats[1:]:
            if stats[0].lower() == nameParts[0].lower() and \
               stats[1].lower() == nameParts[1].lower():
                print(stats[0], stats[1], stats[2], ":", float(stats[27]))
        player = input("\nEnter player name (e.g., 'LeBron James'): ")
```

for each player name entered by the user:
    traverse the playerStats list, looking at each player data set
    if the first & last names match, then print the name, team & PPG

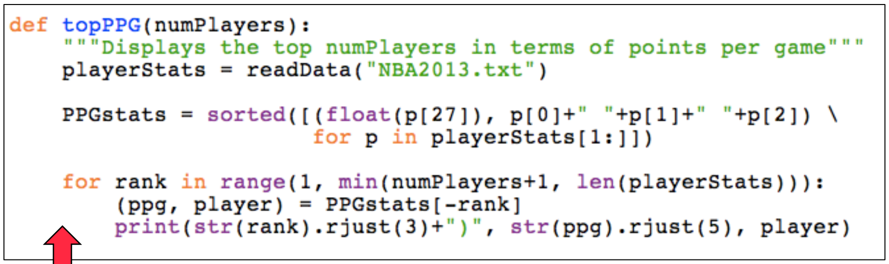
Processing the stats



```
>>> topPPG(10)
  1)  28.7 Carmelo Anthony (NYK)
  2)  28.1 Kevin Durant (OKC)
  3)  27.3 Kobe Bryant (LAL)
  4)  26.8 LeBron James (MIA)
  5)  25.9 James Harden (HOU)
  6)  23.2 Russell Westbrook (OKC)
  7)  22.9 Stephen Curry (GSW)
  8)  22.5 Kyrie Irving (CLE)
  9)  21.2 Dwyane Wade (MIA)
 10)  21.1 LaMarcus Aldridge (POR)
```



```
>>> topPPG(5)
  1)  28.7 Carmelo Anthony (NYK)
  2)  28.1 Kevin Durant (OKC)
  3)  27.3 Kobe Bryant (LAL)
  4)  26.8 LeBron James (MIA)
  5)  25.9 James Harden (HOU)
```



```
def topPPG(numPlayers):
    """Displays the top numPlayers in terms of points per game"""
    playerStats = readData("NBA2013.txt")

    PPGstats = sorted([(float(p[27]), p[0]+" "+p[1]+" "+p[2]) \
                       for p in playerStats[1:]])

    for rank in range(1, min(numPlayers+1, len(playerStats))):
        (ppg, player) = PPGstats[-rank]
        print(str(rank).rjust(3)+")", str(ppg).rjust(5), player)
```

use a comprehension to build a list of [PPG, PLAYER] pairs & sort

then, display the highest ranking players from the end of the sorted list

(be careful that there are that many players)

Really big data

if the data can be stored in a list, it can be accessed faster and repeatedly

- sometimes, there is just too much data to store

infochimps.com provides twitter data sets for research & analysis

- collected from March 2006 to November 2009
- > 35 million users, > 500 million tweets

- there are 41,746,479 lines in the raw data file (1.81 GB)
- each line consists of 4 fields, identifying tokens taken from tweets

tokenType	either `hashtag`, `url`, or `smiley`
yearAndMonth	e.g., `200901` for January, 2009
count	# of times that token appeared during that month
token	the text of the token

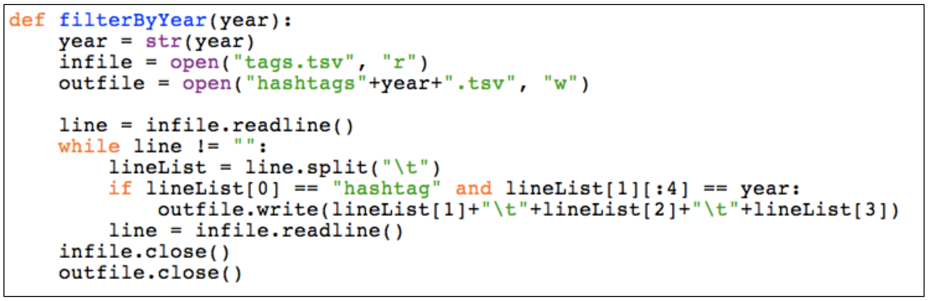
WAY TOO BIG TO FIT IN A PYTHON LIST!!!

Preprocessing

searching the entire data set takes too long

- better to partition it into more manageable subsets

- we only care about hashtags, so filter out the other entries
- also, we will search based on year, so can group data into separate files by year



```
def filterByYear(year):
    year = str(year)
    infile = open("tags.tsv", "r")
    outfile = open("hashtags"+year+".tsv", "w")

    line = infile.readline()
    while line != "":
        lineList = line.split("\t")
        if lineList[0] == "hashtag" and lineList[1][:4] == year:
            outfile.write(lineList[1]+"\t"+lineList[2]+"\t"+lineList[3])
        line = infile.readline()
    infile.close()
    outfile.close()
```

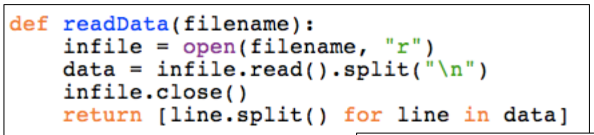
since the filtered files only contain hashtags, can omit the first field

use to generate `hashtags2006.tsv`, …, `hashtags2009.tsv`

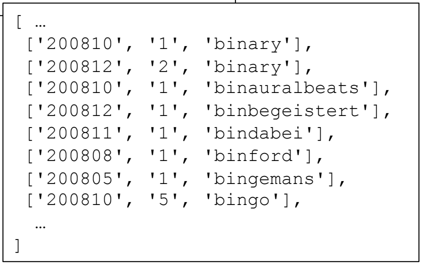
Mining the data

the yearly hashtag files are small enough to store and process

- hashtags2006.tsv → 18 lines (???)
- hashtags2007.tsv → 2,821 lines
- hashtags2008.tsv → 98,261lines
- hashtags2009.tsv → 1,588.960 lines



```
def readData(filename):
    infile = open(filename, "r")
    data = infile.read().split("\n")
    infile.close()
    return [line.split() for line in data]
```

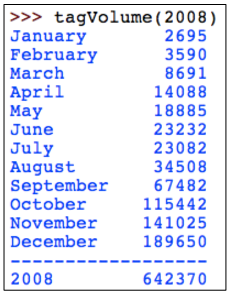


```
[ …
 ['200810', '1', 'binary'],
 ['200812', '2', 'binary'],
 ['200810', '1', 'binauralbeats'],
 ['200812', '1', 'binbegeistert'],
 ['200811', '1', 'bindabei'],
 ['200808', '1', 'binford'],
 ['200805', '1', 'bingemans'],
 ['200810', '5', 'bingo'],
  …
]
```

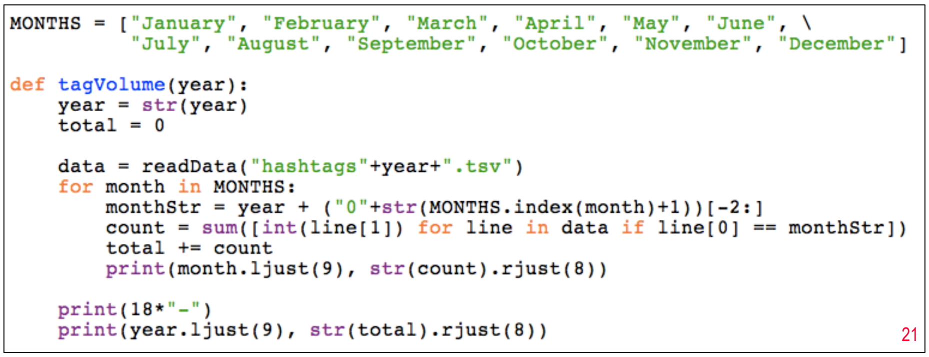
Tag volume

suppose we want the tag volume per month and year

- for each month,
  - use a comprehension to build a list of the counts from entries that corresponds to that month
  - call the sum function on that list to get the monthly volume
- also add the monthly count to a running total for the year



```
>>> tagVolume(2008)
January       2695
February      3590
March         8691
April        14088
May          18885
June         23232
July         23082
August       34508
September    67482
October     115442
November    141025
December    189650
------------------
2008        642370
```



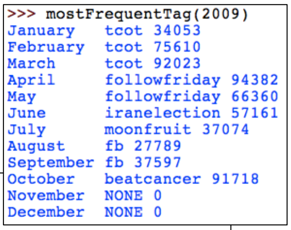
```
MONTHS = ["January", "February", "March", "April", "May", "June", \
          "July", "August", "September", "October", "November", "December"]

def tagVolume(year):
    year = str(year)
    total = 0

    data = readData("hashtags"+year+".tsv")
    for month in MONTHS:
        monthStr = year + ("0"+str(MONTHS.index(month)+1))[-2:]
        count = sum([int(line[1]) for line in data if line[0] == monthStr])
        total += count
        print(month.ljust(9), str(count).rjust(8))

    print(18*"-")
    print(year.ljust(9), str(total).rjust(8))
```

Most frequent tag

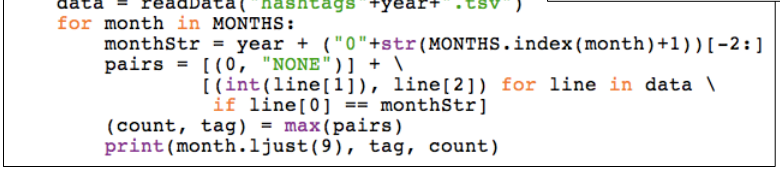
can look at historical snapshots by finding the most popular tags

- for each month, identify those entries corresponding to that month
  - use a comprehension to construct a list of (count,tag) pairs
  - then use max to find the pair with largest count

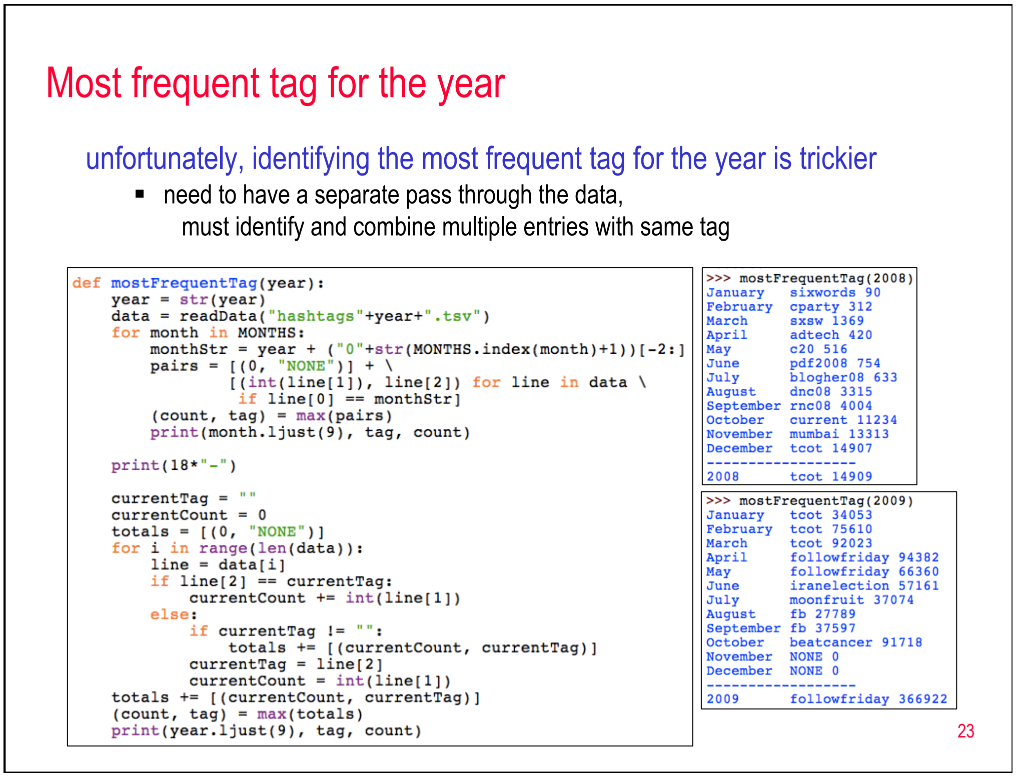
What is [0, "NONE"] for?



```
>>> mostFrequentTag(2009)
January    tcot 34053
February   tcot 75610
March      tcot 92023
April      followfriday 94382
May        followfriday 66360
June       iranelection 57161
July       moonfruit 37074
August     fb 27789
September  fb 37597
October    beatcancer 91718
November   NONE 0
December   NONE 0
```



```
def mostFrequentTag(year):
    year = str(year)
    data = readData("hashtags"+year+".tsv")
    for month in MONTHS:
        monthStr = year + ("0"+str(MONTHS.index(month)+1))[-2:]
        pairs = [(0, "NONE")] + \
                [(int(line[1]), line[2]) for line in data \
                 if line[0] == monthStr]
        (count, tag) = max(pairs)
        print(month.ljust(9), tag, count)
```



Most frequent tag for the year

unfortunately, identifying the most frequent tag for the year is trickier

- need to have a separate pass through the data,
  must identify and combine multiple entries with same tag

```python
def mostFrequentTag(year):
    year = str(year)
    data = readData("hashtags"+year+".tsv")
    for month in MONTHS:
        monthStr = year + ("0"+str(MONTHS.index(month)+1))[-2:]
        pairs = [(0, "NONE")] + \
                [(int(line[1]), line[2]) for line in data \
                 if line[0] == monthStr]
        (count, tag) = max(pairs)
        print(month.ljust(9), tag, count)

    print(18*"-")

    currentTag = ""
    currentCount = 0
    totals = [(0, "NONE")]
    for i in range(len(data)):
        line = data[i]
        if line[2] == currentTag:
            currentCount += int(line[1])
        else:
            if currentTag != "":
                totals += [(currentCount, currentTag)]
            currentTag = line[2]
            currentCount = int(line[1])
    totals += [(currentCount, currentTag)]
    (count, tag) = max(totals)
    print(year.ljust(9), tag, count)
```

```
>>> mostFrequentTag(2008)
January   sixwords 90
February  cparty 312
March     sxsw 1369
April     adtech 420
May       c20 516
June      pdf2008 754
July      blogher08 633
August    dnc08 3315
September rnc08 4004
October   current 11234
November  mumbai 13313
December  tcot 14907
------------------
2008      tcot 14909
```

```
>>> mostFrequentTag(2009)
January   tcot 34053
February  tcot 75610
March     tcot 92023
April     followfriday 94382
May       followfriday 66360
June      iranelection 57161
July      moonfruit 37074
August    fb 27789
September fb 37597
October   beatcancer 91718
November  NONE 0
December  NONE 0
------------------
2009      followfriday 366922
```