Problem:

Ask the user for two integer **n** and **m where m>n**.

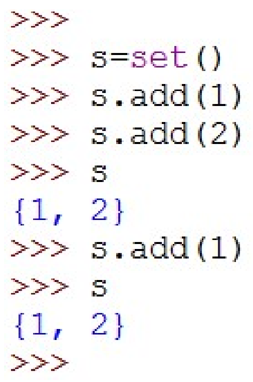
Create a list x of size n. Populate the list with n **random integers** in the range 1through m.

Problem:

Modify the answer to the problem above so that all of the integers in list x are **unique**.

Sets

Like a list, a set is a "container" that gives us a place to store a collection of elements.
Unlike a list, a set can contain **only one copy** of any element.



```
>>>
>>> s=set()
>>> s.add(1)
>>> s.add(2)
>>> s
{1, 2}
>>> s.add(1)
>>> s
{1, 2}
>>>
```

We can use Python's built-in **set** container to check for duplicates.

We can create a set S with the following syntax:

s=set([iterable])

S=set() creates an initially empty set.

If IT is some iterable (like a set) the s=set(IT) creates a set that is initialized with the elements of IT (for example a list)

The set provides the following operations (among others):

add(elem)
Add element elem to the set.
remove(elem)
Remove element elem from the set. Raises KeyError if elem is not contained in the set.
discard(elem)
Remove element elem from the set if it is present.
pop()
Remove and return an arbitrary element from the set. Raises KeyError if the set is empty.
clear()
Remove all elements from the set.
len(s)
Return the cardinality (number of elements) of set s.
x in s
Test x for membership in s.
x not in s
Test x for non-membership in s.

Problem:

Ask the user for three integer **n** and **m and k, where k>m*n**.

Create a two dimensional table (list of lists) x of size n*m. Populate the list with n*m **unique random integers** in the range 1through k.

Problem:

Write a function get_row(x,i) which **takes a square matrix** (=table=list of lists) and **returns a list** containing the ith row of matrix X.

Problem:

Write a function get_col(x,i) which **takes a square matrix** (=table=list of lists) and **returns a list** containing the ith column of matrix X.

Problem:

Write a function to calculate and return the sum of all the elements on the "main diagonal" of the square two dimensional list x passed to it as an argument. This is the diagonal going from the upper left to the bottom right.

Problem:

Like the problem above, calculate and return the diagonal sum, but for the elements along the diagonal going from the top right to the lower left.

List Comprehensions

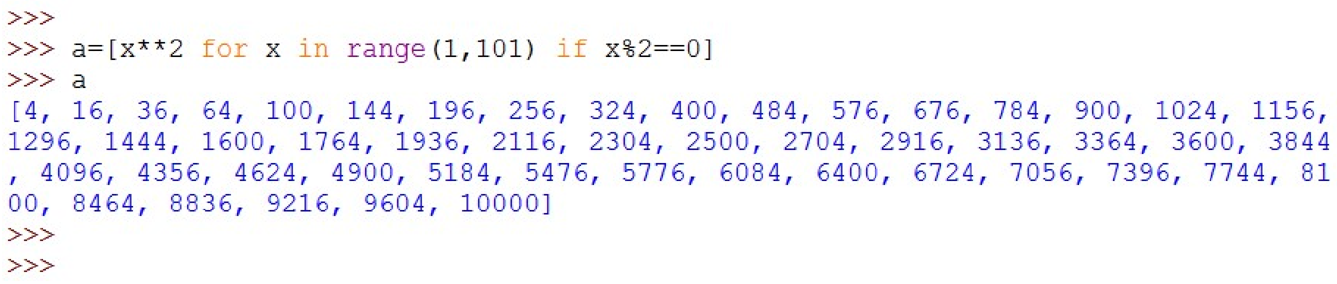
A list comprehension is a very compact and useful notation that Python provides for **creating lists**.

It is very similar to the notation that is uses in math for specifying sets. For example it is pretty clear what this means:

$$\{x^2 : \ 1 \leq x \leq 100 \text{ , if x is even}\}$$

It is the set of the squares of the even numbers between 1 and 100.

Here is a list comprehension that creates a list with the same numbers:



```
>>>
>>> a=[x**2 for x in range(1,101) if x%2==0]
>>> a
[4, 16, 36, 64, 100, 144, 196, 256, 324, 400, 484, 576, 676, 784, 900, 1024, 1156,
1296, 1444, 1600, 1764, 1936, 2116, 2304, 2500, 2704, 2916, 3136, 3364, 3600, 3844
, 4096, 4356, 4624, 4900, 5184, 5476, 5776, 6084, 6400, 6724, 7056, 7396, 7744, 81
00, 8464, 8836, 9216, 9604, 10000]
>>>
>>>
```

Notice that this is equivalent to the following:

```
a=[]
for x in range(1,101):
   if x%2==0:
      a.append(x)
```

The list comprehension notation is more compact and much clearer (once you get used to it) than the equivalent code above

The general form has three components:

[expression using a value from an iterable   for iterable   if conditions]

(1)   (2)   (3)

Some more examples:

Example:

Assume that we have a function prime(i) which returns True if i is s prime number and False otherwise.

Create a list of all the primes between 2 and 120.

```
y = [  k     for k in range(2, 121)    if prime(k)    ]
```

Example:

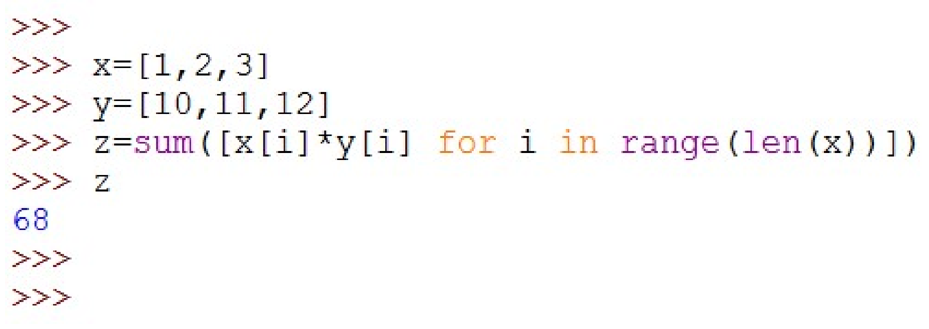
Let x and y be two lists of numbers, both of length n. To get the <u>**dot product**</u> of x and y we do the following:

- form the pairs x[i]*y[i] for $0 \le i < n$
- sum up all the pairs.

For example:

x=[1,2,3] and y=[10,11,12] then x·y=1*10+2*11+3*12.

We can write this as list comprehension like this:



```
>>>
>>> x=[1,2,3]
>>> y=[10,11,12]
>>> z=sum([x[i]*y[i] for i in range(len(x))])
>>> z
68
>>>
>>>
```

Example:

Let z be a list of 20 integers, and we want a **list of tuples** giving the value and position of each even number in the list. The following will do this:

```
>>> z
[13, 14, 7, 12, 11, 3, 18, 20, 5, 7, 19, 17, 10, 16, 14, 12, 5, 19, 15, 1]
>>> k=[ ( i , z[i] )  for i in range(20)    if z[i]%2==0 ]
>>> k
[(1, 14), (3, 12), (6, 18), (7, 20), (12, 10), (13, 16), (14, 14), (15, 12)]
>>>
```

Example:

Write a function get_row(x,i) which **takes a square matrix** (=table=list of lists) and **returns a list** containing the element in the ith **row** of matrix X.

Using a list comprehension we could write:

```
def get_row(x,i):
    return [ x[i][j] for j in range(len(x)) ]
```

Or …we could also write

```
def get_row(x,i):
    return x[i]
```

since the ith row of x is just the ith sublist of x.

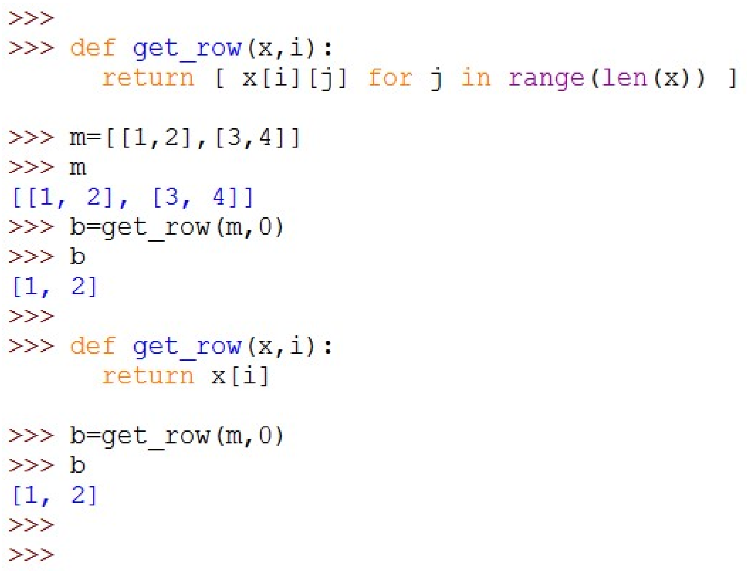
However there is an **important difference between these two implementations** of get_row().

The first version creates a new list made up of the elements of row i of matrix X.

The second one, which is much faster, just returns a reference to the ith row of matrix X.

In the following example, we call **both** versions of get_row(m,0) on the matrix **m=[[1,2],[3,4]].**

Notice that both versions return what **seems to be** the same result: [1, 2]



```
>>>
>>> def get_row(x,i):
	return [ x[i][j] for j in range(len(x)) ]

>>> m=[[1,2],[3,4]]
>>> m
[[1, 2], [3, 4]]
>>> b=get_row(m,0)
>>> b
[1, 2]
>>>
>>> def get_row(x,i):
	return x[i]

>>> b=get_row(m,0)
>>> b
[1, 2]
>>>
>>>
```

Question: What is the important **practical difference** between the two versions?

Hint: What happens when we change list b after it is returned from get_row(). Why does this happen?

Example:

Write a function get_col(x,i) which **takes a square matrix** (=table=list of lists) and **returns a list** containing the items in the ith **column** of matrix X.

```
def get_col(x,i):
    return [ x[j][i] for j in range(len(x)) ]
```

Problem:

Write a function sum_col(x,i) which **takes a square matrix** (=table=list of lists) and **the sum of** the items in the ith **column** of matrix X. Use a list comprehension.

Problem:

Write a function diag_diff(x,i) which **takes a square matrix** (=table=list of lists) and **the difference of two main diagonals of matrix X.** In other words, let d1 be the diagonal of X from upper left to bottom right, and d2 to be the diagonal from upper right to lower left. The function returns d1-d2. Use list comprehensions**.**

Problem:

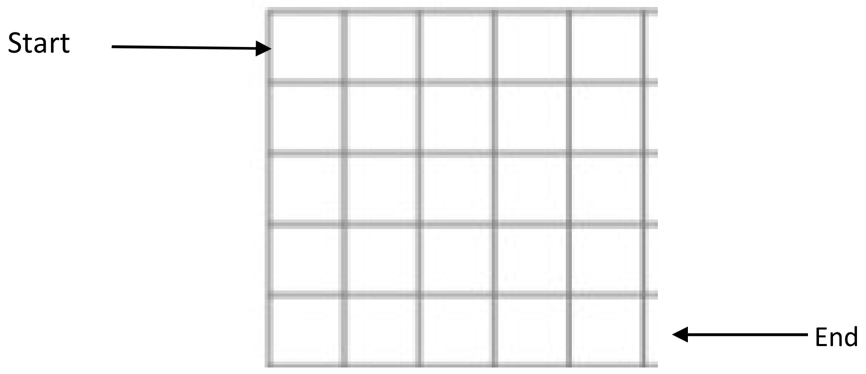
Now, here is a definition: A **saddle point** in a 2 dimensional square table is an entry in the table whose value is the minimum in its row **and** maximum in its column. In the table below, 0 is a saddle point.

2	0
0	-2

Write a function to find a saddle point in a 2 dimensional table in any two dimensional square of integers, if one exists. If a saddle point was found return a triple (value, x pos, y pos). If a saddle point was not found return "not found"

Problem:

. Consider the 5X5 square below:



We would like to have a robot travel from the start square at the upper left to the end square at the lower right. At each square the robot has only one of two moves:

- It may go one square to the right or
- It may go one square down.

Write a program that determines in how many ways may this be done? In other words, how many paths are there from start to end with each move restricted as above?

Solution:

```
#robot paths
a=5*[5*[0]]
#set up the left and top boundaries of the table (5X5 square array)
for i in range(5):
    a[i][0]=1
    a[0][i]=1

for i in range(1,5):
    for j in range(1,5):
        a[i][j]=a[i-1][j]+a[i][j-1]

print(a[4][4])
```

Here is another approach. Can you figure out how this works? What happened to the table??

```
n=int(input("Please enter the 'size' of the array, n= "))

def num_paths(n,i,j):
    if i==0 or j ==0:
        return 1
    return num_paths(n,i-1,j)+num_paths(n,i,j-1)

print(num_paths(n, n-1, n-1))
```

Problem:

Write a program to generate all the eight-digit base 8 integers from 00000000→77777777

What is a base eight integer?

The following code will do it.

```
for i0 in range(8):
    for i1 in range(8):
        for i2 in range(8):
            for i3 in range(8):
                for i4 in range(8):
                    for i5 in range(8):
                        for i6 in range(8):
                            for i7 in range(8):
                                q=[i0,i1,i2,i3,i4,i5,i6,i7]
                                print(q)
```

Problem:

Write a function get_num(n) which returns a list of length 8 representing the base eight representation of the decimal number n.

More on Strings … and Files

We have been using strings all along. Some of the things we have seen include:

- Strings are immutable.
- Strings are iterables so we can use "for".
- Since strings are sequences, we can access elements and substrings them [].
- We can concatenate strings: s1+s2.

But Python provides many many functions for working with strings. Let's check out some of the functions as described in the online documentation. The full list is in the Python Library Reference documentation in section 4.6.1.

Here are some of the most useful with definitions and examples from the Library Reference. .

`str.count`(*sub*[, *start*[, *end*]])

> Return the number of non-overlapping occurrences of substring *sub* in the range [*start*, *end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

`str.find`(*sub*[, *start*[, *end*]])

> Return the lowest index in the string where substring *sub* is found, such that *sub* is contained in the slice `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` if *sub* is not found.

`str.join`(*iterable*)

> **Return a string which is the concatenation of the strings in the *iterable*** *iterable*. A `TypeError` will be raised if there are any non-string values in *seq*, including `bytes` objects. The separator between elements is the string providing this method.

`str.replace`(*old*, *new*[, *count*])

> Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.split`([*sep*[, *maxsplit*]])

> **Return a list of the words in the string**, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified, then there is no limit on the number of splits (all possible splits are made).
>
> If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an empty string with a specified separator returns `['']`.

If *sep* is not specified or is `None`, a different splitting algorithm is applied: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace. Consequently, splitting an empty string or a string consisting of just whitespace with a `None` separator returns `[]`.

For example, `' 1  2   3  '.split()` returns `['1', '2', '3']`, and
`'  1  2   3  '.split(None, 1)` returns `['1', '2   3`

`str.splitlines`([*keepends*])

Return a list of the lines in the string, breaking at line boundaries. **Line breaks are not included in the resulting list** unless *keepends* is given and true.

`str.rstrip`([*chars*])

Return a copy of the string with trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. **The *chars* argument is not a suffix; rather, all combinations of its values are stripped:**

There are also functions lstrip() and rstrip() that act in the expected way.

`str.upper`()

Return a copy of the string converted to uppercase.

`str.lower`()

Return a copy of the string converted to lowercase

Problem:

Write a function that reverses a string. Remember, a string is not mutable! Do it two ways.

Answer:

Problem:

Write a function is_len(s,n) which returns True if string s is at least n characters long, and False otherwise.

Answer:

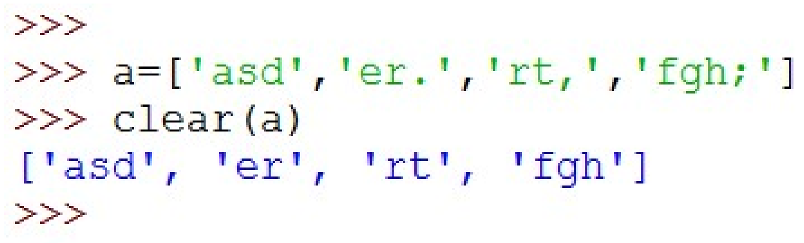
Problem:

Write a function one_upper(s) which returns True if **exactly one** character in string s is capitalized, and False otherwise. You can assume that the string s contains only alphabetic characters and no blanks. You might want to consider other string functions from the documentation.

Answer:

Problem:

Write a function clean(x) where x is a list of “words”. Each word is a string that might have either a ‘.’ ‘,’, or’;’ tacked on at the end. clear() will return a list with the original words stripped of the punctuation.



```
>>>
>>> a=['asd','er.','rt,','fgh;']
>>> clear(a)
['asd', 'er', 'rt', 'fgh']
>>>
```

Answer:

Problem:

Write a **function scrape(s)** which take a string s representing the HTML of a web page and returns a list of all links found on page s. We recognize the beginning of a link by looking for ‘http://’.

Answer:

Files

What are “files” and how are they represented on the drives?

Basic file operations

output = open(r'C:\spam', 'w')	Create output file ('w'means write)
input = open('data', 'r')	Create input file ('r'means read)
input = open('data')	Same as prior line ('r'is the default)
aString = input.read()	Read entire file into a single string
aString = input.read(N)	Read up to next Ncharacters (or bytes) into a string
aString = input.readline()	Read next line (including \nnewline) into a string
aList = input.readlines()	Read entire file into list of line strings (with \n)
output.write(aString)	Write a string of characters (or bytes) into file
output.writelines(aList)	Write all line strings in a list into file
print(value, …file='filename')	**Write to file "filename" instead of to the screen**
output.close()	Manual close (done for you when file is collected)
output.flush()	Flush output buffer to disk without closing any File.
seek(N)	Change file position to offset Nfor next operation
for line in open('data'):*use line*	File iterators read line by line
open('f.txt', encoding='latin-1')	Python 3.0 Unicode text files (strstrings)
open('f.bin', 'rb')	Python 3.0 binary bytes files (bytesstrings)

Problem:

Create this file (bears.txt) in your python directory:

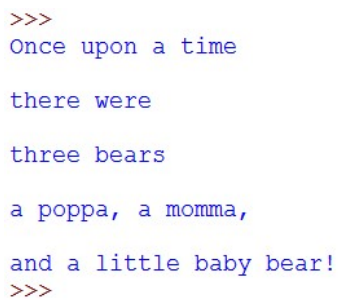
```
Once upon a time
there were
three bears,
a poppa, a momma,
and a little baby bear!
```

Read the file line by line and print it out..

Answer:

```
f=open('bears.txt')
for i in f:
     print(i)
```

produces: Why the spaces between the lines?



```
>>>
Once upon a time

there were

three bears

a poppa, a momma,

and a little baby bear!
>>>
```

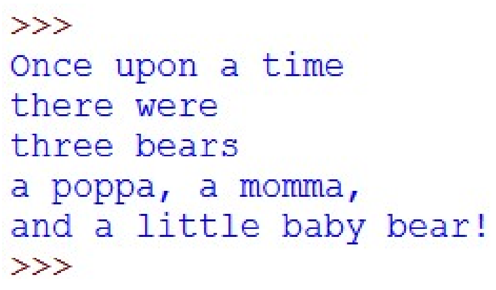
Problem:

Read the file into one string and print the string.

Answer:

```
f=open('bears.txt')
z=f.read()
print(z)
```

produces: What happened to the blank lines?



```
>>>
Once upon a time
there were
three bears
a poppa, a momma,
and a little baby bear!
>>>
```

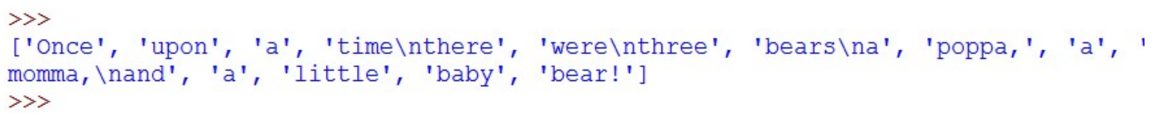
Problem:

Read the file into a string s, separate the words in to a list (use ' ' to indicate the separator between words. Print out the list.

Answer:

```
f=open('bears.txt')
z=f.read()
z=z.split(' ')
print(z)
```

produces:



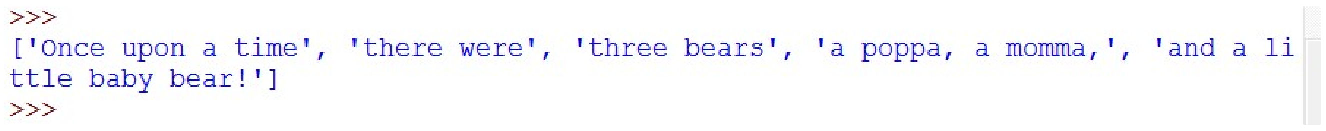
```
>>>
['Once', 'upon', 'a', 'time\nthere', 'were\nthree', 'bears\na', 'poppa,', 'a', '
momma,\nand', 'a', 'little', 'baby', 'bear!']
>>>
```

Note the '\n' , the newline character. This causes a like break.

but …

```
f=open('bears.txt')
z=f.read()
z=z.splitlines()
print(z)
```

produces:



```
>>>
['Once upon a time', 'there were', 'three bears', 'a poppa, a momma,', 'and a li
ttle baby bear!']
>>>
```

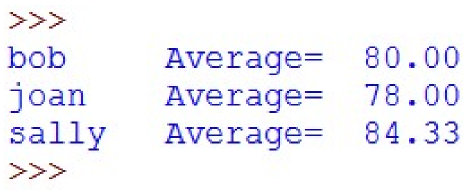
Notice that the newline characters are gone.

Problem:

Create a text file, 'grades.txt', with the following data.

```
Bob 78 98 67 77
Joan 78
Sally 90 97 77 56 88 98
```

Write a program to read this file and print out the students name followed by their average on all exams.



```
>>>
bob     Average=  80.00
joan    Average=  78.00
sally   Average=  84.33
>>>
```

Answer:

```
#Student averages
f=open('grades.txt')
for i in f:
  s=i.split()
  average=sum([int(s[i]) for i in range(1,len(s))])/(len(s)-1)
  print(format(s[0],'<7s'),'Average= ',format(average,'.2f'))
```

Problem:

Modify the above program so that it writes the result to a file called averages.

Answer:

Dictionaries

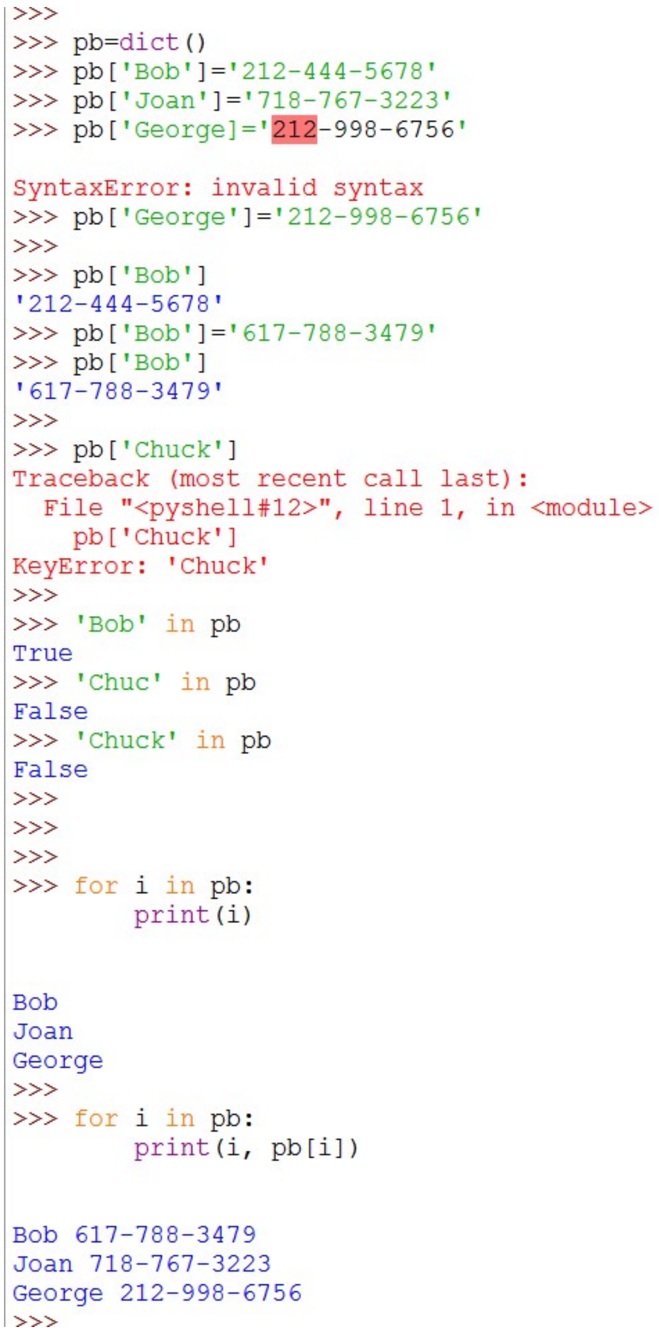
Think of an on-line dictionary. You type in a word and the dictionary returns one or more meanings of the word you entered.

We can think of the dictionary as a "list" that is indexed by the "word" whose definition you seek and the value that is returned is the set of meaning associated with that word.

Or …

Think of an on-line phone book. You time in the name of the person whose phone number you want and the phone book app returns the associated number.

We can model the above in Python using the **dictionary**.



```
>>>
>>> pb=dict()
>>> pb['Bob']='212-444-5678'
>>> pb['Joan']='718-767-3223'
>>> pb['George]='212-998-6756'

SyntaxError: invalid syntax
>>> pb['George']='212-998-6756'
>>>
>>> pb['Bob']
'212-444-5678'
>>> pb['Bob']='617-788-3479'
>>> pb['Bob']
'617-788-3479'
>>>
>>> pb['Chuck']
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    pb['Chuck']
KeyError: 'Chuck'
>>>
>>> 'Bob' in pb
True
>>> 'Chuc' in pb
False
>>> 'Chuck' in pb
False
>>>
>>>
>>>
>>> for i in pb:
        print(i)


Bob
Joan
George
>>>
>>> for i in pb:
        print(i, pb[i])


Bob 617-788-3479
Joan 718-767-3223
George 212-998-6756
>>>
```

Here are some of the dictionary methods:

`dict`()

Create a new dictionary.

`len(d)`

Return the number of items in the dictionary *d*.

`d[key]`

Return the item of *d* with key *key*. Raises a `KeyError` if *key* is not in the map.

`d[key] = value`

Set `d[key]` to *value*.

`del d[key]`

Remove `d[key]` from *d*. Raises a `KeyError` if *key* is not in the map.

`key in d`

Return `True` if *d* has a key *key*, else `False`.

`key not in d`

Equivalent to `not key in d`.

`clear`()

Remove all items from the dictionary.

`copy`()

Return a shallow copy of the dictionary.

Create a new dictionary with keys from *seq* and values set to *value*.

`items`()

Return a new view of the dictionary's items (`(key, value)` pairs). See below for documentation of view objects.

`keys`()

Return a new view of the dictionary's keys. See below for documentation of view objects.

`pop`(*key*[, *default*])

If *key* is in the dictionary, remove it and return its value, else return *default*. If *default* is not given and *key* is not in the dictionary, a `KeyError` is raised.

`values()`
Return a new view of the dictionary's values

There are additional methods. Check out the Python on-line documentation.

Problem:

What does the following program do?

```
def squish(x):
    result=[]
    count=0

    a=x[0]
    for i in range(len(x)):

        if x[i]==a:
            count+=1
        else:
            result.append((count,a))
            count=1
            a=x[i]
    result.append((count,a))

    return(result)

def clean(x):
    y=''
    s=['.',',','-']
    for i in x:
        if i in s:
            continue
        else:
            y+=i
    return y

d={}
ln=0
for line in open('GB.txt'):
    ln+=1
    line=clean(line)
    l=line.split()
    for word in l:
        if word not in d:
            d[word]=[]
            d[word].append(1)
            d[word].append([ln])
        else:
            d[word][0]+=1
            d[word][1].append(ln)

print('list of d')
ld=list(d)
ld.sort()
for k in ld:
    print(k,d[k])
    #print(k,d[k][0],squish(d[k][1]))
```

Problem:

What does the following program do?

```
import pickle

def signature(w):
   w1=list(w)
   w1.sort()
   w1=''.join(w1)
   return w1

#create or load?
mode=input("Create or Load C or L: ")
print()
if mode.upper()=='C':
# create a "Scrabble Dictionary"
   d={}
   print('Creating dictionary ... please wait.')
   f = open('C:/python32/six letter words.txt', 'r')
   print()
   sl = f.read()

   z=sl.split(' ')
   print()
   for w in z:
      sig=signature(w)
      if sig not in d:
         d[sig]=[]
         d[sig].append(w)
      else:
         d[sig].append(w)
   f.close()
else:
   print('Unpickling dictionary ... please wait.')
   f=open('slwords','rb')
   d=pickle.load(f)

word=input("Please enter word: ")
print()

while word!='done':
   if len(word)!=6:
      print("word not 6 chars")
   else:
      word=word.upper()
      word=signature(word)

      if word in d:
            print(d[word])
        else:
            print(word,' not found.')
    word=input("Please enter word: ")
    print()

f=open('slwords','wb')
print('Pickling ... please wait.')
pickle.dump(d,f)
f.close()
```