Lab 7 – Iterating

For and range

The **for** loop in Python has the ability to iterate over the items of any *sequence*, such as a list or a string.

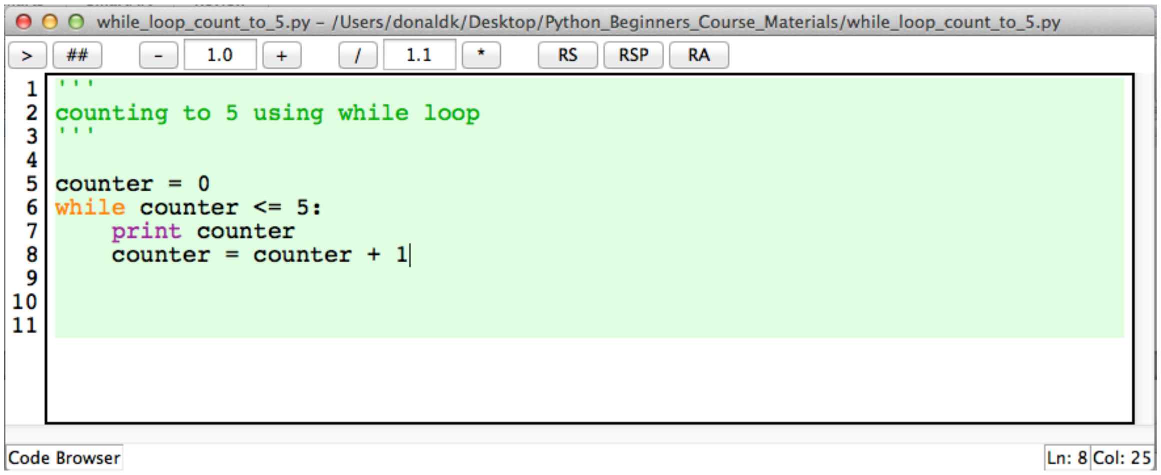
Syntax:

The syntax of a **for** loop look is as follows:

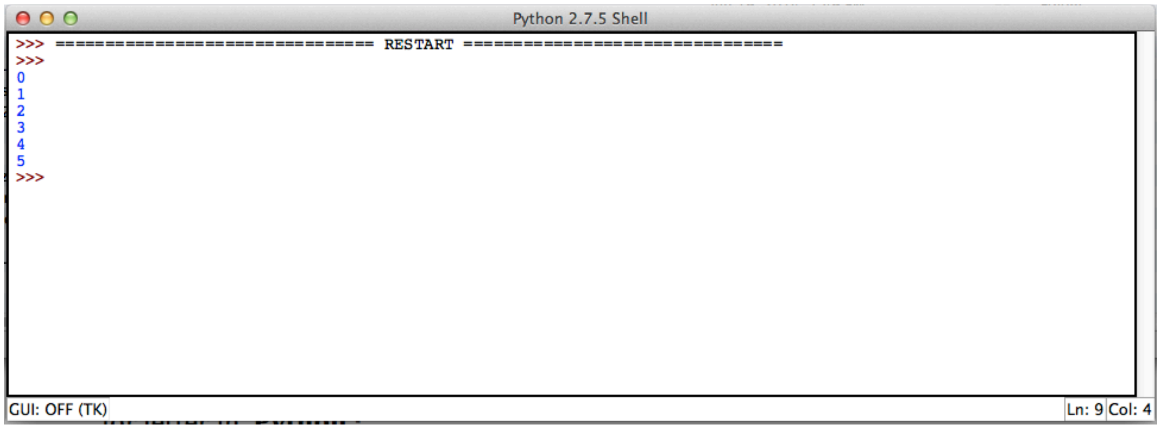
```
for iterating_var in sequence:
   statements(s)
```

The first item in the sequence is assigned to the iterating variable *iterating_var*.
Next, the statements block is executed.
Each item in the list is assigned to *iterating_var*, and the statement(s) block is executed until the entire sequence is exhausted.

We already know the **while** loop. We can do the following with while:


while_loop_count_to_5.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/while_loop_count_to_5.py
> ## - 1.0 + / 1.1 * RS RSP RA
Code Browser
Ln: 8 Col: 25

```
'''
counting to 5 using while loop
'''

counter = 0
while counter <= 5:
    print counter
    counter = counter + 1
```



```
>>> ================================ RESTART ================================
>>>
0
1
2
3
4
5
>>>
```

We can do this with a **for** loop, but we need to know the built-in **range( )** function first.

Python's range() Parameters:

The **range( )** function has two sets of parameters, as follows:

range(stop)

- stop: The number of numbers (integers) to generate, starting from zero.

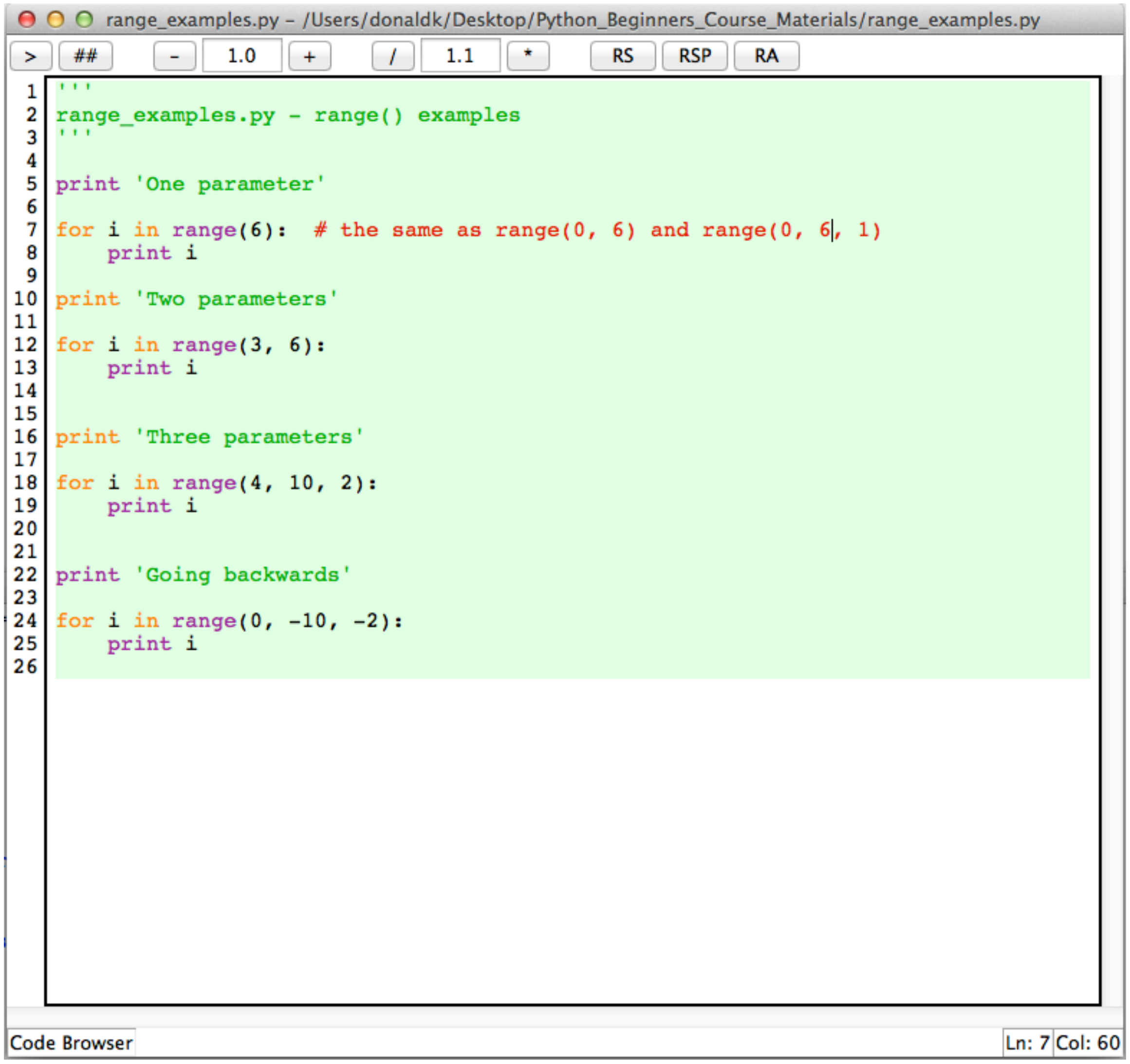
range([start], stop[, step])

- start: The starting number of the sequence.
- stop: The number of numbers (integers) to generate, starting from zero.
- step: The difference between each number in the sequence.

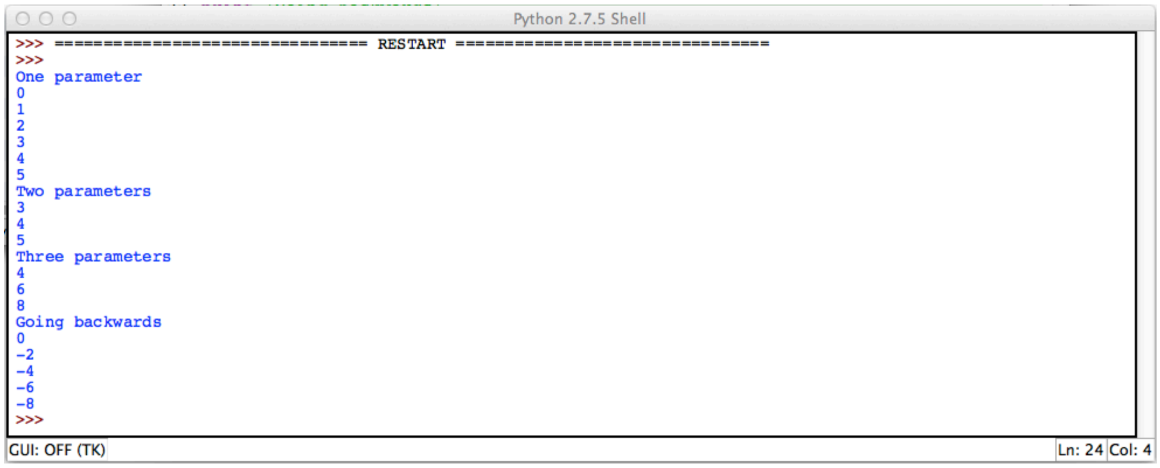
Note that:

- All parameters must be integers.
- All parameters can be positive or negative.

- The stop parameter is **not** the number the function will stop on. It specifies that it will stop on the Nth number produced, where stop is the Nth number.


range_examples.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/range_examples.py
> ## - 1.0 + / 1.1 * RS RSP RA
Code Browser
Ln: 7 Col: 60

```python
'''
range_examples.py - range() examples
'''

print 'One parameter'

for i in range(6):  # the same as range(0, 6) and range(0, 6, 1)
    print i

print 'Two parameters'

for i in range(3, 6):
    print i


print 'Three parameters'

for i in range(4, 10, 2):
    print i


print 'Going backwards'

for i in range(0, -10, -2):
    print i

```



```
>>> ================================ RESTART ================================
>>> 
One parameter
0
1
2
3
4
5
Two parameters
3
4
5
Three parameters
4
6
8
Going backwards
0
-2
-4
-6
-8
>>> 
```

Lab 7 – Exercises:

1. Using the built-in range( ) function generate the following output:
    a. [5, 10, 15, 20]
    b. [5, 105, 205]
    c. [-1, -21, -41, -61, -81]

2. Print the following result using a for loop:

    3 2 1 CONTACT

3. We haven't really covered this yet, but try this in the interpreter:

    ```
    for char in 'Python':
       print char
    ```

    Also try:

    ```
    list_1 = ['GO', 'PACK', 'GO!']
    for chant in list_1:
       print chant,
    ```

    Next lab we will be covering **sequences**. *Stings*, *lists* and *tuples* are sequences and can be iterated over using **for** and **in**

    The list can contain any type of object:

    ```
    for item in [12, 'Rodgers', [1929, 1930, 1931, 1936, 1939, 1944, 1961, 1962,
    1965, 1966, 1967, 1996, 2010]]:
       print item
    ```

4. Write a Python script that will print out the lyrics of "99 Bottles of Beer on the Wall".

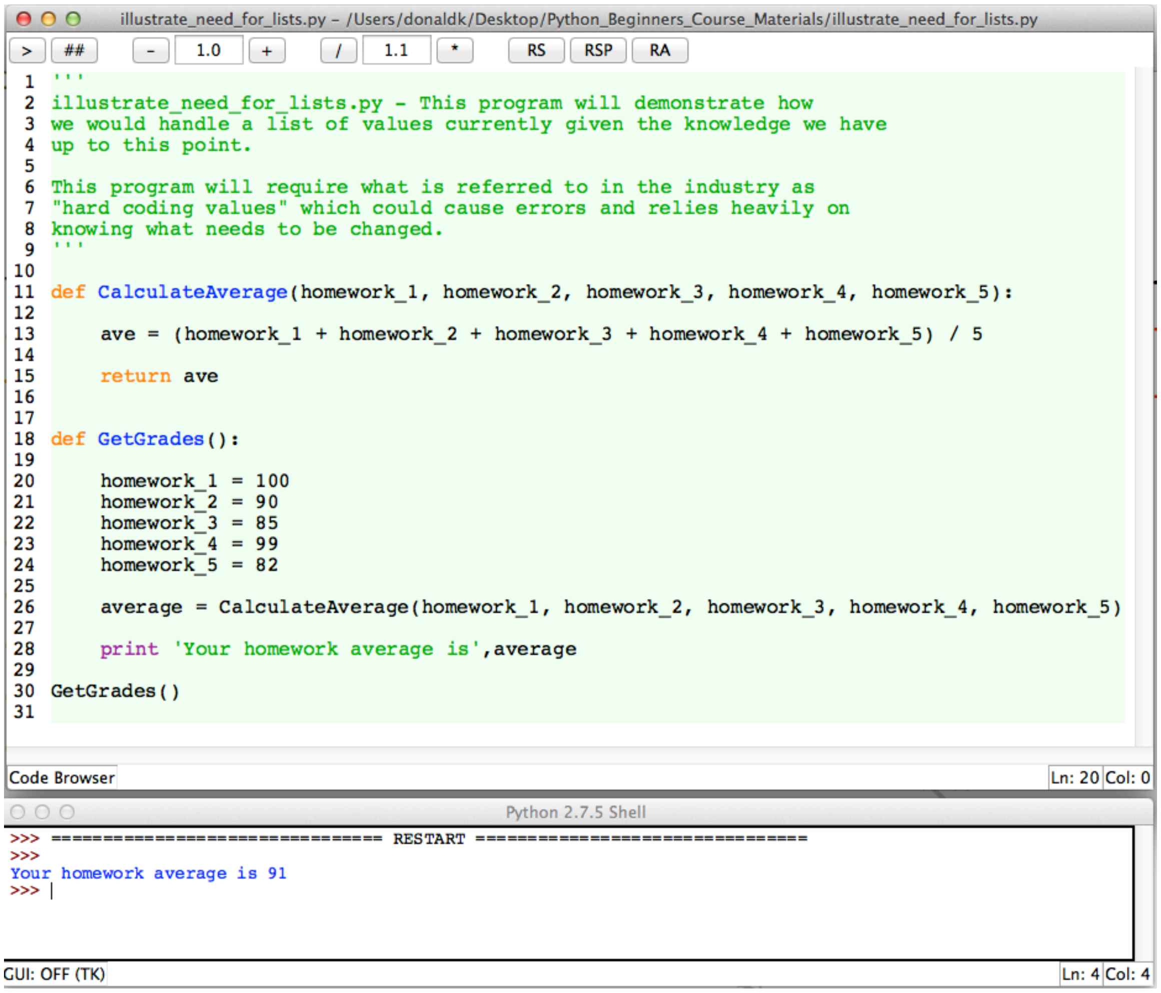
    Song goes like this:
    99 bottles of beer on the wall, 99 bottles of beer!
    So take it down, pass it around, 98 more bottles of beer on the wall!

5. Write a python script that will compute 10!. This is 10 factorial. Remember: 10 factorial is 10 * 9 * 8 * 7 * . . . . . N

Lab 8 – Sequences

sequences
enumerate
indexing

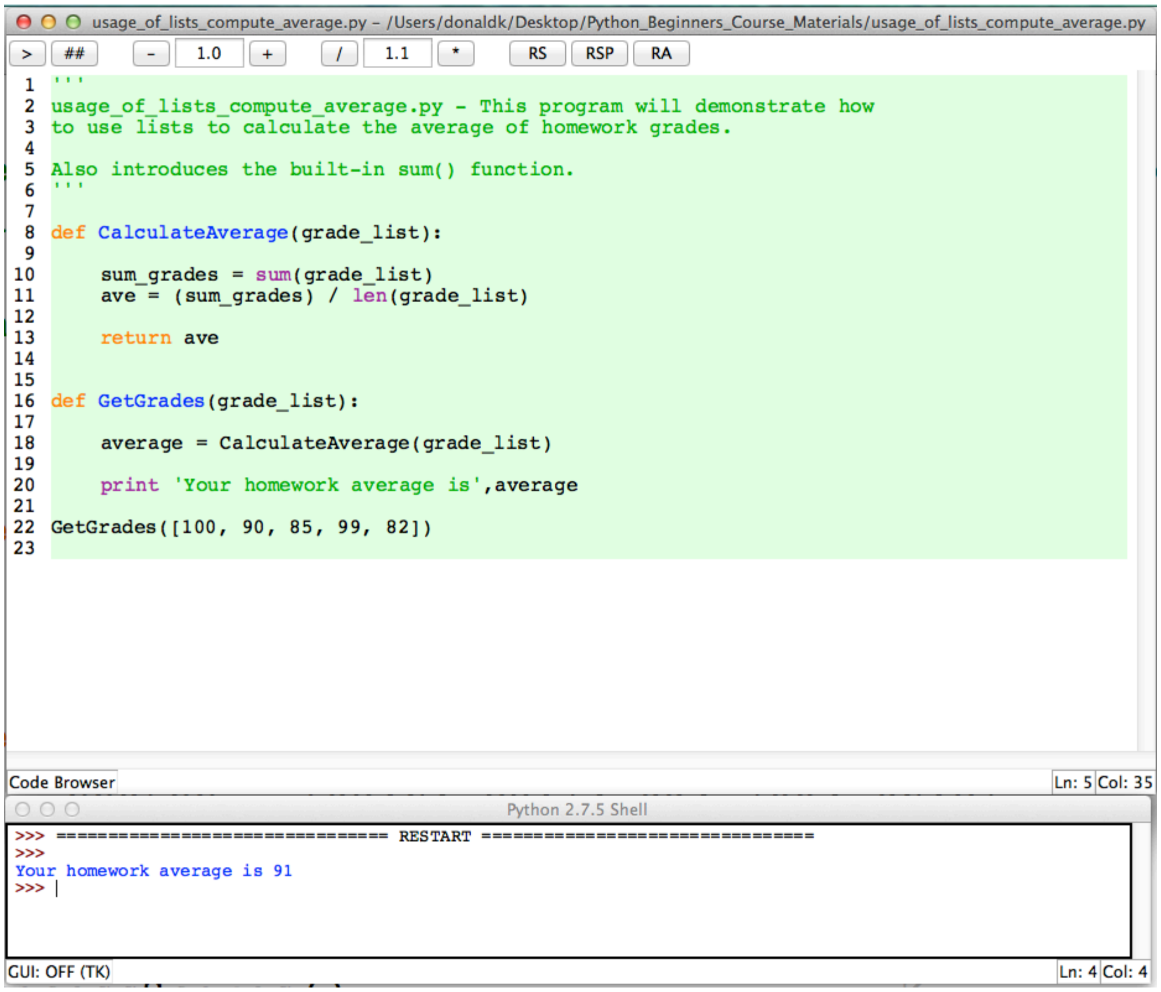
If we need to compute the average of five numbers we currently could do it by writing a small program that looks like this:


illustrate_need_for_lists.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/illustrate_need_for_lists.py
Code Browser
Ln: 20 Col: 0
Python 2.7.5 Shell
GUI: OFF (TK)
Ln: 4 Col: 4

```python
'''
illustrate_need_for_lists.py - This program will demonstrate how
we would handle a list of values currently given the knowledge we have
up to this point.

This program will require what is referred to in the industry as
"hard coding values" which could cause errors and relies heavily on
knowing what needs to be changed.
'''

def CalculateAverage(homework_1, homework_2, homework_3, homework_4, homework_5):

    ave = (homework_1 + homework_2 + homework_3 + homework_4 + homework_5) / 5

    return ave


def GetGrades():

    homework_1 = 100
    homework_2 = 90
    homework_3 = 85
    homework_4 = 99
    homework_5 = 82

    average = CalculateAverage(homework_1, homework_2, homework_3, homework_4, homework_5)

    print 'Your homework average is',average

GetGrades()

```

```
>>> ================================ RESTART ================================
>>>
Your homework average is 91
>>> |
```

Can you think of why this is probably not the best way to implement calculation of student averages this way?


usage_of_lists_compute_average.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/usage_of_lists_compute_average.py
Code Browser
Ln: 5 Col: 35
Python 2.7.5 Shell
GUI: OFF (TK)
Ln: 4 Col: 4

```python
'''
usage_of_lists_compute_average.py - This program will demonstrate how
to use lists to calculate the average of homework grades.

Also introduces the built-in sum() function.
'''

def CalculateAverage(grade_list):

    sum_grades = sum(grade_list)
    ave = (sum_grades) / len(grade_list)

    return ave


def GetGrades(grade_list):

    average = CalculateAverage(grade_list)

    print 'Your homework average is',average

GetGrades([100, 90, 85, 99, 82])
```

```
>>> ================================ RESTART ================================
>>>
Your homework average is 91
>>> |
```

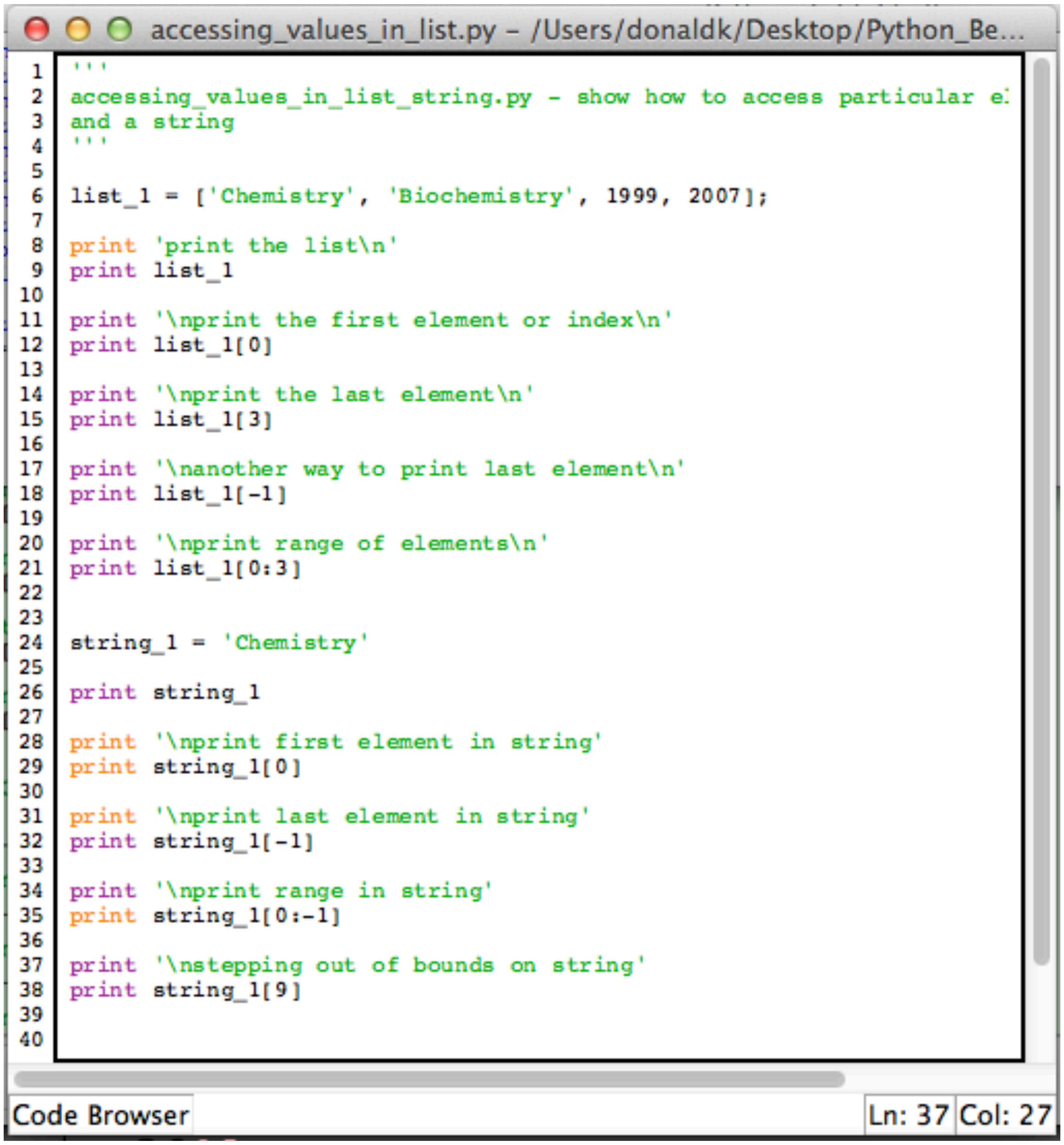
Like string indices, list indices start at 0.

Accessing Values in Lists:

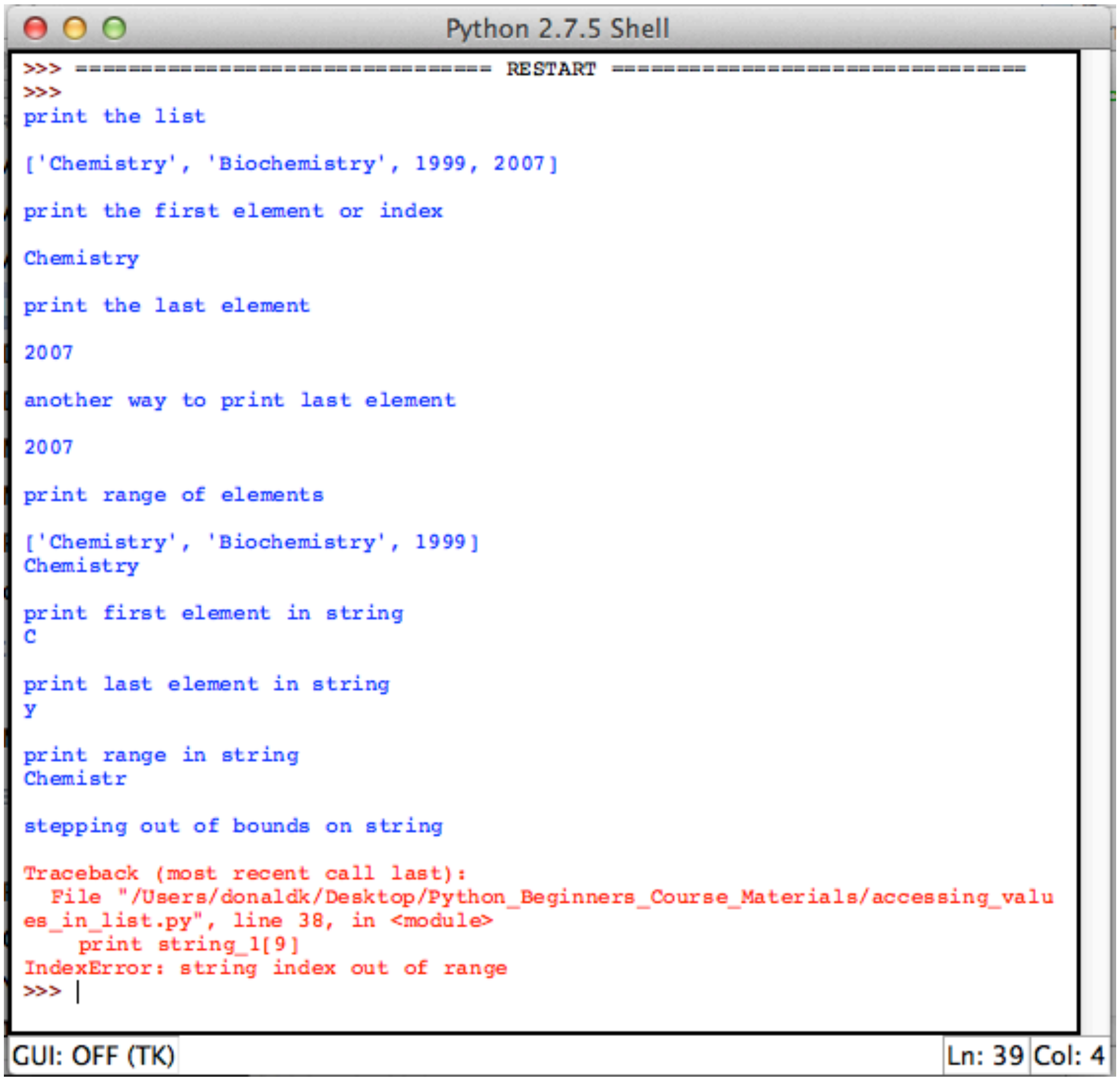
To access values in lists, use the square brackets for slicing along with the **index** or **indices** to obtain value available at that index.

[i] accesses a particular element of a list or a string: **i** stands for *index* which represents the position in the sequence.

Following is a simple example with a list and a string:

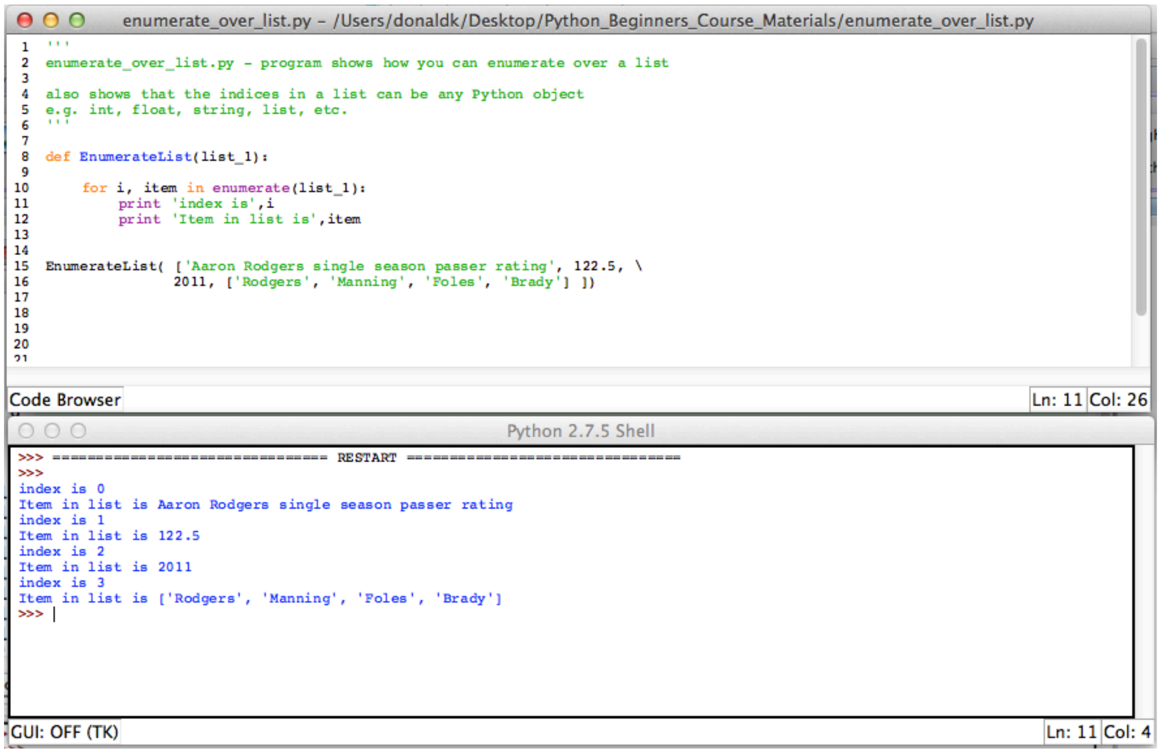

accessing_values_in_list.py - /Users/donaldk/Desktop/Python_Be...

```
'''
accessing_values_in_list_string.py - show how to access particular el
and a string
'''

list_1 = ['Chemistry', 'Biochemistry', 1999, 2007];

print 'print the list\n'
print list_1

print '\nprint the first element or index\n'
print list_1[0]

print '\nprint the last element\n'
print list_1[3]

print '\nanother way to print last element\n'
print list_1[-1]

print '\nprint range of elements\n'
print list_1[0:3]


string_1 = 'Chemistry'

print string_1

print '\nprint first element in string'
print string_1[0]

print '\nprint last element in string'
print string_1[-1]

print '\nprint range in string'
print string_1[0:-1]

print '\nstepping out of bounds on string'
print string_1[9]


```



```
>>> ============================== RESTART ==============================
>>> 
print the list

['Chemistry', 'Biochemistry', 1999, 2007]

print the first element or index

Chemistry

print the last element

2007

another way to print last element

2007

print range of elements

['Chemistry', 'Biochemistry', 1999]
Chemistry

print first element in string
C

print last element in string
y

print range in string
Chemistr

stepping out of bounds on string

Traceback (most recent call last):
  File "/Users/donaldk/Desktop/Python_Beginners_Course_Materials/accessing_valu
es_in_list.py", line 38, in <module>
    print string_1[9]
IndexError: string index out of range
>>> |
```

Lists can also be enumerated over using built-in function **enumerate( )**:


enumerate_over_list.py - /Users/donaldk/Desktop/Python_Beginners_Course_Materials/enumerate_over_list.py

```
'''
enumerate_over_list.py - program shows how you can enumerate over a list

also shows that the indices in a list can be any Python object
e.g. int, float, string, list, etc.
'''

def EnumerateList(list_1):

    for i, item in enumerate(list_1):
        print 'index is',i
        print 'Item in list is',item


EnumerateList( ['Aaron Rodgers single season passer rating', 122.5, \
               2011, ['Rodgers', 'Manning', 'Foles', 'Brady'] ])
```

Code Browser | Ln: 11 Col: 26

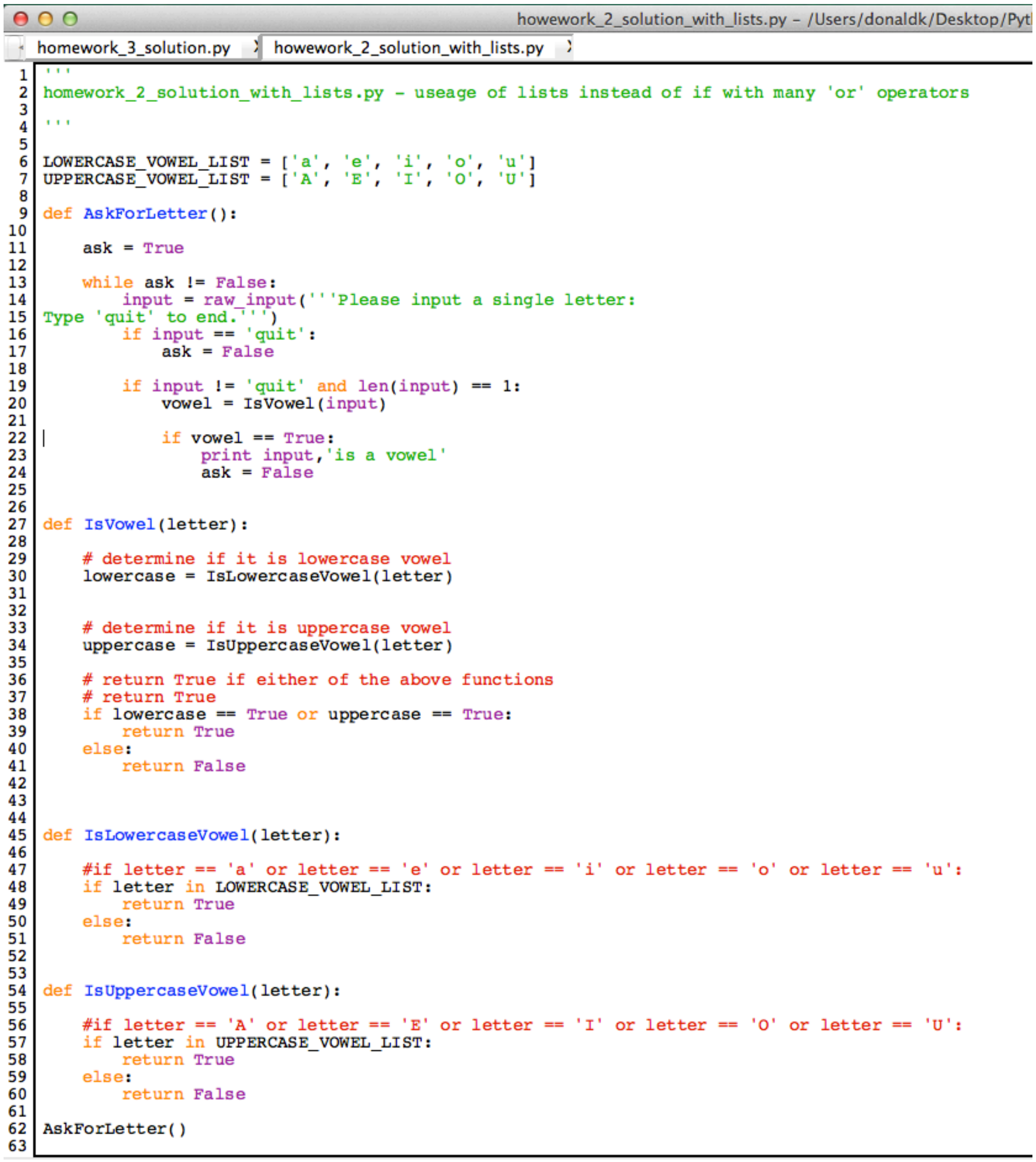
Python 2.7.5 Shell

```
>>> ================================ RESTART ================================
>>>
index is 0
Item in list is Aaron Rodgers single season passer rating
index is 1
Item in list is 122.5
index is 2
Item in list is 2011
index is 3
Item in list is ['Rodgers', 'Manning', 'Foles', 'Brady']
>>> |
```

GUI: OFF (TK) | Ln: 11 Col: 4

Donald Keidel, Ph.D. 2016

Now that we know about lists and how to determine if value is in list, we can re-write Homework 2 and use lists for the uppercase and lowercase vowels.



howework_2_solution_with_lists.py – /Users/donaldk/Desktop/Pyt

homework_3_solution.py | howework_2_solution_with_lists.py

```
'''
homework_2_solution_with_lists.py - useage of lists instead of if with many 'or' operators

'''

LOWERCASE_VOWEL_LIST = ['a', 'e', 'i', 'o', 'u']
UPPERCASE_VOWEL_LIST = ['A', 'E', 'I', 'O', 'U']

def AskForLetter():

    ask = True

    while ask != False:
        input = raw_input('''Please input a single letter:
Type 'quit' to end.''')
        if input == 'quit':
            ask = False

        if input != 'quit' and len(input) == 1:
            vowel = IsVowel(input)

            if vowel == True:
                print input,'is a vowel'
                ask = False


def IsVowel(letter):

    # determine if it is lowercase vowel
    lowercase = IsLowercaseVowel(letter)


    # determine if it is uppercase vowel
    uppercase = IsUppercaseVowel(letter)

    # return True if either of the above functions
    # return True
    if lowercase == True or uppercase == True:
        return True
    else:
        return False



def IsLowercaseVowel(letter):

    #if letter == 'a' or letter == 'e' or letter == 'i' or letter == 'o' or letter == 'u':
    if letter in LOWERCASE_VOWEL_LIST:
        return True
    else:
        return False


def IsUppercaseVowel(letter):

    #if letter == 'A' or letter == 'E' or letter == 'I' or letter == 'O' or letter == 'U':
    if letter in UPPERCASE_VOWEL_LIST:
        return True
    else:
        return False

AskForLetter()
```

Lab 8 – Exercises:

1. Write a Python program that will ask the user for the ages of individuals in your family. Then write functions that will compute the minimum age (built-in function min( )), maximum age (built-in max( )), calculate the average, append new ages to list (built-in function append( )), and sort the list (built-in sorted( )).

2. Given the following lists:

   list_1 = [1, 3, 5, 7, 9, 11]
   list_2 = [2, 4, 6, 8, 10, 12]

   Write functions that do the following:
   1. Update the element 5 in list_1 to 55
   2. Delete element with value 10 in list_2
   3. Concatenate the two lists together

   You will need to use the built-in methods for list called list.index( ) and list.remove( ). Do a help on this to determine how to use it in the python shell.

   This exercise will also illustrate how Python lists are mutable.

3. Plot a histogram graph from a list of numbers, with each number in the list on its own line. For instance, if you start with numbers like this:

   vals = [0, 2, 4, 8, 16, 18, 17, 14, 9, 7, 4, 2, 1]
   you might plot something like this:

```
0
2 **
4 ****
8 ********
16 ****************
18 ******************
17 *****************
14 **************
9 *********
7 *******
4 ****
2 **
1 *
```

Homework 4:

Print out a copy of your homework and bring it to class. Make sure to include the output.

Write a program that will ask the user to input the names of cities they would like to visit. This means that the program should ask the user first how many cites they want to input (function – AskForNumberCities). Then the program should allow the user to input this and only this number of cities (function - AskForCityName). If they input the same city name you should not count that as one of the cities. You only need to consider cities spelled EXACTLY the same way (e.g. Haifa and Haifa). Do not worry about upper and lowercase characters in city names.
After all the cities have been collected the program should then print a sentence that looks exactly like the following (function - PrintFirstCitySentence):

"You would like to visit Tel Aviv as city 1 and Haifa as city 2 and Negev as city 3 on your trip."

Next the program will take this sentence string and add 1 to each city number and then output the string with the changes (function – PrintAddOneCityNumSentence). For example,

"You would like to visit Tel Aviv as city 2 and Haifa as city 3 and Negev as city 4 on your trip."

NOTE: To do this I am requiring that you take the first sentence, split this sentence into a list, use isdigit() to determine if element of list is digit, add one to these elements, and join the new list elements together using join(). Since we just learned how to use the for loop and for loop with enumerate I am also requiring that you loop using for loops to show that you know how to use them. You may also use while loops where needed.

Remember, to find these built-in methods for a string type do the following:

```
>>> string_1 = 'hello'
>>> dir(string_1)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__',
'__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '_formatter_field_name_split', '_formatter_parser', 'capitalize',
'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```