Part 1: Primer

1: Lists

1) Make a list with the spelled-out number strings ‘one’, ‘two’, ‘three’, ‘four’, and ‘five’ in that order and call it myList.

2) Remove ‘three’ from the list using positional indexing.

3) Check if ‘four’ is in the list.

4) Append ‘six’ to the end of the list, then print the length of the list.

5) Print the contents of the list, but also next to each item print the length of the string (e.g. one is 3, four is 4) using a for loop.

6) Create a list only of the lengths of the strings and show your result. You can use the loop before to fill the list.

In [1]:
```
myList = ['one', 'two', 'three', 'four', 'five']
print(myList)
```
```
['one', 'two', 'three', 'four', 'five']
```

In [2]:
```
myList.remove('three')
print(myList)
```
```
['one', 'two', 'four', 'five']
```

In [3]:
```
'four' in myList
```
Out[3]:
```
True
```

In [4]:
```
myList.append('siz')
print(len(myList))
```
```
5
```

In [5]:
```
for i in myList:
    print(i, 'value is', len(i))
```
```
one value is 3
two value is 3
four value is 4
five value is 4
siz value is 3
```

In [6]:
```
res = []
for i in myList:
    res.append(len(i))
print(res)
```

```
[3, 3, 4, 4, 3]
```

2: Dictionaries

1) Make a dictionary with the keys be English words as below, and the values be the translation. You can use this language example (German) or choose your own. Note: you need to make sure all of these words are represented as strings, in quotes.

apple - Apfel

apples - Äpfel

I - Ich

and - und

like - mag

strawberries - Erdbeeren

2) Use the dictionary to look up the translation for ‘apple’ and ‘like’.

3) Make a variable var with the string “I like apples and strawberries”

4) Now create a list from var with each word a separate item (this is a string split operation).

5) Iterate through the list you’ve created and replace any word in your dictionary with the translation.

6) Now take your new list and turn it into a string with spaces between the words.

In [10]:
```
dictio = {}
dictio = {'apple': 'Apfel','apples':'Apfel','I':'Ich', 'and':'und','like':'
print(dictio)
```

```
{'apple': 'Apfel', 'apples': 'Apfel', 'I': 'Ich', 'and': 'und', 'like':
'mag', 'strawberries': 'Erdbeeren'}
```

In [11]:
```
print(dictio['apple'])
print(dictio['like'])
```

```
Apfel
mag
```

In [12]:
```
var="I like apples and strawberries"
```

In [16]:
```
mylist = list(var.split(" "))
print(mylist)
```

```
['I', 'like', 'apples', 'and', 'strawberries']
```

In [17]:
```
for i in range(len(mylist)):
    mylist[i] = dictio[mylist[i]]
print(mylist)
```

```
['Ich', 'mag', 'Apfel', 'und', 'Erdbeeren']
```

In [18]:
```
var2=" ".join(mylist)
var2
```

Out[18]:
```
'Ich mag Apfel und Erdbeeren'
```

3: Arrays

1) Create an array of zeros of size 8 x 8 and print the data type of the array.

2) Fill the array with the numbers 1 to 64 first by row, then by column. You may want to use a for loop inside a for loop to do this.

3) Transpose the array.

4) Print only the top 4 rows and columns.

5) Make a 1D array out of your 2D array with the numbers 1 to 64 in order (note the column vs row issue, you may need transposes.)

6) Now take that 1D array you made from before and reshape it back to the original 2D array.

In [19]:
```
import numpy as np
arr = np.ones((8,8))
type(arr)
```

Out[19]:
```
numpy.ndarray
```

In [20]:
```
for i in range(8):
    for j in range(8):
        arr[i,j]=8*i+j+1;
print(arr)
```

```
[[ 1.  2.  3.  4.  5.  6.  7.  8.]
 [ 9. 10. 11. 12. 13. 14. 15. 16.]
 [17. 18. 19. 20. 21. 22. 23. 24.]
 [25. 26. 27. 28. 29. 30. 31. 32.]
 [33. 34. 35. 36. 37. 38. 39. 40.]
 [41. 42. 43. 44. 45. 46. 47. 48.]
 [49. 50. 51. 52. 53. 54. 55. 56.]
 [57. 58. 59. 60. 61. 62. 63. 64.]]
```

In [21]:
```
arr=arr.transpose()
print(arr)
```

```
[[ 1.  9. 17. 25. 33. 41. 49. 57.]
 [ 2. 10. 18. 26. 34. 42. 50. 58.]
 [ 3. 11. 19. 27. 35. 43. 51. 59.]
 [ 4. 12. 20. 28. 36. 44. 52. 60.]
 [ 5. 13. 21. 29. 37. 45. 53. 61.]
 [ 6. 14. 22. 30. 38. 46. 54. 62.]
 [ 7. 15. 23. 31. 39. 47. 55. 63.]
 [ 8. 16. 24. 32. 40. 48. 56. 64.]]
```

In [22]:
```
print(arr[:4,:4])
```

```
[[ 1.  9. 17. 25.]
 [ 2. 10. 18. 26.]
 [ 3. 11. 19. 27.]
 [ 4. 12. 20. 28.]]
```

In [23]:
```
arr1=arr.transpose().flatten()
print(arr1)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.
 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36.
 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54.
 55. 56. 57. 58. 59. 60. 61. 62. 63. 64.]
```

In [24]:
```
arr2 = arr1.reshape(8,8)
print(arr2)
```

```
[[ 1.  2.  3.  4.  5.  6.  7.  8.]
 [ 9. 10. 11. 12. 13. 14. 15. 16.]
 [17. 18. 19. 20. 21. 22. 23. 24.]
 [25. 26. 27. 28. 29. 30. 31. 32.]
 [33. 34. 35. 36. 37. 38. 39. 40.]
 [41. 42. 43. 44. 45. 46. 47. 48.]
 [49. 50. 51. 52. 53. 54. 55. 56.]
 [57. 58. 59. 60. 61. 62. 63. 64.]]
```

Part 2: Applications

1. Word Counts

Word counts are often used in text processing to automatically classify documents by topic. They are also used to automatically measure the “sentiment” by counting, for example, the number of positive or negative words used in the comment or essay. Write code to count the number of unique words in a very large string using the following steps.

1. First convert the string to a list with each word a separate item in the list. Hint: use a string split function for your language, and make sure it separates by “ ”.

2. Then use a dictionary to associate each word with a count. Note, the dictionary won't be able to increment a key unless you add it first, so you may have to check to see if it exists before setting the original count of a word to 1.
3. Print each word and its count afterwards, and test with an interesting block of text that will have multiple words counted multiple times. (Note, the words don't have to be in any particular order.) For example: "how much wood would a woodchuck chuck if a woodchuck could chuck wood"

Expected output:

how - 1

much -1

wood - 2

would - 1

a - 2

woodchuck - 2

chuck - 2

if - 1

could - 1

In [25]:
```python
str = 'how much wood would a woodchuck chuck if a woodchuck could chuck woo
wlist = str.split(' ')
print(wlist)
```

```
['how', 'much', 'wood', 'would', 'a', 'woodchuck', 'chuck', 'if', 'a', 'w
oodchuck', 'could', 'chuck', 'wood']
```

In [26]:
```python
wdict = {}
for w in wlist:
    if w in wdict:
        wdict[w] +=1
    else:
        wdict[w]=1
for w in wdict:
    print(w," - ", wdict[w])
```

```
how  -  1
much  -  1
wood  -  2
would  -  1
a  -  2
woodchuck  -  2
chuck  -  2
if  -  1
could  -  1
```

2. Adding an Array Border

As we will see later in the class, arrays can be used to represent images. In particular, black and white images can be represented by a 2D array, with one number (usually 0) representing black and another number (usually 1, but sometimes something else) representing white, and gray being everything in between. Here we just want to test your ability to handle a 2D array and add a border (filled with 0's) around it. In code this means creating an array of 0's of width+2 by height+2 size and copying the original array into the middle appropriately (Note: do not use np.pad() function).

Test with an array in the center that has non-zero elements. First print the original array, then print the new array with the added border of 0's. For example:

Expected Output:

Original array:

[[1 1 1 1]

[1 1 1 1]

[1 1 1 1]]

New array (0 on the border and 1 inside in the array):

[[0 0 0 0 0 0]

[0 1 1 1 1 0]

[0 1 1 1 1 0]

[0 1 1 1 1 0]

[0 0 0 0 0 0]]

In [27]:
```
or_arr = np.ones((3,4))
or_arr
```

Out[27]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [31]:
```
narr = np.zeros((len(or_arr)+2,len(or_arr[0])+2))
narr[1:-1,1:-1]= or_arr
narr
```

Out[31]:
```
array([[0., 0., 0., 0., 0., 0.],
       [0., 1., 1., 1., 1., 0.],
       [0., 1., 1., 1., 1., 0.],
       [0., 1., 1., 1., 1., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

In [ ]:
```

```

```
In [ ]:
```