## ***4.2 Lists***

*Creating an Empty List*

A list is a sequence of items. In python, a list is an ordered sequence of items, not necessarily of the same type, but typically, most lists contain items all of the same type. Here is how we create an empty list in python:

```
food = []
```

*Adding an Item to the End of a List*

To add something to the end of a list, we can use the append function:

```
food.append("ham")
print(food)
```

The outcome of these two lines is as follows:

```
['ham']
```

Now,  let's add a couple more items:

```
food.append("cheese")
food.append("ice cream")
print(food)
```

Which results in the following list being printed:

```
['ham', 'cheese', 'ice cream']
```

*Removing an Item from a List*

To remove an item from a list, use the remove method:

```
food.remove("ham")
print(food)
```

Specifically, this removes the FIRST instance of the item listed in the parentheses, resulting in the following output:

```
['cheese', 'ice cream']
```

To more specifically illustrate this issue, consider adding the following segment of code:

```
food.append("ice cream")
food.append("cheese")
food.append("ice cream")
food.append("cheese")
food.remove("ice cream")
print(food)
```

This results in the following output:

```
['cheese', 'ice cream', 'cheese', 'ice cream', 'cheese']
```

Note that it's now clear that the first occurrence of "ice cream" was removed.

Just like strings, we can index into a list in the exact same manner:

```
print(food[0])
print(food[1])
print(food[-1])
```

results in the following output:

```
cheese
ice cream
cheese
```

Namely, non-negative indexes count from the beginning of the string, so we first printed the first two items in the list, and negative indexes count from the end of the string, so the last line printed out the last element of the list.

*Searching for an item in a list*

In most programming languages, you are required to search for an item, one by one, in a list. In order to do this, we must know the length of the list. In python, we can use the len function to determine the length of a list, much like we used the len function to determine the length of a string. To implement this strategy in python, we might do the following:

```
item = input("What food do you want to search for?\n")
for i in range(len(food)):
    if food[i] == item:
        print("Found",item,"!",sep="")
```

In this particular code segment, we print out a statement each time we find a copy of the  item in the list. We very easily could have set a flag to keep track of the item and only made a single print statement as follows:

```
item = input("What food do you want to search for?\n")
flag = False
for i in range(len(food)):
    if food[i] == item:
        flag = True

if flag:
    print("Found ", item, "!", sep="")
else:
    print("Sorry, we did not find ", item, ".", sep="")
```

One advantage here is that we always have a single print statement with the information we care about. The first technique may produce no output, or multiple outputs. Either can easily be adjusted to count HOW many times the item appears in the list.

Python, however, makes this task even easier for us. Rather than having to run our own for loop through all of the items in the list, we can use the in operator, just as we did for strings:

```
item = input("What food do you want to search for?\n")
if item in food:
    print("Found ", item, "!", sep="")
else:
    print("Sorry, we did not find ", item, ".", sep="")
```

The in operator allows us to check to see if an item is in a list. If an instance of an object is in a list given, then the expression is evaluated as true, otherwise it's false.

Also, just like strings, we can slice a list:

```
allfood = ["ham", "turkey", "chicken", "pasta", "vegetables"]
meat = allfood[:3]
print(meat)
```

The result of this code segment is as follows:

```
['ham', 'turkey', 'chicken']
```

Thus, essentially, what we see is that many of the operations we learned on strings apply to lists as well. Programming languages tend to be designed so that once you learn some general rules and principles, you can apply those rules and principles in new, but similar situations. This makes learning a programming language much easier than a regular language, which requires much more memorization.

If we want, we can assign a particular item in a list to a new item. Using our list meat, we can do the following:

```
meat[0] = "beef"
print(meat)
```

This produces the following output:

```
['beef', 'turkey', 'chicken']
```

In addition, we can assign a slice as follows:

```
meat[:2] = ['ham', 'beef', 'pork', 'lamb']
print(meat)
```

Here, we take the slice [:2] and replace it with the contents listed above. Since we've replaced 2 items with 4, the length of the list has grown by 2. Here is the result of this code segment:

```
['ham', 'beef', 'pork', 'lamb', 'chicken']
```

*del Statement*

We can also delete an item or a slice of a list using the del statement as illustrated below:

```
>>> del meat[3]
>>> print(meat)
['ham', 'beef', 'pork', 'chicken']
>>> meat.append('fish')
>>> meat.append('lamb')
>>> print(meat)
['ham', 'beef', 'pork', 'chicken', 'fish', 'lamb']
>>> del meat[2:5]
>>> print(meat)
['ham', 'beef', 'lamb']
```

*sort and reverse Methods*

Python allows us to sort a list using the sort method. The sort method can be called on a list, and the list will be sorted according to the natural ordering of the items in the list:

```
>>> meat.sort()
>>> print(meat)
['beef', 'ham', 'lamb']
```

We can then reverse this list as follows:

```
>>> meat.reverse()
>>> print(meat)
['lamb', 'ham', 'beef']
```

*Using lists to store frequencies of items*

A compact way to store some data is as a frequency chart. For example, if we asked people how many hours of TV they watch a day, it's natural to group our data and write down how many people watch 0 hours a day, how many people watch 1 hour a day, etc. Consider the problem of reading in this information and storing it in a list of size 24, which is indexed from 0 to 23. We will assume that no one watches 24 hours of TV a day!

We first have to initialize our list as follows:

```
freq = []
```

```
for i in range(24):
    freq.append(0)
```

At this point, we are indicating that we have not yet collected any data about TV watching.

Now, we will prompt the user to enter how many people were surveyed, and this will be followed by reading in the number of hours each of these people watched.

The key logic will be as follows:

```
hrs = input("How many hours does person X watch?\n")
freq[hrs] = freq[hrs] + 1
```

The key here is that we use the number of hours watched as an index to the array. In particular, when we read that one person has watched a certain number of hours of TV a day, we simply want to increment the appropriate counter by 1. Here is the program in its entirety:

```
def main():

    freq = []
    for i in range(24):
        freq.append(0)

    numPeople = int(input("How many people were surveyed?\n"))

    for i in range(numPeople):
        hrs = int(input("How many hours did person "+(str(i+1))+" watch?\n"))
        freq[hrs] = freq[hrs] + 1

    print("Hours\tNumber of People")
    for i in range(24):
        print(i,'\t',freq[i])

main()
```

*Using lists to store letter frequencies*

Imagine we wanted to store the number of times each letter appeared in a message. To simplify our task, let’s assume all of the letters are alphabetic letters. Naturally, it seems that a frequency list of size 26 should be able to help us. We want 26 counters, each initially set to zero. Then, for each letter in the message, we can simply update the appropriate counter. It makes the most sense for index 0 to store the number of a’s, index 1 to store the number of b’s, and so on.

Internally, characters are stored as numbers, known as their Ascii values. The Ascii value of ‘a’ happens to be 97, but it’s not important to memorize this. It’s only important because solving this problem involves having to convert back and forth from letters and their associated integer values from 0 to 25, inclusive. There are two functions that will help in this task:

Given a letter, the ord function converts the letter to its corresponding Ascii value.
Given a number, the chr function converts an Ascii value to its corresponding character.

This short IDLE transcript should illustrate how both functions work:

```
>>> ord('a')
97
>>> ord('c')
99
>>> ord('j')
106
>>> chr(102)
'f'
>>> chr(97)
'a'
>>> chr(122)
'z'
```

You’ll notice that the Ascii values of each lowercase letter are in numerical order, starting at 97. The same is true of the uppercase letters, starting at 65. This, means that if we have a lowercase letter stored in a variable ch, then we can convert it to its corresponding number in between 0 and 25 as follows:

```
ord(ch) – ord('a')
```

Similarly, given a number, num, in between 0 and 25, we can convert it to the appropriate character as follows:

```
chr(num + ord('a'))
```

In essence, ord and chr are inverse functions.

In the following program we’ll ask the user to enter a sentence with lowercase letters only and we’ll print out a frequency chart of how many times each letter appears:

```
def main():

    # Set up frequency list.
    freq = []
    for i in range(26):
        freq.append(0)

    sentence = input("Please enter a sentence.\n")

    # Go through each letter.
    for i in range(len(sentence)):

        # Screen for lower case letters only.
        if sentence[i] >= 'a' and sentence[i] <= 'z':
            num = ord(sentence[i]) - ord('a')
            freq[num] = freq[num] + 1

    # Print out the frequency chart.
    print("Letter\tFrequency")
    for i in range(len(freq)):
        print(chr(i+ord('a')),'\t',freq[i])

main()
```