2 Week 2: Data Structures, If Statements and Loops

This week we are going to take a look at some of the useful data structures and functions that are available to us in python.

2.1 Lists

Last week we learned that we can store lots of different data types as variables in our program. A python list is an ordered sequence of elements. Making a list is very simple, we demonstrate it with an example.

```
#Example 2.0
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed data types
my_list = [1, "Hello", 3.4]
```

A list can contain nothing (as with the first list) or it can contain a mixture of different data types (as with the last list). Importantly, a list can also contain other lists; we call this a nested list.

```
#Example 2.1
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

2.1.1 Accessing elements

When we want to assess the contents of a list we can use square brackets to call the index; note that python indexes a list from 0, so if we want the first element of the list in *example 2.1* we would write the following:

```
#Example 2.2
my_list = ["mouse", [8, 4, 6], ['a']]
my_list[0]
----------
"mouse"
```

We can find out how many elements a list has by calling the *len()* function, this will return the number of elements in the list as an integer; lets see this in practice.

```
#Example 2.3
my_fruit_list = ['Apple', 'Banana', 'Kewi', 'Orange', 'Pineapple']
len(my_fruit_list)
----------
5
```

We can find elements at the end of a list by using negative values, so if we wanted to find the last element of the list in *example 2.1* we would write the following:

```
#Example 2.4
my_list[-1]
----------
['a']
```

2.2 Functions for Lists

As we have seen previously, the *len()* function is useful for quickly finding how many entries we have in a list. We will now introduce two more functions useful for lists.

- *max()* : The max function returns the largest element of a list of numbers or the string which would appear last if sort in alphabetical order.
- *min()*: Akin to *max()* this function returns the smallest number or the first string when they are sorted alphabetically

Lets see how to use these functions with some more examples.

```
#Example 2.5
user_names = ['Adam','Jake','Mike','Jason','Mark','Ben','Jemma','Dave']
user_age = [17,28,15,90,53,9,40,26]

print(max(user_names))
print(max(user_age))
print(min(user_names))
print(min(user_age))
----------
Mike
90
Adam
9
```

2.3 Methods for Lists

This is the first time we meet a *method* in this course. A method is a special type of function that belongs to an object. A useful method for lists is that of the append method. We call the method using the .append suffix. Let see how to append elements to a list with an example.

```
#Example 2.6
current_users = ['Adam','Jake','Mike','Jason','Mark','Ben','Jemma','Dave']
current_users.append('Kyle')
print(current_users)
current_users.append('James')
print(current_users)
----------
['Adam', 'Jake', 'Mike', 'Jason', 'Mark', 'Ben', 'Jemma', 'Dave', 'Kyle']
['Adam', 'Jake', 'Mike', 'Jason', 'Mark', 'Ben', 'Jemma', 'Dave', 'Kyle', 'James']
```

There are also other methods that can be used on lists although we don't cover them here.

2.4 The if statements.

We use conditional statements everyday! They normally have the word *if* in them. A simple example is

if it is 9:00 am, I will drink a coffee

If statements allow us to do something called flow control where we no longer execute every line of code in sequential order but we can use *logic* to decide which parts of the code get executed. Lets see how we would rewrite our coffee expression in language python would understand.

```
#Example 2.7
current_time = float(input('what time is it? : '))

if current_time == 9.00:
  print('Have some coffee!')
----------
what time is it? : 9.00 #(we inputted time of nine here)
Have some coffee!
```

Python also has the option to check if multiple expressions are true with the else-if (*elif*) command, this means we can check multiple expressions and only execute the code when they are true. We also have the else command, this code will be executed if all the other if and else-if statements have been false. Lets return to *example 2.7* to see how it works.

```
#Example 2.8
current_time = float(input('what time is it? : '))

if current_time == 9.00:
  print('Have some coffee!')
elif current_time == 10.00:
  print('Have a sandwich!')
else:
  print('Do some programming!')
----------[run 1]
what time is it? : 10.00 #(we inputted time of ten here)
Have a sandwich!

----------[run 2]
what time is it? : 12.00 #(we inputted time of twelve here)
Do some programming!
```

2.5 Loops

There are two types of loops in Python, for and while. Loops provide a method for iterating over operations when programming. We will begin by looking at the for loop.

2.6 The For Loop

The *for* loop in Python is used to loop over a sequence, for example a list; it provides a method for taking every element within a sequence and providing some form of operation on it. The general syntax for a *for* loop is given by

```
    for value in sequence:
            operations on value
```

Lets begin with an example. Recall the list of users in *example 2.6*, if we implement a for loop we can take each of the strings within the list and do some operation, for example print to the screen. This is what we do in *example 2.9.*

```
#Example 2.9
current_users = ['Adam','Jake','Mike','Jason','Mark','Ben','Jemma','Dave']

for name in current_users:
    print(name)
----------
Adam
Jake
```

```
Mike
Jason
Mark
Ben
Jemma
Dave
```

It is safe to say that the *for* loop is one of the most important tools you have in programming; it will come in very useful!

2.7 While loop

The final topic for this week is the while loop. The while loop is similar to the for loop but rather than working through a finite sequence the while loop will operate while some condition holds. The general syntax for the *while* loop is given by

```
    while condition hold:
            perform operation
```

To understand this we make use of an example in example 2.10.

```
#Example 2.10
counter = 0

while counter <= 10:
  print(counter)
  counter = counter + 1

----------

0
1
2
3
4
5
6
7
8
9
10
```