1 Python Classes - Week 5

1.1 While loops

Before reading week (a long time ago), we covered while loops

- With while loops, we were able to ensure that a condition was met before our code moved on
- However, we also saw that multiple conditions in a while statement means we can't be certain why we left the loop

```
In [ ]: # A simple while loop
        i = 0
        while i < len(numbers):
            print(numbers[i])
            i = i + 1
```

1.2 Lists

Lists were the first signficant "datastructure" we've used. We saw how lists could allow us to create a database of username/password combinations, without needing to write hundreds of if/elif/elif... statements

```
In [ ]: my_list = [1, "two", 3 , "four", 17]
```

1.3 Range function

Python's in-built range function can generate lists of numbers for us

```
In [17]: numbers = range(0,4)
         print(list(numbers))
```

1.3.1 List Slicing

We can use list slicing's colon notation, to access sub-sections of our lists

- This works like **my_list[begin : end : step_size]**

Note you don't need to specify all three of these, python will take default values:

- begin - start of list
- end - end of list
- step_size - 1 item

```
In [20]: sentence = ["What", "A", "Great", ",", "Guy","Never","Said", "Anything","Bad"]

         # Emulate the media
         out_of_context = sentence[4:8:1] # Get part of the list
         print(out_of_context)
```

['Guy', 'Never', 'Said', 'Anything']

1.3.2 List operations

We can add and multiply lists using the familiar **+** and * operators:

```
In [22]: # Multiplying a list
         my_list = ["All", "Work", "And", "No", "Play", "Makes", "Alex", "A", "Dull", "Boy"]
         new_list = my_list * 20
         print(new_list)
```

['All', 'Work', 'And', 'No', 'Play', 'Makes', 'Alex', 'A', 'Dull', 'Boy', 'All', 'Work', 'And', 'No', 'Play', 'Makes', 'Alex', 'A', 'Dull',....

```
In [23]: # Adding two lists
         list_one = ["One", "Two", "Three"]
         list_two = [4, 5, 6]
         new_list = list_one + list_two
         print(new_list)
```

['One', 'Two', 'Three', 4, 5, 6]

1.3.3 Append

The **.append()** function provides us with an easy way of adding something to the end of our list.

```
In [26]: my_list = [1, 2 ,3 ,4]
         my_list.append(7) # Counting is hard
         print(my_list)
```

[1, 2, 3, 4, 7]

1.3.4 Strings as lists of characters

Some of you may have found it strange that we can use the "in" function on strings as well as lists:

```
In [28]: # Remember this?
         if "a" in "alex":
             print("Yeehaa")
         else:
             print("Blargh >:(")
```

We know we can also do:

```
In [30]: # What about
         if "a" in ['a','l','e','x']:
             print("Strange")
         else:
             print("Don't be silly")
```

Python essentially treats strings as lists of characters!

```
In [33]: # Strings are a lists
         the_dream = "desserts"
         university = the_dream[::-1] # We can go backwards too!

         print(university)
```

stressed

```
In [ ]: # So maybe...
        my_string = "Luke is on a roll"
        daily_mail_string = my_string[0:8] + my_string[11:13] + "t" + my_string[13:]
        print(daily_mail_string) # fake news
```

Luke is a troll

2 Program Flow Pt. II

2.1 For loops

For Is a control statement which allows us to do something **for** every item in a list.
For example (heh):

```
In [52]: # A simple for loop - runs 5 times!
         for counter in range(1, 6):
             print(str(counter) + " Cookie... Nom nom" )
```

1 Cookie... Nom nom
2 Cookie... Nom nom
3 Cookie... Nom nom
4 Cookie... Nom nom
5 Cookie... Nom nom

2.1.1 For loops <3 lists

For loops allow us to access items from our lists and do stuff with them

```
In [58]: # We can use for loops to get things directly from our list
         my_list = ["Python", "classes", "ftw"]

         for item in my_list:
             print(item)
```

Python
classes
ftw

2.2 Nesting

We can put statements inside of each other, like we did with *if* in the past.

```
In [46]: # We can also use if inside for
         # Print even numbers less than 10
         for i in range(0, 20):
             if (i == 17):
                 print(str(i) + " is the best number")
```

17 is the best number

```
In [47]: # Sometimes we might want to nest for loops
         for i in range(0,5):
             row = []
             for j in range(1, i + 2):
                 row.append(j)
             print(row)
```

```
In [48]: # Beware! There's often a cleaner / faster way
         for i in range(0,5):
             row = range(1, i+2)
             print(list(row))
```

```
In [50]: # Another example - a duplicate search
         sentence = ['She', 'sells', 'sea', 'shells', 'on', 'the', 'sea', 'shore']

         for i in range(len(sentence)):
             for j in range(len(sentence)):
                 if i == j: # Don't compare to yourself
                     continue
                 elif(sentence[i] == sentence[j]):
                     print(sentence[i] + " is duplicated!")
```

3 More Useful Functions

Lists are very useful datastructures, and thus there are a bunch of in-built python functions which allow us manipulate lists in varied ways. A few of the most useful for our purposes are listed here (you can use these to streamline the solutions to many of our exercises!)

3.0.1 Split

split() is a function which allows us to break up strings into sub-strings stored in a list

```
In [4]: # Example of split()
        my_sentence = "I am too lazy to make sentences into lists"
        my_list = my_sentence.split(" ") # Split up string by empty spaces

        print(my_list)
```

['I','am', 'too', 'lazy', 'to', 'make', 'sentences', 'into', 'lists']

3.0.2 Join

join() is essentially the opposite of split, and allows us to nicely format our lists when we want to print them

```
In [8]: # Example of join()
        # NOTE: all items in the list must be strings - unless you do something clever! ;)
        my_list = ["I'm", "12", "now", "mum", "I", "can", "do", "what", "I", "want"]
        my_sentence = " ".join(my_list) # Join items in list with an empty space

        print(my_sentence)
```

I'm 12 now mum I can do what I want

3.0.3 Replace

The **.replace()** function allows us to replace substrings with other substrings of our choice

```
In [32]: # Example of replace
         # string.replace("what to replace", "replace with")
         my_string = "5, 6, 7, 8, Who do we hate? Alex!"
         my_string = my_string.replace("hate", "appreciate")

         print(my_string) # Much better!
```

5, 6, 7, 8, Who do we appreciate? Alex!

3.0.4 Min and Max

min() and **max()** do what you'd expect - So long as you have numbers in the list; string comparison is a bit less straightforward

```
In [17]: my_numbers = list(range(5,25))
         print(max(my_numbers))
         print(min(my_numbers))
```

24
5

3.0.5 Sort

Under the hood, min() and max() work by sorting the list first. We can do this manually by using **.sort()**

```
In [24]: # Example of sorting
         # NOTE: don't mix strings and numbers if you want to sort!
         my_list = [4, 7, -20, 32, 9.3]
         my_list.sort() # This sorts the list

         print(my_list)
```

[-20, 4, 7, 9.3, 32]