Loops

Thursday, November 12, 2020 2:17 PM

For Loops

For loops are very useful for iterating over lists, dictionaries, tuples, sets, and strings
The for loops in python are less like for loops in other languages, and more like an iterator

```
for x in range(0, 3):
    print("We're on time ", x)
```

The range function can be used to loop through a section of code a specified number of times
- range needs a sequence of numbers. Default is to start and 0 and iterate by 1 each time

When iterating through sequential data you do not need declare an indexing variable as you would in other languages.
The enumerate command allows us to pull the index as well as the value in the for loop
The break statement will stop a loop before it completes looping through all items
The continue statement will stop the current iteration of the loop and continue to the next
The else key word inside a loop is what gets executed when the loop is complete
You can perform nested loops where the inner loop gets fully completed each time the outer loop iterates a single time.
Pass statement inside a loop if it's empty

List Comprehension

A super short syntax for creating new list based on the values of an existing list
- newlist = [expression for item in iterable if condition == True]
- newlist = [x for x in oldlist if "asdf" in x]

Much faster!

Dictionary comprehension is almost the exact same with small caveat of including key : value
- newdict = [returned key:value for key, value in iterable]

While Loops

Use these loops when you don't know the number of iterations before a task will be complete