Notes - Tuesday 1/5/21

1/5/21 Tuesday

New year, same course. Let's learn more about Python.

For Loops

Many objects in Python are iterable or are capable of being repeated. This means we can iterate over every element in the object. For example, we can:

- Iterate or repeat every element in a list
- Each character in a string
- Every key in a dictionary

An associated term is ***iterable*** which means the object can be iterated.

We can use For Loops to execute a block of code for each iteration.

Basic Syntax looks like:

```
1 variable_2_iterate = [1,2,3]
2 for my_variable_name_4_item in variable_2_iterate
3    print(my_variable_name_4_item)
```

So in this for loop, I'm saying that if I see this variable, it's a stand-in for the list of numbers. When I see the list of numbers, I want it to print out the list of numbers. But I could also just say that when I see that list of numbers, I want it to print out "Hello World". So let's change my example:

```
1 for_loop_list1 = [1,2,3,4,5,6,7,8,9,10]
2 for bananas in for_loop_list1:
3    print(bananas)
```

If line 3 says to print `bananas`, it will just output the list of numbers I had in for_loop_list1. If I said for it to print hello world, I'd get Hello World back 10 times since I have 10 items in the list.

```
1 for_loop_list1 = [1,2,3,4,5,6,7,8,9,10]
2 for bananas in for_loop_list1:
3    print('Hello World')
```

Advanced Example 1

Take note of the indentations.

```
1 for num in for_loop_list1:
2     # Check for even numbers
3     if num % 2 == 0:
4     # This says I'm setting variable to num then
5     # checking that when I divide by 2, the
6     # remainder is 0
7         print(f'Even Number: {num}')
8     else:
9         print(f'Odd Number: {num}')
10     # f-string literal to make the printing look nicer
11     # so I'm setting up what I want it to say then putting
12     # variable 'num' in the {}
```

Advance Example 2

Let's look at iterations in strings now. The below example will print out each character in `'Hello World'`.

```
1 mystring = 'Hello World'
2 for letter in mystring:
3    print(letter)
```

Note that you don't have to set any variables if you don't want to. Using `_` is useful for when you want the code to do something but you don't intend on using the block of code again or don't need the variable in the future.

No variable for the string:

```
1 for banana in 'Hello World':
2    print(banana)
```

No variables at all:

```
1 for _ in 'Hello World':
2   print('Fuck you')
```

Tuples in a For Loop

Tuples have a special feature called Tuple Unpacking. This is commonly used in Python libraries. There will be tuples inside a list. If you call them in a for loop, the tuples in the list will be repeated as the objects being iterated on. But you could also further unpack them into their individual objects. So the For Loop could also pull out one object in the tuple. It does this by repeating the tuple as a packed object and then you can unpack it to a more granular level. So the syntax would be:

```
1 # your_data_here - could be a list or a tuple or a string
2 for ur_variable_name in your_data_here:
3   print(variable_u_want_2_print)
```

So an example might be:

```
1 tuple_list = [(1,2,3),(4,5,6),(7,8,9)]
2 for x,y,z in tuple_list:
3   print(y)
```

With your output being: 2,5,8.

Dictionaries in For Loops

You can setup dictionaries in For Loops and they have the same tuple unpacking features. Remember that dictionaries are NOT sorted. The default output in dictionaries is the key. The syntax would be as seen below. And you'd get the keys as the outputs.

```
1 ur_dictionary = {'k1':1,'k2':2,'k3':3}
2 for item in ur_dictionary:
3   print(item)
```

Let's say I want to see both the keys and the values. I have to modify the code in line 2 by adding `.items()` to the end of object.

```
1 for_dict = {'k1':1,'k2':2,'k3':3}
2 for item in for_dict.items():
3   print(item)
```

Let's say I want to see the tuple unpacking in action. I have to modify the code in line 2 by adding the variable setup so `key` would be the left value and `value` would be the right value. Then I'd set `.key()` to the end of object. This would output the key. And I can run it for value by swapping in `.value()` in line 2 and `value` in line 3.

```
1 for_dict = {'k1':1,'k2':2,'k3':3}
2 for key,value in for_dict.key():
3   print(key)
```

Revision #3
Created 5 January 2021 21:16:22 by cba88
Updated 6 January 2021 03:48:19 by cba88