5.1 - Uses of collection and datatype

Transform Lists into Other Data Structures

Convert A Python List To A String

You convert a list to a string by using `''.join()` . This operation allows you to glue together all strings in your list together and return them as a string

```
# List of Strings to a String
listOfStrings = ['One', 'Two', 'Three']
strOfStrings = ''.join(listOfStrings)
print(strOfStrings)

# List Of Integers to a String
listOfNumbers = [1, 2, 3]
strOfNumbers = ''.join(str(n) for n in listOfNumbers)
print(strOfNumbers)


OneTwoThree
123
```

Convert A List To A Tuple

You can change a list to a tuple in Python by using the tuple() function. Pass your list to this function and you will get a tuple back!

NOTE: tuples are immutable thus can't change them afterwards :(

Convert Your List To A Set In Python

a set is an unordered collection of unique items. That means not only means that any duplicates that you might have had in your original list will be lost once you convert it to a set, but also the order of the list elements.

You can change a list into a set with the set() function. Just pass your list to it!

Convert Lists To A Dictionaries

A dictionary works with keys and values, so the conversion from a list to a dictionary might be less straightforward. `helloWorld = ['hello','world','1','2']`

You will need to make sure that `hello` and `world` and '1' and '2' are interpreted as key-value pairs. The way to do this is to select them with the slice notation and pass them to zip().

zip() actually works like expected: it zips elements together. In this case, when you zip the helloWorld elements helloWorld[0::2] and helloWorld[1::2]

```
helloWorld = ['hello','world','1','2']
# print(list(zip(helloWorld)))

helloWorldDictionary = dict(zip(helloWorld[0::2], helloWorld[1::2]))

# Print out the result
print(helloWorldDictionary)
```

```
[('hello',), ('world',), ('1',), ('2',)]
{'hello': 'world', '1': '2'}
```

Note that the second element that is passed to the zip() function makes use of the step value to make sure that only the world and 2 elements are selected. Likewise, the first element uses the step value to select hello and 1.

If your list is large, you will probably want to do something like this:

```
a = [1, 2, 3, 4, 5]

# Create a list iterator object
i = iter(a)
for k in i:
    print(k)

# Zip and create a dictionary
print(dict(zip(i, i)))
```

```
{}
```

```
## Difference Between The Python append() and extend() Methods?
# Append [4,5] to `shortList`
# This is your list
shortList  = [1, 2, 3]
longerList = [1, 2, 3]
# Check whether it's iterable
list.__iter__

shortList.append([4, 5])

# Use the `print()` method to show `shortList`
print(shortList)

# Extend `longerList` with [4,5]
longerList.extend([4, 5])

# Use the `print()` method to see `longerList`
print(longerList)
```

```
[1, 2, 3, [4, 5]]
[1, 2, 3, 4, 5]
```

Clone Or Copy A List in Python

here are a lot of ways of cloning or copying a list:

```
You can slice your original list and store it into a new variable: newList = oldList[:]
You can use the built-in list() function: newList = list(oldList)
You can use the copy library:
    With the copy() method: newList = copy.copy(oldList)
    If your list contains objects and you want to copy those as well, you can use copy.deepcopy(): copy.deepcopy(oldList)
```

```
# Copy the grocery list by slicing and store it in the `newGroceries` variable
groceries = [1, 2, 3, 4, 5, 6]
newGroceries = groceries[:]
# Copy the grocery list with the `list()` function and store it in a `groceriesForFriends` variable
groceriesForFriends = list(groceries)
# Import the copy library
import copy as c
# Create a `groceriesForFamily` variable and assign the copied grocery list to it
groceriesForFamily = c.copy(groceries)
# Use `deepcopy()` and assign the copied list to a `groceriesForKids` variable
```

```
groceriesForKids = c.deepcopy(groceries)
```