Lists Part 2

Lists Containing Other Lists

Lists can have objects of any type.

So, lists can have other lists as elements.

To access the element of a list within another list, you must index twice. For example, if we had this list:

```
my_list = [["hello", "bye"], [1, 5]]
```

To access `"hello"`, that would be `my_list[0][0]`. It's index 0 of index 0 of `my_list`.

Each new layer of depth can be seen as a "dimension". So if you have a list within a list within a list, that's 3 dimensions.

Exercise

How do you access each element in this list by index?

```
my_list = ["hello", 1, ["dog", 3], "cat", [True, ["frog", 5]]]
```

For example, `"hello"` is `my_list[0]`. How do you access all the other elements?

2D Lists

We can create a 2 dimensional list.

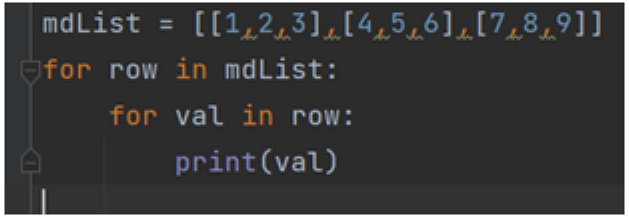
Another way to think of them is a list of lists.

We can think of them like a table.

If we want to access an individual point, we need to use 2 indexes.

List[i] selects the 'row'

List[i][j] selects the 'row' and 'column'



```
mdList = [[1,2,3],[4,5,6],[7,8,9]]
for row in mdList:
    for val in row:
        print(val)
```

Nested Loops Review

Nested Loops are when inside of a loop we place another one. With every iteration of the outer loop, the inner loop runs all the way through.

One use of nested loops is they are great for displaying or changing multidimensional lists. They are also great if we need to loop through something multiple times.

We try to avoid them because they can make our code take a long time to run. This isn't a concern for what we are doing for the most part but it is something to be aware of.

To create one we just simply write out a loop inside of one that we already have written out.

Make sure you keep track of where your code should be and what indent it should have. To help with this use your IDE to your advantage.

Exercise

Write a 2D list that is a 3x3 grid of numbers.

Write some code that prints out that grid nicely with proper formatting.

Example:

```
lis = [[1,2,3],[4,5,6],[7,8,9]]

1 2 3

4 5 6

7 8 9
```

Exercise

Write some code that goes through a 2D list and prints the columns.

Example:

```
lis = [[1,2,3],[4,5,6],[7,8,9]]
```

```
1 4 7

2 5 8

3 6 9
```

Hint: First create a **new 2D list** with swapped rows and columns. (You will need 2 nested for loops.) Then it's the same as the last problem.

Exercise

You are given a 2D list representing a table of data with rows and columns. Write a Python program to calculate the sum and average of each column in the table.

For example, if this is your list:

```
data = [[45,56,89],[67,34,78],[23,67,34]]
```

This would be your output:

```
Column 1: Sum = 135, Average = 45.0
Column 2: Sum = 157, Average = 52.33
Column 3: Sum = 201, Average = 67.0
```

Hint: Make a list to store the sums, and a list to store the averages.

List Comprehensions

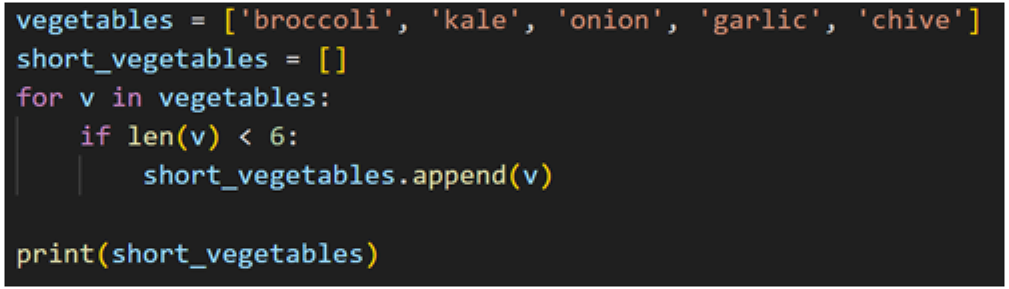
List comprehensions let you quickly make a new list based on the values of an existing list.

You can make a new list with only one line of code.

It doesn't modify the old list; instead, it returns a new list which is a filtered version of the old list.

List Comprehensions

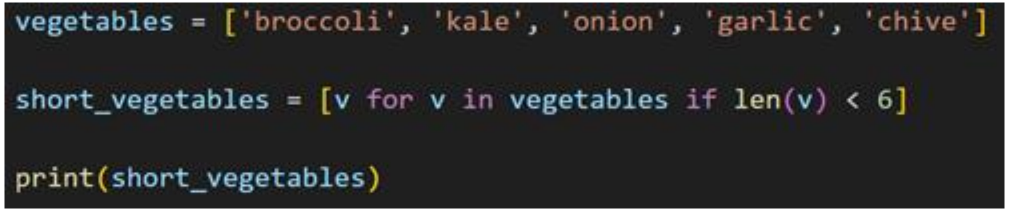
Let's say you have a list of vegetables, and you want a new list containing only the vegetables that are less than 6 letters long.



```python
vegetables = ['broccoli', 'kale', 'onion', 'garlic', 'chive']
short_vegetables = []
for v in vegetables:
    if len(v) < 6:
        short_vegetables.append(v)

print(short_vegetables)
```

Solution without using list comprehension



```python
vegetables = ['broccoli', 'kale', 'onion', 'garlic', 'chive']

short_vegetables = [v for v in vegetables if len(v) < 6]

print(short_vegetables)
```

Solution using list comprehension

Both result in `short_vegetables = ['kale', 'onion', 'chive']`

List Comprehensions

```
new_list = [x for x in original_list if condition]
```

The new list, which is a filtered version of the original list

Temporary variable, initialized here, which represents an item in the original list

The original list, which we are filtering based off of

Boolean condition involving x. If it's True, x gets added to the new list.

Only add items from the original list into the new list if they meet a certain **condition**

List Comprehensions

```
new_list = [expression for x in original_list]
```

The new list, which is a filtered version of the original list

Expression modifying x in some way. This version of x gets added to the new list.

Temporary variable, initialized here, which represents an item in the original list

The original list, which we are filtering based off of

Add the **expression** into the new list for every item of the original list

Exercise

You are given a list of integers. Write a Python program to create a new list that only includes the **even numbers** from the original list.

You can do this in one line with a list comprehension.

Example:

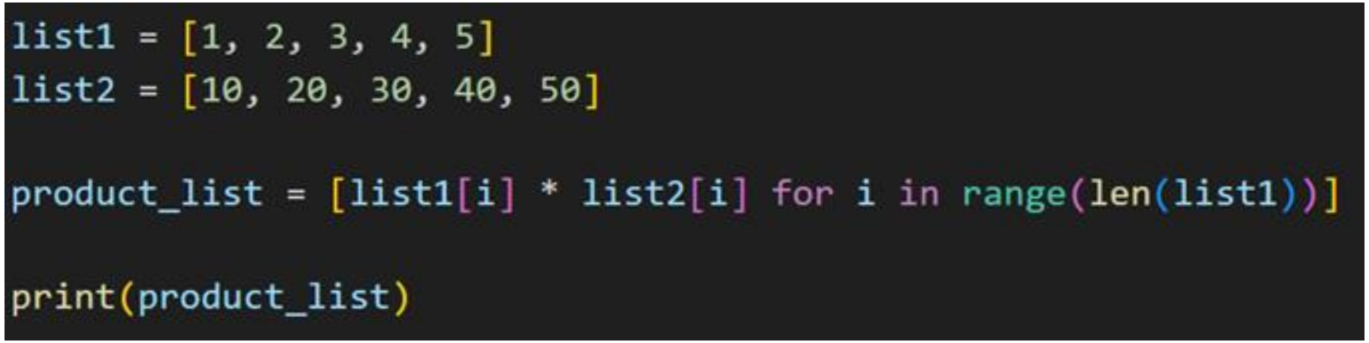
```
original_list = [34, 57, 81, 92, 2, 13]

new_list = [34, 92, 2]
```

Example

You can create more complex variations of list comprehensions. For example, the "original list" could be a range.

Discussion: What is the output of this code?



```
list1 = [1, 2, 3, 4, 5]
list2 = [10, 20, 30, 40, 50]

product_list = [list1[i] * list2[i] for i in range(len(list1))]

print(product_list)
```

Exercise

You work for a sales company and must generate a list of all customers who get a certain discount. The criteria for getting a discount is that they're over 60 years old and have made at least 5 purchases.

You have a list of customers over 60, and a list of customers who have made at least 5 purchases. Use a list comprehension to output a list of customers that fit both criteria for the discount. You can do this in one line of code.

Example:

```
over_60_years = ['Dominic', 'Linda', 'Simone', 'Swathi', 'Olaf']
over_5_purchases = ['Finn', 'Simone', 'Aaron', 'Dominic']
```

Output: `['Dominic', 'Simone']`

Resources

https://www.w3schools.com/python/python_lists_comprehension.asp