**List Methods**

List Comprehension

**--** **How to create a list in python?**

1. Using For Loop:

-- The most common type of loop is the for loop. You can use a for loop to create a list of elements in three steps:

1. Instantiate an empty list.
2. Loop over an iterable or [range](https://realpython.com/python-range/) of elements.
3. [Append](https://realpython.com/python-append/) each element to the end of the list.

-- If you want to create a list containing the first ten perfect squares, then you can complete these steps in three lines of code:

squares = []

*for* i in range(10):

    squares.append(i \* i)

print(squares)

-- Here, you instantiate an empty list, squares. Then, you use a for loop to iterate over range(10). Finally, you multiply each number by itself and append the result to the end of the list.

2. Using map() Object:

-- map() provides an alternative approach that’s based in functional programming.

-- You pass in a function and an iterable, and map() will create an object.

-- This object contains the output you would get from running each iterable element through the supplied function.

-- As an example, consider a situation in which you need to calculate the price after tax for a list of transactions:

txns = [1.09, 23.56, 57.84, 4.56, 6.78]

TAX\_RATE = .08

def *get\_price\_with\_tax*(txn):

*return* txn \* (1 + TAX\_RATE)

final\_prices = map(get\_price\_with\_tax, txns)

print(list(final\_prices))

-- Here, you have an iterable txns and a function get\_price\_with\_tax(). You pass both of these arguments to map(), and store the resulting object in final\_prices. You can easily convert this map object into a list using list().

3. Using List Comprehension:

-- List comprehensions are a third way of making lists.

-- With this elegant approach, you could rewrite the for loop from the first example in just a single line of code:

squares = [i \* i *for* i in range(10)]

print(squares)

-- Rather than creating an empty list and adding each element to the end, you simply define the list and its contents at the same time by following this format:

new\_list = [expression for member in iterable]

-- Every list comprehension in Python includes three elements:

1. **expression** is the member itself, a call to a method, or any other valid expression that returns a value. In the example above, the expression i \* i is the square of the member value.
2. **member** is the object or value in the list or iterable. In the example above, the member value is i.
3. **iterable** is a list, set, sequence, [generator](https://realpython.com/introduction-to-python-generators/), or any other object that can return its elements one at a time. In the example above, the iterable is range(10).

-- Because the **expression** requirement is so flexible, a list comprehension in Python works well in many places where you would use map().

-- You can rewrite the pricing example with its own list comprehension:

txns = [1.09, 23.56, 57.84, 4.56, 6.78]

TAX\_RATE = .08

def *get\_price\_with\_tax*(txn):

*return* txn \* (1 + TAX\_RATE)

final\_prices = [get\_price\_with\_tax(i) *for* i in txns]

print(final\_prices)

-- The only distinction between this implementation and map() is that the list comprehension in Python returns a list, not a map object.