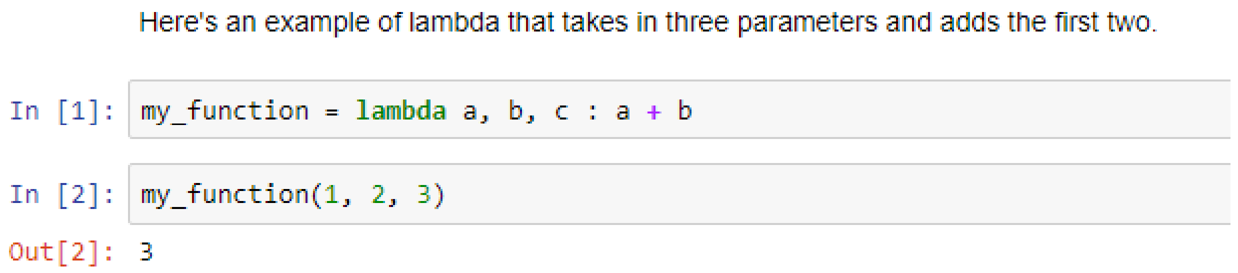
Advanced Python Lambda and List Comprehensions

Lambda's are Python's way of creating anonymous functions. These are the same as other functions, but they have no name. The intent is that they're simple or short lived and it's easier just to write out the function in one line instead of going to the trouble of creating a named function.

You declare a lambda function with the word lambda followed by a list of arguments, followed by a colon and then a single expression, and this is key, there's only one expression to be evaluated in a lambda.
The expression value is returned on execution of the lambda.
The return of a lambda is a function reference.
So in this case, you would execute my_function and pass in three different parameters.


Here's an example of lambda that takes in three parameters and adds the first two.

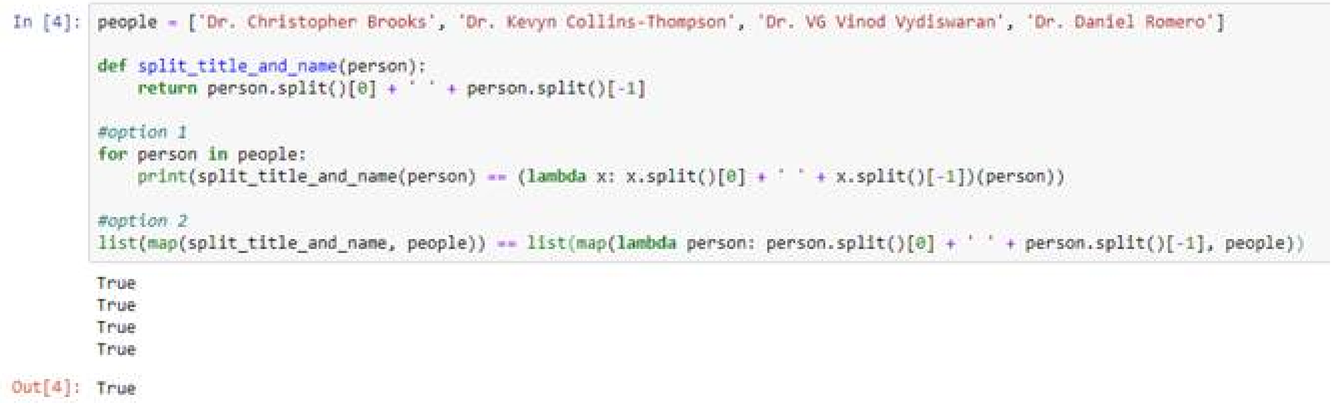
```
In [1]: my_function = lambda a, b, c : a + b
```

```
In [2]: my_function(1, 2, 3)
Out[2]: 3
```

Note that you can't have default values for lambda parameters and you can't have complex logic inside of the lambda itself because you're limited to a single expression.

So lambdas are really much more limited than full function definitions. But I think they're very useful for simple little data cleaning tasks.

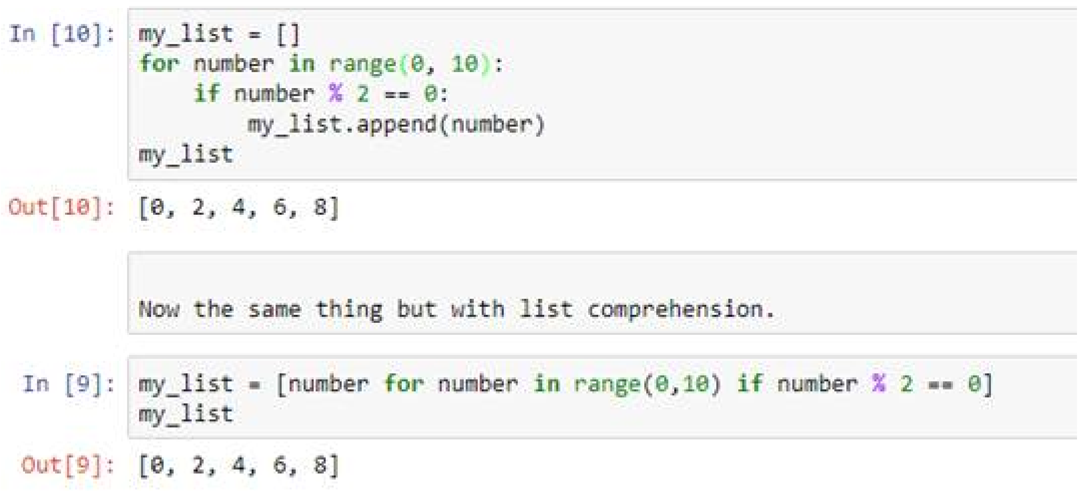
Here is more complex example.



```
In [4]: people = ['Dr. Christopher Brooks', 'Dr. Kevyn Collins-Thompson', 'Dr. VG Vinod Vydiswaran', 'Dr. Daniel Romero']

        def split_title_and_name(person):
            return person.split()[0] + ' ' + person.split()[-1]

        #option 1
        for person in people:
            print(split_title_and_name(person) == (lambda x: x.split()[0] + ' ' + x.split()[-1])(person))

        #option 2
        list(map(split_title_and_name, people)) == list(map(lambda person: person.split()[0] + ' ' + person.split()[-1], people))

        True
        True
        True
        True
Out[4]: True
```

We've learned a lot about sequences and in Python, Tuples, lists, dictionaries and so forth. Sequences are structures that we can iterate over, and often we create these through loops or by reading in data from a file.

Python has built in support for creating these collections using a more abbreviated syntax called **list comprehensions**.
Here's an example.
First we write up a little for-loop. Here I'm iterating between zero and 10 and then checking with the modulus operator to see if the number divided by two results in any decimals.
If the modulus two of the number is zero, then I know it divides evenly so this must be an even number and I'll add it to our list.



```
In [10]: my_list = []
         for number in range(0, 10):
             if number % 2 == 0:
                 my_list.append(number)
         my_list

Out[10]: [0, 2, 4, 6, 8]
```

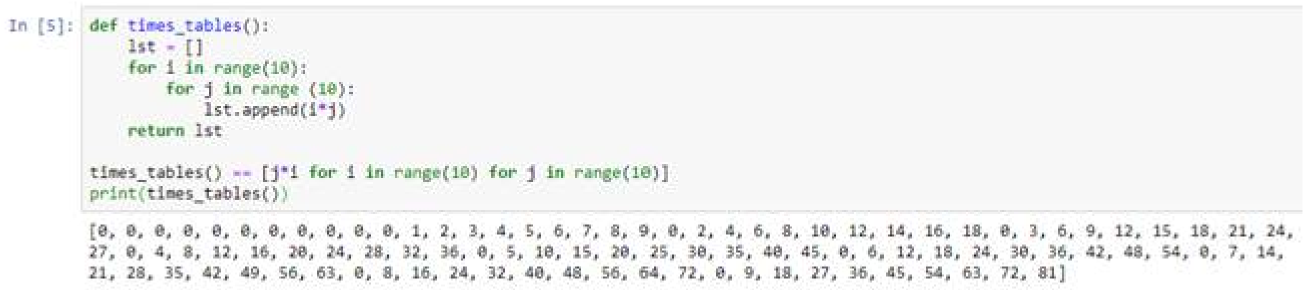
Now the same thing but with list comprehension.

```
In [9]: my_list = [number for number in range(0,10) if number % 2 == 0]
        my_list

Out[9]: [0, 2, 4, 6, 8]
```

We can rewrite this as a list comprehension by pulling the iteration on one line.
We start the list comprehension with the value we want in the list.
In this case, it's a number.
Then we put it in the for-loop, and then finally, we add any condition clauses.
You can see that this is much more compact of a format. And it tends to be faster as well.

Just like with lambdas, list comprehensions are a condensed format which may offer readability and performance benefits

Here is another example:



```
In [5]: def times_tables():
            lst = []
            for i in range(10):
                for j in range (10):
                    lst.append(i*j)
            return lst

        times_tables() == [j*i for i in range(10) for j in range(10)]
        print(times_tables())

        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 0, 3, 6, 9, 12, 15, 18, 21, 24,
        27, 0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 0, 7, 14,
        21, 28, 35, 42, 49, 56, 63, 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 0, 9, 18, 27, 36, 45, 54, 63, 72, 81]
```