Using Lists with For Loops

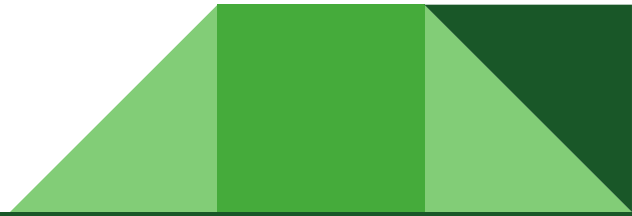
Reviewing For Loops

What does the `range()` function do when I give it `1` parameter?

```
for i in range(5):
    print(i)
```

What are the indices of a list with 5 elements?

```
["a", "b", "c", "d", "e"]
```



Reviewing For Loops

What does the `range()` function do when I give it 1 parameter?

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

What are the indices of a list with 5 elements?

```
["a", "b", "c", "d", "e"]
```

Reviewing For Loops

What does the `range()` function do when I give it 1 parameter?

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

What are the indices of a list with 5 elements?

```
["a", "b", "c", "d", "e"]
  0     1     2     3     4
```

Reviewing For Loops

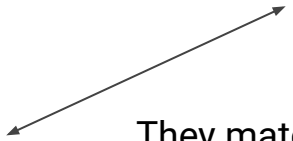
What does the `range()` function do when I give it 1 parameter?

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

What are the indices of a list with 5 elements?

```
["a", "b", "c", "d", "e"]
  0     1     2     3     4
```



They match!

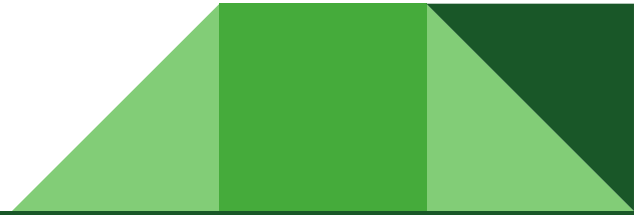
Let's apply that!

```
my_list = ["a", "b", "c", "d", "e"]
for i in range(5):
    print(my_list[i])
```

Let's apply that!

```
my_list = ["a", "b", "c", "d", "e"]
for i in range(5):
    print(my_list[i])
```

```
a
b
c
d
e
```



Let's apply that!

```
my_list = ["a", "b", "c", "d", "e"]
for i in range(5):
    print(my_list[i])
```

What if `my_list` became longer or shorter, though?

```
a
b
c
d
e
```

Let's apply that!

```
my_list = ["a", "b", "c", "d", "e", "f"]
for i in range(5):
    print(my_list[i])
```

What if `my_list` became longer or shorter, though?

```
a
b
c
d
e
```

Let's apply that!

```
my_list = ["a", "b", "c", "d", "e", "f"]
for i in range(5):
    print(my_list[i])
```

What if `my_list` became longer or shorter, though?

We can use the length of the list to determine the number of times to loop!

```
a
b
c
d
e
```

Let's apply that!

```
my_list = ["a", "b", "c", "d", "e", "f"]
for i in range(len(my_list)):
    print(my_list[i])
```

What if `my_list` became longer or shorter, though?

We can use the length of the list to determine the number of times to loop!

```
a
b
c
d
e
```

Let's apply that!

```
my_list = ["a", "b", "c", "d", "e", "f"]
for i in range(len(my_list)):
    print(my_list[i])
```

What if `my_list` became longer or shorter, though?

We can use the length of the list to determine the number of times to loop!

```
a
b
c
d
e
f
```

An Alternative Type of Loop

If I want to do a for loop in which I loop over the **elements** instead of the **indices**, there's a way to do that!

```
my_list = ["a", "b", "c", "d", "e"]
for element in my_list:
    print(element)
```

An Alternative Type of Loop

If I want to do a for loop in which I loop over the **elements** instead of the **indices**, there's a way to do that!

```
my_list = ["a", "b", "c", "d", "e"]
for element in my_list:
    print(element)
```

```
a
b
c
d
e
```

Doing both at once!

We can use a new function to perform the abilities of both styles of for loop at once - enumerate().

The way it works is that we make 2 loop variables - one for the index, one for the element, and enumerate() will fill both of them each time through the loop.

```
my_list = ["a", "b", "c", "d"]
for i, element in enumerate(my_list):
    print(str(i) + ": " + element)
```

Doing both at once!

We can use a new function to perform the abilities of both styles of for loop at once - enumerate().

The way it works is that we make 2 loop variables - one for the index, one for the element, and enumerate() will fill both of them each time through the loop.

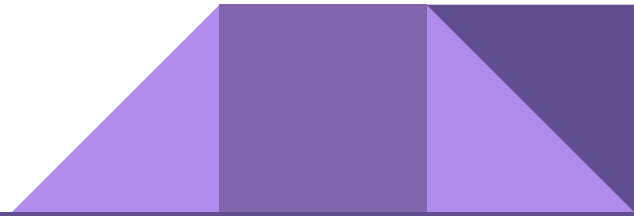
```
my_list = ["a", "b", "c", "d"]
for i, element in enumerate(my_list):
    print(str(i) + ": " + element)
```

```
0: a
1: b
2: c
3: d
```

Bonus Question

What do you think will be printed in this scenario?

```
my_list = [1, 2, 3, 4]
for num in my_list:
    num = num + 5
print(my_list)
```



Bonus Question

What do you think will be printed in this scenario?

```
my_list = [1, 2, 3, 4]
for num in my_list:
    num = num + 5
print(my_list)
```

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

Because `num` isn't actually pointing towards the elements in the list - it's just grabbing their values one at a time.