Applications of For Loops

A Quick Shortcut

Before we talk about `for` loops anymore, we're gonna take a brief detour and discuss **Compound Assignment Operators**.

When we want to update the value of a variable using its own current value, we can do this:

```
my_num = 5
print(my_num)
my_num = my_num + 5
print(my_num)
```

```
5
10
```

A Quick Shortcut

If I want to avoid having to re-type my variable's name, though, I can use a **Compound Assignment Operator**. It looks like this:

```
my_num = 5
print(my_num)
my_num += 5
print(my_num)
```

```
5
10
```

A Quick Shortcut

If I want to avoid having to re-type my variable's name, though, I can use a **Compound Assignment Operator**. It looks like this:

```
my_num = 5
print(my_num)
my_num += 5
print(my_num)
```

```
5
10
```

This will work with **any** of the binary mathematical operators we learned about, so we can update a variable's value in up to 7 different mathematical ways!

+= -= *= /= //= **= %=

Summing with a loop

One of the most common applications of a `for` loop is finding the **sum** of a sequence of values.

We can do this by storing our temporary total in a variable and updating its value each time through the loop!

```
total = 0
for i in range(1, 4):
    total += i
print("Total: " + str(total))
```

Summing with a loop

One of the most common applications of a `for` loop is finding the **sum** of a sequence of values.

We can do this by storing our temporary total in a variable and updating its value each time through the loop!

total
0

```
total = 0
for i in range(1, 4):
    total += i
print("Total: " + str(total))
```

Summing with a loop

One of the most common applications of a `for` loop is finding the **sum** of a sequence of values.

We can do this by storing our temporary total in a variable and updating its value each time through the loop!

i	total
-	0
1	1

```
total = 0
for i in range(1, 4):
    total += i
print("Total: " + str(total))
```

Summing with a loop

One of the most common applications of a `for` loop is finding the **sum** of a sequence of values.

We can do this by storing our temporary total in a variable and updating its value each time through the loop!

i	total
-	0
1	1
2	3

```
total = 0
for i in range(1, 4):
    total += i
print("Total: " + str(total))
```

Summing with a loop

One of the most common applications of a `for` loop is finding the **sum** of a sequence of values.

We can do this by storing our temporary total in a variable and updating its value each time through the loop!

```
total = 0
for i in range(1, 4):
    total += i
print("Total: " + str(total))
```

i	total
-	0
1	1
2	3
3	6

Summing with a loop

One of the most common applications of a `for` loop is finding the **sum** of a sequence of values.

We can do this by storing our temporary total in a variable and updating its value each time through the loop!

```
total = 0
for i in range(1, 4):
    total += i
print("Total: " + str(total))
```

```
6
```

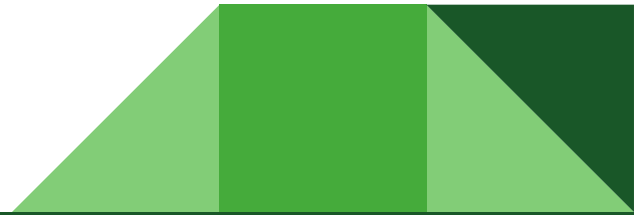
i	total
-	0
1	1
2	3
3	**6**

Adding Constants

Constants are variables whose values will not change throughout a program. We usually show that a variable is a constant by putting its name in ALL_CAPS.

We use constants when we have a value that we plan to use throughout our program, and we want to give it a readable name - for instance, if we wanted to establish the `MIN` and `MAX` values for a `for` loop, we might use constants.

This lets us localize these values as well, making it easy to make a change in 1 place that will affect the whole program.



Adding Constants to Summing w/ Loop

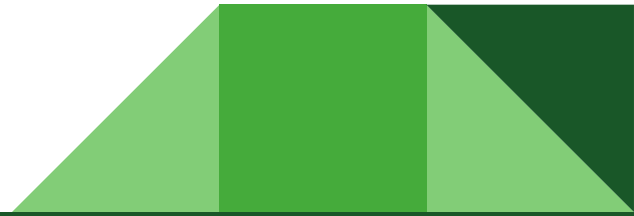
```
MIN = 1
MAX = 4

total = 0
for i in range(MIN, MAX):
    total += i
print("Total: " + str(total))
```

Doubling Up

What would happen if I were to put a `for` loop… inside another `for` loop?

```
for i in range(3):
    for j in range(3):
        print(str(i) + " " + str(j))
```

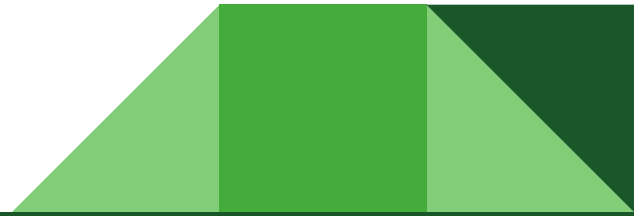


Doubling Up

What would happen if I were to put a `for` loop… inside another `for` loop?

```
for i in range(3):
    for j in range(3):
        print(str(i) + " " + str(j))
```

Before we look at what this does, notice that the name for the 2 loops' **iterator** variable is different.



Doubling Up

What would happen if I were to put a `for` loop… inside another `for` loop?

```
for i in range(3):
    for j in range(3):
        print(str(i) + " " + str(j))
```

When we have **nested** `for` loops, the loop on the inside will complete all of its loops each time the outside loop runs.

i	j
0	0
0	1
0	2
1	0
1	1
1	2
2	0
2	1
2	2