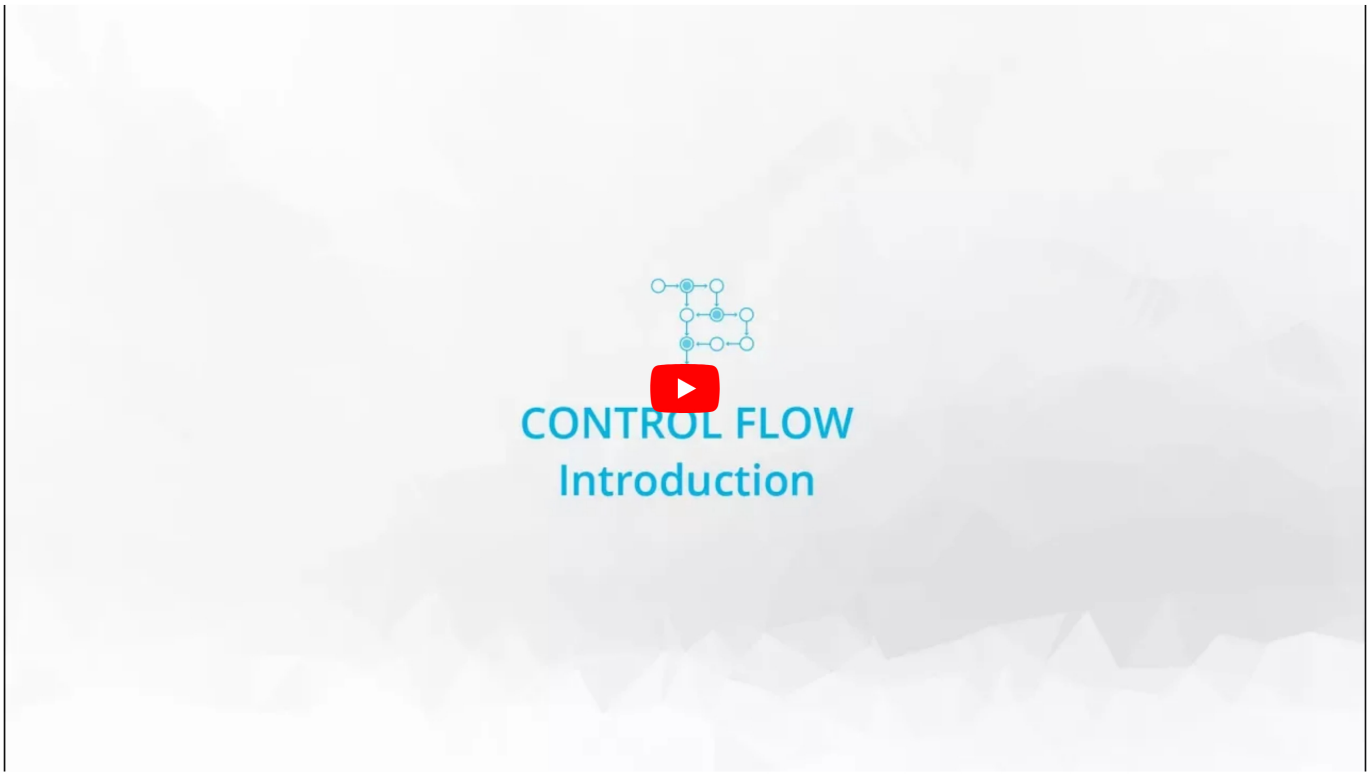
For Loops



For Loops

Python has two kinds of loops - `for` loops and `while` loops. A `for` loop is used to "iterate", or do something repeatedly, over an **iterable**.

An **iterable** is an object that can return one of its elements at a time. This can include sequence types, such as strings, lists, and tuples, as well as non-sequence types, such as dictionaries and files.

Example

Let's break down the components of a `for` loop, using this example with the list `cities`:

```
cities = ['new york city', 'mountain view', 'chicago', 'los angeles']
for city in cities:
    print(city)
print("Done!")
```

Components of a `for` Loop

1. The first line of the loop starts with the `for` keyword, which signals that this is a `for` loop
2. Following that is `city in cities`, indicating `city` is the iteration variable, and `cities` is the iterable being looped over. In the first iteration of the loop, `city` gets the value of the first element in `cities`, which is “new york city”.
3. The `for` loop heading line always ends with a colon `:`
4. Following the `for` loop heading is an indented block of code, the body of the loop, to be executed in each iteration of this loop. There is only one line in the body of this loop - `print(city)`.
5. After the body of the loop has executed, we don't move on to the next line yet; we go back to the `for` heading line, where the iteration variable takes the value of the next element of the iterable. In the second iteration of the loop above, `city` takes the value of the next element in `cities`, which is "mountain view".

6. This process repeats until the loop has iterated through all the elements of the iterable. Then, we move on to the line that follows the body of the loop - in this case, `print("Done!")`. We can tell what the next line after the body of the loop is because it is unindented. Here is another reason why paying attention to your indentation is very important in Python!

Executing the code in the example above produces this output:

```
new york city
mountain view
chicago
los angeles
Done!
```

You can name iteration variables however you like. A common pattern is to give the iteration variable and iterable the same names, except the singular and plural versions respectively (e.g., 'city' and 'cities').

Using the `range()` Function with `for` Loops

`range()` is a built-in function used to create an iterable sequence of numbers. You will frequently use `range()` with a `for` loop to repeat an action a certain number of times. Any variable can be used to iterate through the numbers, but Python programmers conventionally use `i`, as in this example:

```
for i in range(3):
    print("Hello!")
```

Output:

```
Hello!
Hello!
Hello!
```

`range(start=0, stop, step=1)`

The `range()` function takes three integer arguments, the first and third of which are optional:

- The 'start' argument is the first number of the sequence. If unspecified, 'start' defaults to 0.
- The 'stop' argument is 1 more than the last number of the sequence. This argument must be specified.
- The 'step' argument is the difference between each number in the sequence. If unspecified, 'step' defaults to 1.

Notes on using `range()`:

- If you specify one integer inside the parentheses with `range()`, it's used as the value for 'stop,' and the defaults are used for the other two.
  e.g. - `range(4)` returns `0, 1, 2, 3`
- If you specify two integers inside the parentheses with `range()`, they're used for 'start' and 'stop,' and the default is used for 'step.'
  e.g. - `range(2, 6)` returns `2, 3, 4, 5`
- Or you can specify all three integers for 'start', 'stop', and 'step.'
  e.g. - `range(1, 10, 2)` returns `1, 3, 5, 7, 9`

Creating and Modifying Lists

In addition to extracting information from lists, as we did in the first example above, you can also create and modify lists with `for` loops. You can **create** a list by appending to a new list at each iteration of the `for` loop like this:

```
## Creating a new list
cities = ['new york city', 'mountain view', 'chicago', 'los angeles']
capitalized_cities = []

for city in cities:
    capitalized_cities.append(city.title())
```

Modifying a list is a bit more involved, and requires the use of the `range()` function.

We can use the `range()` function to generate the indices for each value in the `cities` list. This lets us access the elements of the list with `cities[index]` so that we can modify the values in the `cities` list in place.

modify the values in the `cities` list in place.

```
cities = ['new york city', 'mountain view', 'chicago', 'los angeles']

for index in range(len(cities)):
    cities[index] = cities[index].title()
```

Previous Next

Give Page Feedback