Data Scientist
Codecademy
Python Fundamentals
Loops

What are Loops?

In programming, the process of using an initialization, repetitions, and an ending condition is called a loop. In a loop, we perform a process of iteration.

Python implements two types of iteration:

- indefinite iteration
- definite iteration

Why Loops?

When we have to repeat a task many times we could end up with inconsistencies and mistakes.

For Loops: Introduction

for loop is a type of definite iteration. In a for loop, we know in advance how many times the loop will need to iterate because we will be working on a collection with a predefined length.

Syntax:

```
for <temporary variable> in <collection>:
    <action>
```

- for keyword indicates the start of the loop.
- <temporary variable> is used to represent the value of the element in the collection the loop is currently on.
- in keyword separates the temporary variable from the collection used for iteration

- <collection> to loop over
- <action> to do anything on each iteration of the loop

- temporary variable's name is arbitrary and does not need to defined before hand.
- everything at the same level of indentation after the for loop declaration is included in the loop body and is run on every iteration of the loop.
- if we forget indentation we get IndentationError or unexpected behavior
- simple for loops can be written in a single line which have to perform a single action

For Loops: Using Range

When we don't want to be iterating through a specific list, but rather only want to perform a certain action multiple times, we can use range.

```
for temp in range(6):
    print("Something")
```

While Loops: Introduction

while loop is a form of indefinite iteration. A while loop performs a set of instructions as long as a given condition is true.

Syntax:

```
while <conditional statement>:
    <action>
```

- indentation for loop body must be kept in mind
- while loop also supports one line loops. here each statement is separated with a ;

While Loops: Lists

We know that a list has a predetermined length. If we use the length of a list as the basis for how long the while loop needs to run, we can iterate the exact length of the list.

Infinite Loops

A loop that never terminates is called an infinite loop. These are very dangerous for our code because they will make our program run forever and thus consume all of our computer's resources.

Loop Control: Break

We can use break loop control statement to stop iteration from inside the loop body. When a program hits a break statement it immediately terminates a loop.

Loop Control: Continue

continue statement is used to skip a particular part of iteration if some condition is met. Anything after the control statement in the loop body will be skipped for that iteration.

Nested Loops

Loops can be nested in python to create complex logics.

Nested loops are helpful to traverse over each element in a 2-d list

List Comprehensions: Introduction

Take an example:

```
numbers = [2, 0, -1, 3, 5, 7]
doubled = []

for number in numbers:
    doubled.append(number*2)

print(doubled)
```

We can use List comprehensions to solve such problems in one line:

```
numbers = [2, 0, -1, 3, 5, 7]
doubled = [num * 2 for num in numbers]

print(doubled)
```

Syntax:

```
new_list = [ <expression> for <element> in <collection>]
```

List Comprehensions: Conditionals

We can incorporate conditional logic in list comprehensions.

```
numbers = [2, 0, -1, 3, -15, 5, 7]
only_negative_doubled = []

for number in numbers:
    if number < 0:
        only_negative_doubled.append(number*2)

print(only_negative_doubled)
```

Using list comprehensions:

```
numbers = [2, 0, -1, 3, -15, 5, 7]
only_negative_doubled = [num * 2 for num in numbers if num < 0]

print(only_negative_doubled)
```

Another example:

```
numbers = [2, -1, 79, 33, -45]
doubled = [num * 2 if num < 0 else num * 3 for num in numbers]
```