Looping Through an Entire List

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)
---> alice
     david
     carolina
```

A Closer Look at Looping

Doing More Work Within a for Loop

Doing Something After a for Loop

Avoiding Indentation Errors

Making Numerical Lists

Using the `range()` Function

```
for value in range(1, 5):
    print(value)
---> 1
     2
     3
     4
```

Using `range()` to Make a List of Numbers

```
numbers = list(range(1, 6))
print(numbers)
---> [1, 2, 3, 4, 5]

even_numbers = list(range(2, 11, 2))
print(even_numbers)
---> [2, 4, 6, 8, 10]
```

Simple Statistics with a List of Numbers

```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
print(min(digits))
---> 0

print(max(digits))
---> 9

print(sum(digits))
---> 45
```

List Comprehensions

A *list comprehension* allows you to generate this same list in just one line of code.
A list comprehension combines the for loop and the creation of new elements into one line, and automatically appends each new element.

```
squares = [value**2 for value in range(1, 11)]
print(squares)
---> [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Working with Part of a List

Slicing a List

To make a slice, you specify the index of the first and last elements you want to work with.

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[0:3])
---> ['charles', 'martina', 'michael']
```

Looping Through a Slice

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print("Here are the first three players on my team:")
for player in players[:3]:
    print(player.title())
---> Here are the first three players on my team:
     Charles
     Martina
```

```
8       Michael
```

Copying a List

To copy a list, you can make a slice that includes the entire original list by omitting the first index and the second index ([:]). This tells Python to make a slice that starts at the first item and ends with the last item, producing a copy of the entire list.

```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods[:]

print("My favorite foods are:")
print(my_foods)
---> My favorite foods are:
     ['pizza', 'falafel', 'carrot cake']

print("\nMy friend's favorite foods are:")
print(friend_foods)
---> My friend's favorite foods are:
     ['pizza', 'falafel', 'carrot cake']

my_foods.append('cannoli')
friend_foods.append('ice cream')


print("My favorite foods are:")
print(my_foods)
---> My favorite foods are:
     ['pizza', 'falafel', 'carrot cake', 'cannoli']

print("\nMy friend's favorite foods are:")
print(friend_foods)
---> My friend's favorite foods are:
     ['pizza', 'falafel', 'carrot cake', 'ice cream']
```

Tuples

Lists work well for storing collections of items that can change throughout the life of a program. The ability to modify lists is particularly important when you're working with a list of users on a website or a list of characters in a game.

However, sometimes you'll want to create a list of items that cannot change. Tuples allow you to do just that. Python refers to values that cannot change as *immutable*, and an immutable list is called a *tuple*.

Defining a Tuple

A tuple looks just like a list, except you use parentheses instead of square brackets.

```
dimensions = (200, 50)
print(dimensions[0])
---> 200
print(dimensions[1])
---> 50
```

Looping Through All Values in a Tuple

Writing Over a Tuple

Styling Your Code

The Style Guide

When someone wants to make a change to the Python language, they write a *Python Enhancement Proposal (PEP)*.
One of the oldest PEPs is PEP 8, which instructs Python programmers on how to style their code. PEP 8 is fairly lengthy, but much of it relates to more complex coding structures than what you've seen so far.

Indentation

Line Length

Blank Lines

Other Style Guidelines