Working With Lists

Looping Through an Entire List

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

- When writing your own for loops that you can choose any name you want for the temporary variable that will be associated with each value in the list.
- However, it's helpful to choose a meaningful name that represents a single item from the list.
- For example, here's a good way to start a `for` loop for a list of cats, a list of dogs, and a general list of items:
  - `for cat in cats:`
  - `for dog in dogs:`
  - `for item in list_of_items:`

Doing More Work Within a `for` Loop

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\n")

print("Thank you, everyone. That was a great magic show!")
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
magicians.py
Alice, that was a great trick!
I can't wait to see your next trick, Alice.

David, that was a great trick!
I can't wait to see your next trick, David.

Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you, everyone. That was a great magic show!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Avoiding Indentation Errors

Forgetting to Indent

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

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
magicians_1.py
  File "magicians_1.py", line 3
    print(magician)
        ^
IndentationError: expected an indented block
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Forgetting to Indent Additional Lines

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
print(f"I can't wait to see your next trick, {magician.title()}.\n")
~
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
magicians_2.py
Alice, that was a great trick!
David, that was a great trick!
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.
```

Indenting Unnecessarily

```
message = "Hello Python world!"
    print(message)
~
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
hello_world.py
  File "hello_world.py", line 2
    print(message)
    ^
IndentationError: unexpected indent
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Indenting Unnecessarily After the Loop

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\n")

    print("Thank you, everyone. That was a great magic show!")
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
magicians_3.py
Alice, that was a great trick!
I can't wait to see your next trick, Alice.

Thank you, everyone. That was a great magic show!
David, that was a great trick!
I can't wait to see your next trick, David.

Thank you, everyone. That was a great magic show!
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you, everyone. That was a great magic show!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Forgetting the Colon

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

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
magicians_4.py
  File "magicians_4.py", line 2
    for magician in magicians
                            ^
SyntaxError: invalid syntax
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Making Numerical Lists

Using the `range()` Function

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

```
>>> for value in range(5):
...     print(value)
...
0
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]
>>>
```

```
squares = []
for value in range(1, 11):
    square = value ** 2
    squares.append(square)
print(squares)
~
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
squares_1.py
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

```
squares = []
for value in range(1, 11):
    squares.append(value ** 2)
print(squares)
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
squares_2.py
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Simple Statistics with a List of Numbers

```
>>> digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> min(digits)
0
>>> max(digits)
9
>>> sum(digits)
45
>>>
```

List Comprehensions

```
squares = [value**2 for value in range(1, 11)]
print(squares)
~
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
squares.py
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Working with Part of a List

Slicing a List

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

```
>>> players = ['charles', 'martina', 'michael', 'florence', 'eli']
>>> print(players[1:4])
['martina', 'michael', 'florence']
>>>
```

```
>>> players = ['charles', 'martina', 'michael', 'florence', 'eli']
>>> print(players[:4])
['charles', 'martina', 'michael', 'florence']
>>>
```

```
>>> players = ['charles', 'martina', 'michael', 'florence', 'eli']
>>> print(players[2:])
['michael', 'florence', 'eli']
>>>
```

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

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())
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
players.py
Here are the first three players on my team:
Charles
Martina
Michael
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Copying a List

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

print("My favorite foods are:")
print(my_foods)

print("\nMy friend's favorite foods are:")
print(friend_foods)
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
foods_1.py
My favorite foods are:
['pizza', 'falafel', 'carrot cake']

My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake']
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

- To prove that we actually have two separate lists, we'll add a new food to each list and show that each list keeps track of the appropriate person's favorite foods:

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

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

print("My favorite foods are:")
print(my_foods)

print("\nMy friend's favorite foods are:")
print(friend_foods)
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
foods.py
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli']

My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'ice cream']
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

- If we had simply set `friend_foods` equal to `my_foods`, we would not produce two separate lists.

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

# This doesn't work:
friend_foods = my_foods

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

print("My favorite foods are:")
print(my_foods)

print("\nMy friend's favorite foods are:")
print(friend_foods)
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
foods_2.py
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']

My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Tuples

- Python refers to values that cannot change as *immutable*, and an immutable list is called a *tuple*.

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

```
>>> dimensions = (200, 50)
>>> dimensions[0] = 250
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

Looping Through All Values in a Tuple

```
>>> dimensions = (200, 50)
>>> for dimension in dimensions:
...     print(dimension)
...
200
50
>>>
```

Writing over a Tuple

- Although you can’t modify a tuple, you can assign a new value to a variable that represents a tuple.

```
dimensions = (200, 50)
print("Original dimensions:")
for dimension in dimensions:
    print(dimension)

dimensions = (400, 100)
print("\nModified dimensions:")
for dimension in dimensions:
    print(dimension)
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$ python
dimensions_1.py
Original dimensions:
200
50

Modified dimensions:
400
100
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_04$
```

Styling Your Code

- Python programmers have agreed on a number of styling conventions to ensure that everyone’s code is structured in roughly the same way.
- Once you’ve learned to write clean Python code, you should be able to understand the overall structure of anyone else’s Python code, as long as they follow the same guidelines.
- If you’re hoping to become a professional programmer at some point, you should begin following these guidelines as soon as possible to develop good habits.

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.

- The Python style guide was written with the understanding that code is read more often than it is written.
- You'll write your code once and then start reading it as you begin debugging.
- When you add features to a program, you'll spend more time reading your code.
- When you share your code with other programmers, they'll read your code as well.
- Given the choice between writing code that's easier to write or code that's easier to read, Python programmers will almost always encourage you to write code that's easier to read.

Indentation

- PEP 8 recommends that you use four spaces per indentation level.
- Using four spaces improves readability while leaving room for multiple levels of indentation on each line.

- In a word processing document, people often use tabs rather than spaces to indent.
- This works well for word processing documents, but the Python interpreter gets confused when tabs are mixed with spaces.
- Every text editor provides a setting that lets you use the `TAB` key but then converts each tab to a set number of spaces.
- You should definitely use your `TAB` key, but also make sure your editor is set to insert spaces rather than tabs into your document.

- Mixing tabs and spaces in your file can cause problems that are very difficult to diagnose.
- If you think you have a mix of tabs and spaces, you can convert all tabs in a file to spaces in most editors.

Line Length

- Many Python programmers recommend that each line should be less than 80 characters.
- PEP 8 also recommends that you limit all of your comments to 72 characters per line, because some of the tools that generate automatic documentation for larger projects add formatting characters at the beginning of each commented line.

- The PEP 8 guidelines for line length are not set in stone, but be aware that people who are working collaboratively almost always follow the PEP 8 guidelines.
- Most editors allow you to set up a visual cue, usually a vertical line on your screen, that shows you where these limits are.

Blank Lines

- To group parts of your program visually, use blank lines.
- You should use blank lines to organize your files, but don’t do so excessively.