PYTHON CRASH COURSE

3RD EDITION

A Hands-On, Project-Based Introduction to Programming

by Eric Matthes

no starch
press

San Francisco

First printing

26 25 24 23 22 1 2 3 4 5

ISBN-13: 978-1-7185-0270-3 (print)
ISBN-13: 978-1-7185-0271-0 (ebook)

Publisher: William Pollock
Managing Editor: Jill Franklin
Production Editor: Jennifer Kepler
Developmental Editor: Eva Morrow
Cover Illustrator: Josh Ellingson
Interior Design: Octopod Studios
Technical Reviewer: Kenneth Love
Copyeditor: Doug McNair
Compositor: Jeff Lytle, Happenstance Type-O-Rama
Proofreader: Scout Festa

For information on distribution, bulk sales, corporate sales, or translations, please contact No Starch Press, Inc. directly at info@nostarch.com or:

No Starch Press, Inc.
245 8th Street, San Francisco, CA 94103
phone: 1.415.863.9900
www.nostarch.com

The Library of Congress has catalogued the first edition as follows:

```
Matthes, Eric, 1972-
  Python crash course : a hands-on, project-based introduction to programming / by Eric Matthes.
       pages cm
  Includes index.
  Summary: "A project-based introduction to programming in Python, with exercises. Covers general
programming concepts, Python fundamentals, and problem solving. Includes three projects - how to
create a simple video game, use data visualization techniques to make graphs and charts, and build
an interactive web application"-- Provided by publisher.
  ISBN 978-1-59327-603-4 -- ISBN 1-59327-603-6
 1.  Python (Computer program language)  I. Title.
  QA76.73.P98M38 2015
  005.13'3--dc23
                                                                    2015018135
```

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

TRY IT YOURSELF

4-10. Slices: Using one of the programs you wrote in this chapter, add several lines to the end of the program that do the following:

- Print the message *The first three items in the list are:*. Then use a slice to print the first three items from that program's list.
- Print the message *Three items from the middle of the list are:*. Then use a slice to print three items from the middle of the list.
- Print the message *The last three items in the list are:*. Then use a slice to print the last three items in the list.

4-11. My Pizzas, Your Pizzas: Start with your program from Exercise 4-1 (page 56). Make a copy of the list of pizzas, and call it `friend_pizzas`. Then, do the following:

- Add a new pizza to the original list.
- Add a different pizza to the list `friend_pizzas`.
- Prove that you have two separate lists. Print the message *My favorite pizzas are:*, and then use a `for` loop to print the first list. Print the message *My friend's favorite pizzas are:*, and then use a `for` loop to print the second list. Make sure each new pizza is stored in the appropriate list.

4-12. More Loops: All versions of *foods.py* in this section have avoided using `for` loops when printing, to save space. Choose a version of *foods.py*, and write two `for` loops to print each list of foods.

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. Once you define a tuple, you can access individual elements by using each item's index, just as you would for a list.

For example, if we have a rectangle that should always be a certain size, we can ensure that its size doesn't change by putting the dimensions into a tuple:

dimensions.py

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

We define the tuple `dimensions`, using parentheses instead of square brackets. Then we print each element in the tuple individually, using the same syntax we've been using to access elements in a list:

```
200
50
```

Let's see what happens if we try to change one of the items in the tuple `dimensions`:

```
dimensions = (200, 50)
dimensions[0] = 250
```

This code tries to change the value of the first dimension, but Python returns a type error. Because we're trying to alter a tuple, which can't be done to that type of object, Python tells us we can't assign a new value to an item in a tuple:

```
Traceback (most recent call last):
  File "dimensions.py", line 2, in <module>
    dimensions[0] = 250
TypeError: 'tuple' object does not support item assignment
```

This is beneficial because we want Python to raise an error when a line of code tries to change the dimensions of the rectangle.

NOTE *Tuples are technically defined by the presence of a comma; the parentheses make them look neater and more readable. If you want to define a tuple with one element, you need to include a trailing comma:*

```
my_t = (3,)
```

It doesn't often make sense to build a tuple with one element, but this can happen when tuples are generated automatically.

Looping Through All Values in a Tuple

You can loop over all the values in a tuple using a `for` loop, just as you did with a list:

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

Python returns all the elements in the tuple, just as it would for a list:

```
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. For example, if we wanted to change the dimensions of this rectangle, we could redefine the entire 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)
```

The first four lines define the original tuple and print the initial dimensions. We then associate a new tuple with the variable `dimensions`, and print the new values. Python doesn't raise any errors this time, because reassigning a variable is valid:

```
Original dimensions:
200
50

Modified dimensions:
400
100
```

When compared with lists, tuples are simple data structures. Use them when you want to store a set of values that should not be changed throughout the life of a program.

TRY IT YOURSELF

4-13. Buffet: A buffet-style restaurant offers only five basic foods. Think of five simple foods, and store them in a tuple.

- Use a `for` loop to print each food the restaurant offers.
- Try to modify one of the items, and make sure that Python rejects the change.
- The restaurant changes its menu, replacing two of the items with different foods. Add a line that rewrites the tuple, and then use a `for` loop to print each of the items on the revised menu.