

PYTHON CRASH COURSE

3RD EDITION

**A Hands-On, Project-Based
Introduction to Programming**

by Eric Matthes



**no starch
press**

San Francisco

PYTHON CRASH COURSE, 3RD EDITION. Copyright © 2023 by Eric Matthes.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

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.

Numbers

Numbers are used quite often in programming to keep score in games, represent data in visualizations, store information in web applications, and so on. Python treats numbers in several different ways, depending on how they're being used. Let's first look at how Python manages integers, because they're the simplest to work with.

Integers

You can add (+), subtract (-), multiply (*), and divide (/) integers in Python.

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
```

In a terminal session, Python simply returns the result of the operation. Python uses two multiplication symbols to represent exponents:

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

Python supports the order of operations too, so you can use multiple operations in one expression. You can also use parentheses to modify the order of operations so Python can evaluate your expression in the order you specify. For example:

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
```

The spacing in these examples has no effect on how Python evaluates the expressions; it simply helps you more quickly spot the operations that have priority when you're reading through the code.

Floats

Python calls any number with a decimal point a *float*. This term is used in most programming languages, and it refers to the fact that a decimal point

can appear at any position in a number. Every programming language must be carefully designed to properly manage decimal numbers so numbers behave appropriately, no matter where the decimal point appears.

For the most part, you can use floats without worrying about how they behave. Simply enter the numbers you want to use, and Python will most likely do what you expect:

```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
```

However, be aware that you can sometimes get an arbitrary number of decimal places in your answer:

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

This happens in all languages and is of little concern. Python tries to find a way to represent the result as precisely as possible, which is sometimes difficult given how computers have to represent numbers internally. Just ignore the extra decimal places for now; you'll learn ways to deal with the extra places when you need to in the projects in Part II.

Integers and Floats

When you divide any two numbers, even if they are integers that result in a whole number, you'll always get a float:

```
>>> 4/2
2.0
```

If you mix an integer and a float in any other operation, you'll get a float as well:

```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
```

Python defaults to a float in any operation that uses a float, even if the output is a whole number.

Underscores in Numbers

When you're writing long numbers, you can group digits using underscores to make large numbers more readable:

```
>>> universe_age = 14_000_000_000
```

When you print a number that was defined using underscores, Python prints only the digits:

```
>>> print(universe_age)
14000000000
```

Python ignores the underscores when storing these kinds of values. Even if you don't group the digits in threes, the value will still be unaffected. To Python, 1000 is the same as 1_000, which is the same as 10_00. This feature works for both integers and floats.

Multiple Assignment

You can assign values to more than one variable using just a single line of code. This can help shorten your programs and make them easier to read; you'll use this technique most often when initializing a set of numbers.

For example, here's how you can initialize the variables `x`, `y`, and `z` to zero:

```
>>> x, y, z = 0, 0, 0
```

You need to separate the variable names with commas, and do the same with the values, and Python will assign each value to its respective variable. As long as the number of values matches the number of variables, Python will match them up correctly.

Constants

A *constant* is a variable whose value stays the same throughout the life of a program. Python doesn't have built-in constant types, but Python programmers use all capital letters to indicate a variable should be treated as a constant and never be changed:

```
MAX_CONNECTIONS = 5000
```

When you want to treat a variable as a constant in your code, write the name of the variable in all capital letters.

TRY IT YOURSELF

2-9. Number Eight: Write addition, subtraction, multiplication, and division operations that each result in the number 8. Be sure to enclose your operations in `print()` calls to see the results. You should create four lines that look like this:

```
print(5+3)
```

Your output should be four lines, with the number 8 appearing once on each line.

2-10. Favorite Number: Use a variable to represent your favorite number. Then, using that variable, create a message that reveals your favorite number. Print that message.

Comments

Comments are an extremely useful feature in most programming languages. Everything you've written in your programs so far is Python code. As your programs become longer and more complicated, you should add notes within your programs that describe your overall approach to the problem you're solving. A *comment* allows you to write notes in your spoken language, within your programs.

How Do You Write Comments?

In Python, the hash mark (`#`) indicates a comment. Anything following a hash mark in your code is ignored by the Python interpreter. For example:

```
comment.py # Say hello to everyone.  
print("Hello Python people!")
```

Python ignores the first line and executes the second line.

```
Hello Python people!
```

What Kinds of Comments Should You Write?

The main reason to write comments is to explain what your code is supposed to do and how you are making it work. When you're in the middle of working on a project, you understand how all of the pieces fit together. But when you return to a project after some time away, you'll likely have