Python for Teenagers

Learn to Program like a Superhero!

James R. Payne

Apress®

Python for Teenagers

Learn to Program like a Superhero!

James R. Payne

Apress®

Python for Teenagers: Learn to Program like a Superhero!

James R. Payne
Deerfield Beach, FL, USA

ISBN-13 (pbk): 978-1-4842-4549-1 ISBN-13 (electronic): 978-1-4842-4550-7
https://doi.org/10.1007/978-1-4842-4550-7

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Todd Green
Development Editor: James Markham
Coordinating Editor: Jill Balzano

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-4549-1. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To my wife Whitney Payne, for always believing and pretending not to notice when I yell at the computer.

To my parents, Ronnie and Sharon Payne, and my brother, also Ronnie Payne, who all mysteriously have "ron" in their name and who always told me I could become whatever I wanted in life, even when I told them I wanted to be Batman.

To Dorjan Williams, who, years ago, helped me create a universe of ridiculous comic book characters. To Eric Miller, who helps me solve problems big and small, including slaying a dragon in my backyard, so that I can focus – sometimes – on getting work done. Nicholas Rini introduced me to both programming and comic books, and without him, this book would not exist. Nanci Packard and Wendy White provided inspiration with their use of words so big they couldn't possible fit in a book. Thanks to members of the old Dev Shed crew: Jennifer Ruggieri – who got me the job that got me the book – Charles Fagundes, and Keith Lee, for coding help and reminding me (frequently) that my cup overfloweth. Jose Escalante, I thank you here because you were the only one that could see John Cena. Enrique Stone… you know what you did.

A special thanks to Sophie "the Bulldog" Payne for letting me use her likeness in this book and always being such a good helper in the kitchen.

I would be remiss if I failed to thank the mad titan, Thanos, who helped me accomplish so much with just a snap of his fingers. Mister T pitied this fool, while Richard C. helped me "hit em with the hein." And lastly, thank you to a handful of the writers that inspire me: A. Lee Martinez, Neil Gaiman, Frank Miller, Alan Moore, Jim Starlin, and Stephen King – can't you guys write any faster?

Table of Contents

About the Author

James R. Payne was introduced to programming when he was just 10 years old. He started off hacking text-based games like Lemonade Stand to gain an advantage while playing and soon started creating his own text-based role-playing games in the style of Dungeons & Dragons and inspired by his favorite comic books. The enjoyment of those early days stuck with him, and he continues to be drawn back into the programming world throughout his career.

Payne is the former Editor-in-Chief/Community Manager of Developer Shed, an online publication and community consisting of 14 websites and forums dedicated to programming, web development, and Internet marketing. He's written over a thousand articles on coding and marketing, covering virtually every language and platform available. His first book, *Beginning Python* (Wrox Press), was published in 2010. In addition, he has published over 2000 articles covering topics ranging from gaming to aerospace/aeronautics, and also writes adult horror and young adult fantasy books.

Payne decided to write this book to pass on his love of development in the hopes that it would inspire future generations to code. You can find Payne on the web by visiting `www.jamesrpayne.com`.

About the Technical Reviewer

Andrea Gavana has been programming Python for almost 16 years, dabbling with other languages since the late 1990s. He graduated from university with a master's degree in Chemical Engineering, and he is now a Lead Reservoir Engineer working for Total in Copenhagen, Denmark.

Andrea enjoys programming at work and for fun, and he has been involved in multiple open source projects, all Python-based. One of his favorite hobbies is Python coding, but he is also fond of cycling, swimming, and cozy dinners with family and friends.

This is his third book as technical reviewer.

Acknowledgments

This book wouldn't have been possible without Todd Green, who reached out to me to write a book and listened to my ideas and, thankfully, chose the one I wanted to write the most.

Jill Balzano, Coordinating Editor extraordinaire, was invaluable in keeping things rolling during an incredibly busy time, and without her, this book would never have come to fruition either.

James Markham and Andrea Gavana found all of my errors and proved to me that, even at this old age, I still have a lot to learn. Who knew – an old dog can learn new tricks.

Thank you to the entire editorial team at Apress, who were a pleasure to work with and helped me do what I love to do most: write. And make up stupid comic book characters.

Introduction

Who This Book Is For

This book is intended for teenagers looking to program in Python. While that technically means anyone aged 13 through 18, the truth of the matter is, anyone of *any* age can (and should, if I do say so myself!) pick up this book if they want to learn either (a) how to program in Python, (b) how to program as a beginner, or (c) add Python to their current skill set.

Above all, if you are holding this book in your hand, intrepid adventurer, then this book is for *you*. The future is dependent on young heroes like yourself, eager to learn the art of coding and go out into the world and safeguard it from nefarious hackers, dubiously programmed applications, and the rise of artificially intelligent robots!

So whether you are in sixth grade or in college, this book will grant you with super powers galore. Sure, you won't be able to see through walls or lift cars over your head once you finish this book. However, you will be able to speak the language of computers and create some pretty cool programs.

And what could be better than that?

What You Will Learn in This Book

Chapter 1 provides an overview of programming and Python and then shows you how to install Python and a Python IDLE, which will allow you to create your own Python programs and test your code.

In Chapter 2, we will discuss mathematical functions (things like division, addition, and multiplication) and learn about the different *data types* used by Python. We will also begin to build the foundation of a fun super hero generator app – "Super Hero Generator 3000"!

Chapter 3 delves into how to work with text – also known as *strings*. We take a look at the different types of *storage* Python offers as well. We wrap things up by looking at common string functions and build another section of our Super Hero Generator 3000 application.

Sometimes a program will need to take a certain action depending upon feedback from a user or from other influences. This is known as decision making and is the topic of Chapter 4.

Programming logic and loops – known as iterations, where code can "loop" or repeat itself based on certain conditions – are covered in Chapter 5.

Chapter 6 is a refresher course of what you have learned up until this point. We will use all the knowledge we've acquired to finish building the first complete version of Super Hero Generator 3000. By the end, you will be able to randomly create heroes with unique super powers, names, and battle statistics!

In Chapter 7, we begin to learn more advanced techniques. To be a real coder, you must learn efficiency and reduce mistakes in your code. That is where *modules* and *built-in functions* come into play. Learn what they are and why they will make your life a whole lot easier in this exciting chapter!

Chapter 8 looks at even more advanced topics: specifically, we will cover the basics of object-oriented programming (OOP) and cover objects and classes and define a thing called *polymorphism*.

To switch things up a little bit, Chapter 9 will look at some different types of data structures, including tuples and dictionaries.

Chapter 10 brings us up to speed on how to create – and work with – files inside of directories.

One of my personal favorite chapters is Chapter 11, which covers a topic that is near and dear to my heart: Python for Gaming. We will stroll through the world of video games and learn how to work with video game elements, including sound, animation, and more!

Learning how to create games that interact with a users actions and making images move within a game are truly what make games enjoyable. Chapter 12 continues the topic of gaming and focuses specifically on game animation.

In Chapter 13 – don't worry, in this case 13 is lucky, for you at least! – we move into areas of Python we have not yet discussed that do not fit in their own chapter. This includes how to *debug* – or find broken code. We also look at advanced modules and other topics.

Finally, we sum everything up in Chapter 14 and cover a wide range of topics, including how to find work as a Python programmer, common interview questions, the future of Python, and career paths, and answer some of the *frequently asked questions (FAQs)* about our favorite programming language.

So now that we know what we will learn, let's put on our cape and super hero outfits and get ready to leap tall buildings – of knowledge.

Why I Started Programming

I started programming a long, long time ago – back before the Internet or cell phones existed and when wild dinosaurs roamed the earth. Back then, computers didn't have images on them like they do today. Everything was text-based – even most of our games – the horror! While we did have some computer games with animation and graphics, they were 8-bit and not cinematic like the ones of today.

I was fortunate enough to share a computer with my older brother. I'm pretty certain my parents didn't know what a computer was used for, but must have thought: "This future-device will surely make my children Men… of… the… Future… future… future… future…" (just pretend the word is echoing).

And to some degree, they were correct: if they hadn't purchased my brother and me a computer, who knows what I would be doing with my life right now? Certainly not writing this book and helping you to program like a hero!

But having a giant paperweight made of jumbled electronics – back then we called it an Apple IIe – wasn't enough to entice me to use it all that much. After all, I also happened to own a Nintendo Entertainment System (NES) as well and it had an amazing slew of games that I still – embarrassingly – play to this day.

What really got me into computers was this: I had a friend, Nicholas, who knew all about programming computers. He showed me one day how to "hack" into the code of a few of our favorite text-based games to give ourselves an advantage. It was akin to creating your very own cheat code in a video game. In particular, we played a game called Lemonade Stand, which was exactly the same as standing outside your house and selling homemade Lemonade, only you never made real money and you didn't get a sunburn.

In the game, you started out with a couple of dollars – barely enough to make any real profit. However, once we looked at the code running the game, we figured out that we could start out with however much money we wanted if we just changed a few words around. Soon enough, I was the world's first millionaire Lemonade Stand mogul.

I was hooked.

From there, it was not a far stretch to conceive that we could actually create our own video games and that is exactly what we did. From complex role-playing games (RPG) based off of our favorite comic books and Dungeons & Dragons to programs that would ask our friends a series of questions and then make fun of them based off of their answers – shenanigans!

While all of that seemed silly at the time, looking back on it I now know that it helped set the foundation for my love of programming and, to a degree, writing (though I began writing much earlier than that). Without that summer of programming fun, I would never have had the wonderful experiences, friends, jobs, and writing opportunities that have come my way ever since.

And, mostly, I would never have had the fun of programming either.

That is what I am hoping to pass on to you, dear reader: a lifetime love of programming and opportunities all based off of one thing – the fun and joy of writing computer programs and writing code.

Sure, programming applications can be a pain in the butt. You will find yourself banging your head against a keyboard on many nights and yelling at the computer screen for hours only to find that your program isn't working because you forgot a parenthesis () somewhere.

But – once you find that mistake that you or another programmer made – there isn't quite anything like that triumphant moment when you realize that you – YOU – are the greatest coder of all time!

Programming Dos and Don'ts

When reading this book, you may find yourself feeling the urge to skip ahead a little or might want to skip an exercise or two. As in all things in life, this piece of advice holds true in learning to program as well: if you cheat, you are only cheating yourself.

To help keep you on the straight and narrow, here are some dos and don'ts for reading this book and for learning how to program, in general:

Do read the book straight through. While you might be okay to skip a chapter or an exercise here or there, keep in mind that this book is all about building a foundation of not just coding language, but coding practices, theory, and an understanding of programming principles that you can take with you that apply to *other* languages as well.

Don't copy and paste code from this book or any other source (assuming you have a digital copy). Instead, take the time to type in the code so that you can begin to get a feel for writing code and, perhaps, commit some of the code to memory through repetition.

Do experiment with code. One of the best ways, I've found, to learn how to *truly* code is to experiment. If you come across an example in the book, feel free to change the parameters some and see what happens. The worst that can happen is that you can fail. The best? That you learn something new!

Don't be afraid to Google other tutorials and how-tos on Python. This book is supposed to build a beginner's foundation, but it does not teach you everything there is to know – that's what the sequel is for! If you do decide to look up comparative examples, be certain to look at the date of the article and the *version* of Python. If the version does not match the version we are using in this book (Python 3), odds are your code will not work and you will find yourself very confused.

Do document your code. We have not covered this topic – yet – but for now, know that *documentation* means to leave little comments in *blocks*, or sections, of your code that lets you (or another coder in the future) know what you intended to do with a certain section of code. While Python is a very readable language, the way every programmer codes is different, and what might be apparent to you is not always apparent to others. Also, if you have to review your own code at a later date, it will make it easier for you to remember what, exactly, you were trying to do at 4 a.m. 10 years ago!

Do plan out your code. That is, write down how you want your overall program to work and then break that down into little sections. Then, take those little sections and map out what you need to code for each part. This way you will have a roadmap to follow and won't just be coding by the seat of your pants.

Finally, *do* test your code frequently and save your work often. When we programmers are in the thick of things, we like to carry on, plugging away, for hours at a time. However, if we don't stop to test our code and save our files, we risk losing hours of work and, worse, creating a program with problems that are difficult to trace.

CHAPTER 1

Introduction to Computer Programming and Python

Computer programming – commonly referred to as "coding" by the cool kids – is the art of creating applications or software. These programs allow us to do everything from solve simple math problems and watch our favorite YouTube videos (I can't get enough of skydiving bulldogs) to destroying hordes of rampant aliens in our favorite video games and even launching a real-life spaceship into outer space.

I call computer programming an "art" because it *is*. Anytime you create something, you are indulging in an art form. Sure, computer code, the words we type into a shell to create our programs (more on this later!), may not be pretty to look at for the common person on the street – your code will never see the inside of an art exhibit most likely – but when a part of your program does what you created it to… there is almost nothing more magical.

Well, maybe those skydiving bulldogs.

A computer program can come in many shapes and sizes. In addition to an application that runs on your desktop system or a game that plays on your favorite video game console, programs also take the form of mobile apps on a cell phone. You can even find pieces of software that operate things like refrigerators, your mom's minivan, and even something as simple as a toaster oven.

And robots. Armies of robots.

But more on that later.

For now, know that a computer program is a set of code, created in a *programming language*, that tells a device to carry out a set of instructions.

J. R. Payne, *Python for Teenagers*, https://doi.org/10.1007/978-1-4842-4550-7_1

Programming Language Overview

As mentioned, a computer program is written using a programming language. Just like the real language you, I, and the rest of the world speak every day, computer languages come in all shapes and sizes. While most of them make sense to the trained eye, a newcomer to code would sound like a crazy person spouting gibberish if they tried to use it in everyday conversation. That dialogue might look something like this:

```
Normal Person: Hello, how are you?
Programmer (You): Print I am fine! Input, how are you?
```

Fortunately for all involved, computers are fluent in programming languages (thanks, in part, to our friend *the compiler* – but more on this later!) and can easily understand the most complex of sentences you type in.

For the purpose of this book, we will stick to one of the most versatile, yet easy-to-learn, languages, *Python*. While the name sounds frightening, keep in mind, it could be worse: it could be called Cobra. In truth, the language was not named after a reptile at all, but, instead, an old television comedy from Britain called *Monty Python and the Flying Circus*.

Here's your first assignment: Go ask your parents about that show. See you in a few hours!

Oh, you are back. Great. Did what they said make any sense? Probably not. But that's okay; you don't need to understand the complexities of British comedy to learn how to program using this book. All you need is a desire to learn, a computer, and the pages in front of you.

Python Overview

Python is what is known as a high-level, dynamic, interpreted, object-oriented programming language. While all of that may sound a bit intimidating, never fear! By the end of this book, you will be able to impress your friends with sentences much more daunting than the one above! All that statement really means is that Python is not a basic machine-level language, and as such, it needs an "interpreter" to "compile" it to machine language so that the computer can understand what it is you are trying to tell it.

This interpreter takes your code and converts it – or *compiles* it – into a series of 1s and 0s that a computer can plainly understand. All of that happens in the background, so don't worry if you do not quite understand it just yet.

Python is a relatively new programming language that was created in the late 1980s – back when your dad had a big funny mustache and your mom listened to bands with names like *Wham!* and *Poison*.

The man that created the language was a computer genius named Guido Van Rossum, who was bestowed with the fancy, nonsensical title, Benevolent Dictator for Life. Like technology, programming languages evolve as well, and Python is no different. It has gone through several versions over the years and is currently known as Python 3.

How Does Python Differ from Other Programming Languages?

Python differs from other programming languages in a number of different – yet important – ways. For starters, Python is typically easier to learn and use than languages in the same class, such as Java and C++. Programs created in Python also take less time to create, as it requires less code (in general). This is due, in part, to Python's *data types* – a term we will cover in great detail in an upcoming chapter.

Python is also extremely versatile. While it may not be the primary choice, Python *can* be used for applications in virtually every arena, including gaming, desktop software, mobile apps, and even virtual reality. It is also a must for network programming and an essential tool in a computer security toolbox.

The Benefits of Python

Python is currently the most-used programming language in the world today and is the fastest growing as well. And with good reason. Below are just a few ways in which Python can benefit a programmer:

- Increased productivity: By some reports, Python can increase a programmer's productivity – how much work they can accomplish in a given time – by as much as ten times! It literally is faster than a speeding bullet!

- Extensibility: One great advantage of Python is the fact that it has a very extensive library of, well, libraries. A library is a set of existing code you can add-in to your program. These libraries cover things that are common features of a program and save you from having to write the code over and over again yourself. For example, instead of having to write a section of code to perform a complicated mathematical equation, you can supply use a library and save yourself a huge headache.
- Python is easy to read: One tough part of being a programmer is the fact that, sometimes, your code does not work. When that happens, you might find yourself re-reading your code – or worse, someone else's – to try and figure out why your program is not behaving as it should. Fortunately, Python is easy to read and most of the language makes sense at a glimpse. This makes finding issues a lot easier than more complicated languages.
- Portability: Python runs on many platforms and systems, meaning your programs can reach a wider audience.
- Internet of Things (IoT): The Internet of Things may sound like a magical world full of digital beasts, and in some ways, it is. The IoT consists of smart objects – light switches, doorknobs, toaster ovens, appliances – that you find in your everyday home. These household appliances are controllable by voice commands and mobile devices, making them more interactive than their primitive predecessors. I mean sure, your mom and dad yelled at the dishwasher all the time – but did it ever listen? Now, thanks to the IoT and languages like Python, it can! You still have to put your dishes inside of it, but still!
- Python frameworks: Frameworks are like skeletons for a program – they allow you to quickly set up the basics for certain types of applications without needing to code common elements that usually exist in the type of software you are developing. This saves programmers time and reduces the number of errors that can occur when you have to manually code. Python is supported by a large number of frameworks that can make launching a new program very rapid indeed!

- Python is fun: Python is a fun language to learn; as stated, not only is it easy to get started, but the Python community host many fun events and challenges. For example, many people write their Python code in poetry form and there are numerous Python "challenges" released every year to help test a coder's skills.
- Python is flexible: Because Python has so many uses and is used by so many companies around the world, finding a job after learning Python is easier than with other languages. In addition, if you do not like a given field, you can always use your Python skills to try a different path. For example, if you find that coding applications is boring, you could switch to network administration or work at an IT security firm.

And those are just a few of the benefits and advantages that Python offers.

Examples of Python in the Wild

While it is impossible to say just how many companies around the world use Python, there are a number of interesting businesses that rely on the language. Below is just a smattering of them:

- Wayne Enterprises (Batman's Alter Ego's corporation): Well, we don't really know that, but wouldn't that be cool!
- Google: The search engine giant and one day ruler of the galaxy, Google, has been using Python since its inception, partially because developers can build programs so quickly with it and also because the code is easy to maintain.
- Facebook and Instagram: While Python is not the only language used at these two social media platforms, it is one of their most important ones. Facebook uses Python, in part, thanks to its extensive libraries. Instagram, meanwhile, is a firm supporter of one of Python's main web frameworks – Django. We cover web frameworks in great detail later in this book.

- Netflix: If you are a fan of streaming movies, then you are no stranger to Netflix. The company uses Python primarily for its data-analysis capabilities and for security purposes – among other areas.
- Video games: Battlefield 2 and Civilization 4 are just two video games that both rely on Python. Interestingly enough, Civilization uses Python for, among other things, its artificial intelligence (AI) scripts.
- Government agencies and institutions: Government agencies and institutions including NASA, The National Weather Service, and the CIA all use Python – though how it is used is Top Secret! Meet us in the garage with a briefcase full of money and we'll tell you all about it!

Your First Python Program

By now, you are probably wondering what Python code looks like. Well, fear not! I am going to show you a sample snippet. Later, after we install Python and an IDLE (integrated development environment) on your computer, you can try and execute – or run – the code to see it in action. For now, though, I thought it would be a good idea to just give you a taste before delving any further into the language.

Traditionally, when a programmer writes their first *ever* line of code, they create a program called, "Hello, World," as a metaphorical way to introduce themselves to the world. However, as budding super heroes – or villains (no judgment here) – we need something a little flashier.

Behold, your first Python program!

```
print("Look up in the sky! Is it a bird? Is it a plane?")
print("Dun dun dun dun dun dun dun dun dun dun dun dun dun dun dun dun")
print("No you dummy. That's just some guy flying around in his pajamas. Now
get back to work!")
```

If you were to run this code, the result would be:

```
Look up in the sky! Is it a bird? Is it a plane?
Dun dun dun dun dun dun dun dun dun dun dun dun dun dun dun dun

No you dummy. That's some guy flying around in his pajamas. Now get back
to work!
```

Let's examine the code a little more closely. The part that says *print()* is known as a *function*, whose job it is to tell the computer to – in this case – *print* something to the user's screen. The text in between the opening and closing parentheses () is the *parameter* that we are providing the function. The characters in between the quotation marks " " are known as a string.

Don't worry if this doesn't make all the sense in the world just yet – we go over this topic in great detail in the next chapter. For now, just know that this is what Python code looks like. Odds are, you were able to tell exactly what this program would do *before* I told you; that is just one of the things that make Python so great – its readability!

Installing Python

In this section, we are going to learn how to install Python on the various *operating systems*. An operating system is a piece of software that lets you interact with a computer. You are probably familiar with the more popular ones, such as *Microsoft Windows* (if you own a PC) and *Mac OS X* if you own an Apple computer. The version of Python you install will vary depending upon which one of these your computer uses. In addition, we will learn how to install Python on *Linux* and *Ubuntu* systems as well.

Installing Python on Windows

To begin, open up a web browser and navigate to the official Python website and its download page: `www.python.org/downloads/` (Figure 1-1).

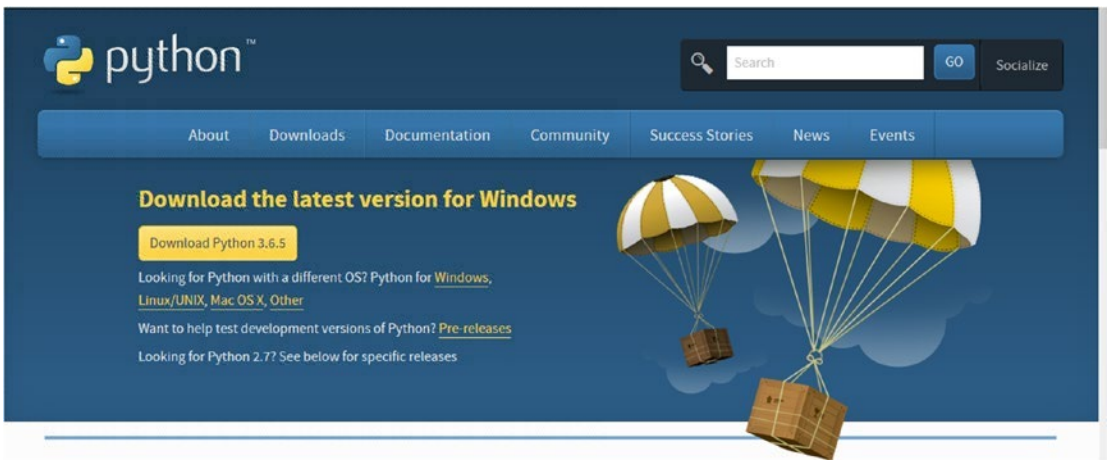

Figure 1-1. *Python.org website*

The current version of Python is 3.6.5; by the time you read this book, it may be higher than that. Whatever the case, click the "Download Python" button under the *Download the Latest Version for Windows* header. Optionally, you could scroll down and download previous versions (just make sure they are version 3.X or higher, as there are incompatibility issues between versions 2.X and 3.X); however, for the purposes of this book, it is always best to use version 3.6.5 or later.

An image will appear asking if you would like to save the file. Click "Save File" (Figure 1-2) and save it to your Desktop or somewhere easily remembered.

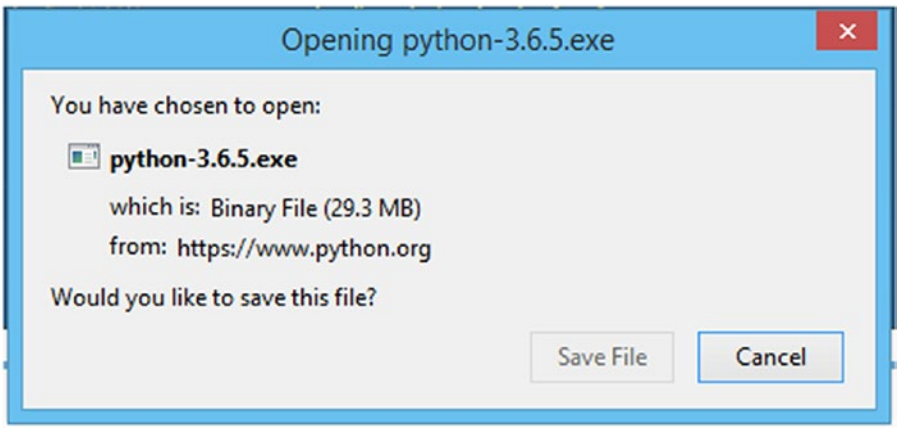

Figure 1-2. *Save file dialogue for Python installation files*

Navigate to your Desktop (or the location where you saved the file) and double-click it. It should appear similar to the image in Figure 1-3.

Figure 1-3. *Python .EXE install file icon*

The installer will launch and will ask you whether you wish to "Install Now" or "Customize Installation." For ease, we are going to allow the installer to "Install Now." Before you click that button, however, make sure that "Install launcher for all users" and "Add Python 3.6 to PATH" are both checked. Then click the "Install Now" option (Figure 1-4).

Figure 1-4. *Python install setup screen*

You may get a pop-up from Windows asking for permission to continue the installation. If so, allow the program to continue. A new pop-up will appear, showing you the Setup Progress (Figure 1-5):

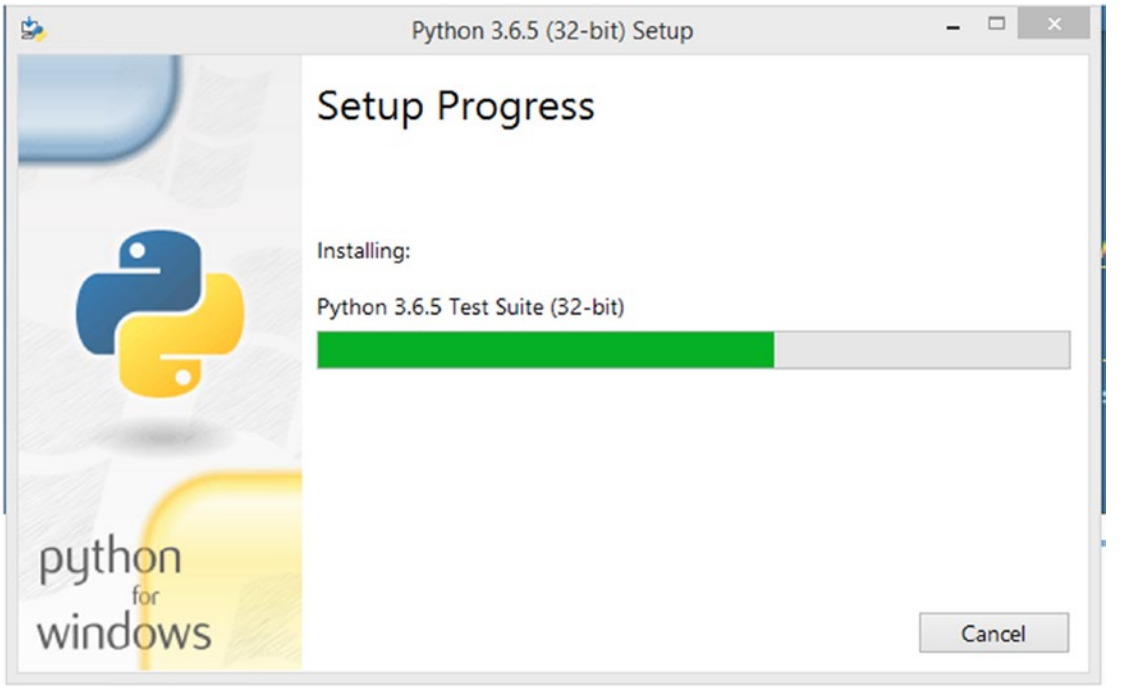

Figure 1-5. *Python installation progress screen*

Once Setup is complete, you will see a screen like the one below. Click the "Close" button to complete installation (Figure 1-6).

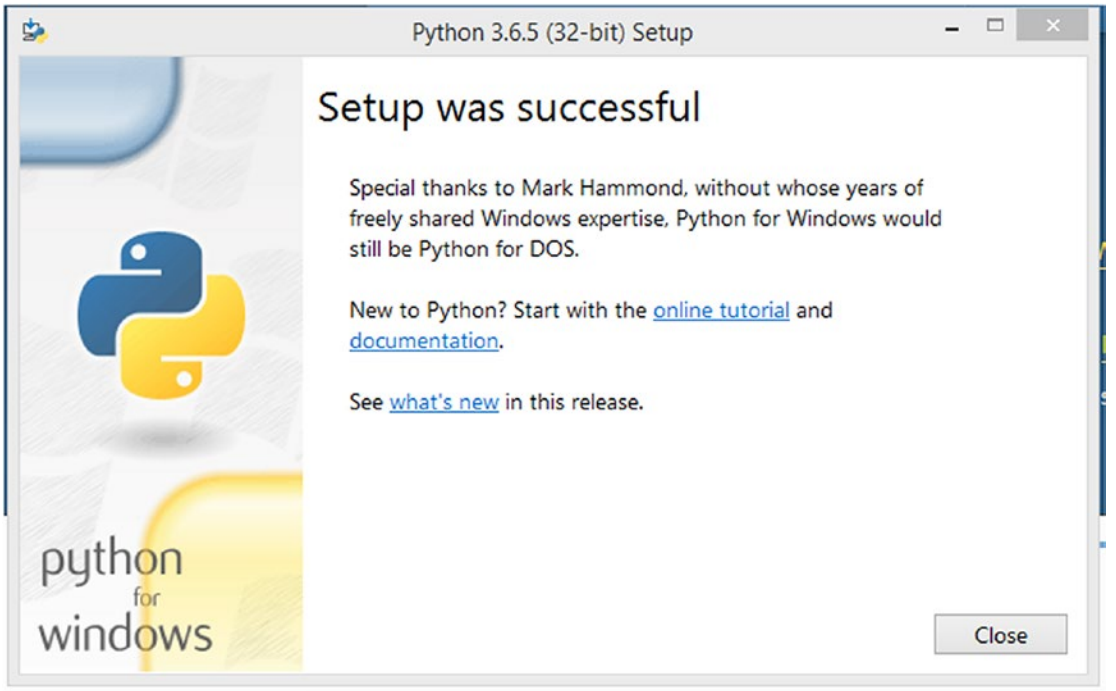

Figure 1-6. *Python install setup successful window*

You should now have Python installed on your computer. You can find it in your "Start" menu, labeled Python 3.6 (or whichever version you installed).

When you launch Python, the first thing you see is the shell, which is a piece of the development environment where you can write a line of code, test code, run code, and create Python files. Figure 1-7 shows an example of how the Python Shell will appear once launched.