Lesson 1. Why should you learn how to program?

After reading lesson 1, you'll be able to

- Understand why programming matters
- Set up a plan for learning how to program

1.1. WHY PROGRAMMING MATTERS

Programming is universal. No matter who you are or what you do, you can learn to write programs that can help make your life easier.

1.1.1. Programming isn't just for professionals

A misconception, both for veteran programmers and for people who have never programmed before, is that after you start to learn how to program, you'll have to continue until you become a professional programmer. Likely, this misconception stems from associating programming with incredibly complex systems: your operating system, car/aviation software, artificial intelligence that learns, and many others.

I think of programming as a skill, like reading/writing, math, or cooking. You don't have to become a best-selling author, a world-class mathematician, or a Michelin star chef. Your life significantly improves with a little bit of knowledge in each of those subjects: if you know how to read and write, you can communicate with others; if you know how to do basic calculations, you can tip appropriately at a restaurant; if you know how to follow a recipe, you can make a meal in a pinch. Knowing a little bit of programming will enable you to avoid having to rely on others to help you and will enable you to finish tasks you may want to do in a specific way more efficiently.

1.1.2. Improve your life

If you learn to program, your skill can be used as you effectively build your own personal toolbox of utilities. The more you try to integrate programming into your daily life, the more you'll be able to solve personal tasks more efficiently.

To keep up with your skill, you can write custom programs that fit your daily needs. The benefit of writing your own programs instead of using ones that already exist is that you can customize them to your exact needs. For example:

- Do you keep track of the checks that you write in the paper logbook that came with your checkbook? Consider typing them in a file and writing a program that reads the file and organizes the information. With programming, after the data is read, you can calculate sums, separate the checks by date ranges, or whatever else you want.
- Have you taken pictures and downloaded them to your computer, but the names given by the camera software aren't what you want? Instead of manually renaming everything by hand for a thousand pictures, you can write a short program that renames all files automatically.
- Are you a student preparing for the SAT and want to make sure your solution for the quadratic equation is correct? You can write a program that takes in missing parameters and solves the equation for you, so that when you do it by hand, you can be sure that the calculations were done correctly.
- Are you a teacher who would like to send a personalized email to each student with that student's grade for a test? Instead of copying and pasting text and filling in the values manually, you can write a program that reads the student name, email address, and score from a file, and then effectively fills in the blank automatically for each student, and sends out the email.

These are just a few situations in which programming can help you to be more organized and self-reliant.

1.1.3. Challenge yourself

ming is also creative. After you become familiar with a few ways to do one task in programming, you get to make decisions about which way would be best to apply. For example, if you're reading a file, do you want to read all the data at once, store it, and then do some analysis, or do you want to read the data one piece at a time and analyze as you go along?

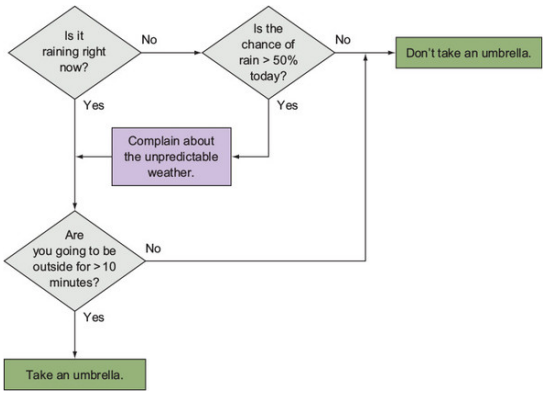
By making these kinds of decisions with the knowledge you gain, you challenge yourself to think critically about what you want to achieve and how to do it most efficiently.

1.2. WHERE YOU ARE NOW AND WHERE YOU'LL BE

This book doesn't assume that you've programmed before. Having said that, you should be familiar with the following:

- *Understanding a variable*—If you took a math course that covers introductory algebra, you should know what a variable is. In the next unit, you'll see how variables in a programming setting are different.
- *Understanding true/false statements*—You can think of statements as sentences that can be determined to be true or false. For example, "it is raining" is a statement that's either true or false. You can also invert statements to take the opposite truth value by using the word *not*. For example, if "it is raining" is true, then "it is not raining" is false.
- *Connecting statements*—When you have more than one statement, you can connect them by using the words *and* or *or*. For example, if "it is raining" is true and "I am hungry" is false, then "it is raining and I am hungry" is false because both parts need to be true. But "it is raining or I am hungry" is true because at least one of the parts is true.
- *Making decisions*—When you have multiple statements, you can make decisions based on whether one statement is true by using *if...then*. For example, "if it is raining, then the ground is wet" is made up of two statements: "it is raining" and "the ground is wet." The statement "the ground is wet" is a consequence of the statement "it is raining."
- *Following flowcharts*—You won't need to know flowcharts to understand this book, but understanding them requires the same skills as understanding basic programming. Other ideas that use the same skill set are playing the game of 20 Questions, following a recipe, reading a choose-your-own-adventure book, or understanding algorithms. You should be familiar with following a set of instructions and making branching decisions. Flowcharts show a list of instructions that flow from one to the next and allow you to make decisions, which lead to different paths. In a flowchart, you're asked a series of questions, whose answer is one of two choices: yes or no. Depending on your answer, you follow certain paths through the flowchart and will eventually end up at a final answer. Figure 1.1 is an example of a flowchart.

Figure 1.1. Flowchart for deciding whether to take an umbrella today



Knowing the preceding skills is all you need to begin your programming journey. After reading this book, you'll know the basics of programming. The basic concepts you'll learn that can apply to any programming language are as follows:

- Using variables, expressions, and statements in programming
- Getting the program to make decisions based on conditions
- Getting the program to automatically repeat tasks under some conditions
- Reusing operations built into the language to help you be more efficient
- Making your code more readable and easy to maintain by breaking a larger task into smaller ones
- Knowing which data structure (a structure already created that can store information in a certain format) is appropriate to use in different situations

You'll be learning how to program by using a language called Python (version 3.5). Any knowledge gained about programming concepts will be easily

book, you'll be familiar with the details of the Python programming language. You'll know the following:

- How to use the syntax of the language (in English, the equivalent is how to form valid sentences).
- How to write more-complex programs with different blocks of code working together in harmony (in English, the equivalent is writing a short story).
- How to use code that other programmers wrote (in English, the equivalent is referencing someone else's work so you don't have to rewrite it).
- How to effectively check that your program works, including testing and debugging (in English, the equivalent is checking for spelling and grammar errors).
- How to write programs that interact with the keyboard and mouse.
- How to write more data-centric or mathematical programs.

1.3. OUR PLAN FOR LEARNING HOW TO PROGRAM

Individual motivation is one of the greatest make-or-break factors when learning a programming language. Taking things slow, getting a lot of practice, and allowing time to absorb the material will make the road to success less bumpy.

1.3.1. First steps

If you've never programmed before, this book is for you. This book is separated into units. A *unit* is a set of lessons that all deal with one particular concept in programming. The first lesson in the unit is usually a motivating lesson. The last lesson in a unit is a capstone project, which introduces a real-life problem or task. You can attempt the capstone on your own or you can read the walk-through of the solution; it's intended to make sure that you're on track with understanding the concepts.

You'll have many opportunities to practice what you read. At the beginning of each lesson, you'll see a simple exercise, called *Consider this*, that will get you thinking about the world around you and the way you interact with it; this exercise introduces you to the main idea of the lesson. It's described without code jargon and hints at the kinds of programming ideas you'll learn in the lesson. Throughout the lesson, you'll discover how to "translate" the English description of the outlined exercise into code. Each lesson contains many exercises to help you understand the concepts; doing all the exercises will help the concepts click. Answers to these exercises will be found in Appendix A so that you can check your work.

Being hands-on with the exercises is important in the first few lessons, as you'll be learning the basics of programming using Python. In the last few lessons, you'll see packages that other programmers wrote, and you'll have an opportunity to learn how to use those packages to build more-complex programs. One of the packages will get you to build programs that you can interact with visually, by mouse click or keyboard input, and you'll see your program update an image on the screen. Another package will show you how to deal with data as input. You'll learn how to read files that have a certain structure, how to analyze the data gathered, and how to write data to another file.

1.3.2. Practice, practice, practice, practice

Each lesson has short exercises with solutions. With Python, and programming in general, lots of practice is essential to truly understand the concepts—this is especially true if you've never programmed before. Don't be frustrated by errors when writing a program; you'll improve your understanding by correcting unexpected mistakes.

You can think of these exercises as checkpoints to help you understand how much you understand. Programming isn't a passive activity. You should be actively engaged with the material presented and the concepts shown by constantly trying things out on your own. The checkpoint exercises touch upon the important ideas presented in the lesson, and you should attempt them all to cover the breadth of the material. If you feel adventurous, you can even come up with variations on the exercises presented and attempt to write new programs for problems you come up with!

1.3.3. Think like a programmer

This book is intended to be a unique learning experience. I don't just want to teach you programming in Python. I also want to teach you how to think like a programmer.

To understand this, consider the following metaphor. There are two people: an author of fiction and a journalist. The author of fiction is someone who comes up with a plot, characters, dialogue, and interactions and then puts these ideas together in interesting ways by using the rules of the English language. The author writes a story for people to enjoy. The journalist doesn't need to employ their creative side but rather hunts down stories based on fact. The journalist then puts the facts on paper, also using the rules of the English language, for people to be informed.

puter scientist and a programmer know how to write computer code, and both adhere to the rules of a programming language in order to create programs that do certain tasks. In the same way that an author thinks about a unique story and how to best pace it, a computer scientist may put more effort into coming up with ideas rather than putting their ideas into words. A computer scientist thinks about brand-new algorithms or studies theoretical questions, such as what a computer can and can't do. On the other hand, a programmer implements programs based on preexisting algorithms or a set of requirements to which they must adhere. A programmer knows the details of a language well and can implement code quickly, efficiently, and correctly. In practice, the roles of a programmer and computer scientist often overlap, and there isn't always a clear distinction.

This book will show you how to implement tasks on a computer by giving the computer detailed instructions and will help you become proficient at doing this.

Thinking like a programmer

Be on the lookout throughout the rest of the book for this box.

You'll get useful tips on which principles of thinking like a computer programmer apply to the ideas being discussed. These principles tie the book together, and I hope that revisiting these ideas will help you get into the mindset of a programmer.

The next lesson outlines several principles that get at what it means to think like a programmer. Throughout each lesson, you'll be reminded of these principles whenever possible, and I hope that you'll start to think about these principles on your own as you progress through the book.

SUMMARY

In this lesson, my objective was to inspire you to learn to program. You don't have to become a professional programmer. Use basic programming ideas and concepts to improve your personal life, even in simple ways. Programming is a skill, and you'll get better at it the more you practice. As you read this book, try to think of tedious tasks that you're doing manually that can be solved more effectively with programming, and try to do it.

Let's begin!

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
Unit 0. Learning how to program

NEXT
Lesson 2. Basic principles of learning a programming language