LEARNING MADE EASY

2nd Edition

Algorithms for dummies®

A Wiley Brand

Understand how data drives algorithms

Create your own algorithms using Python

Discover how algorithms are used in the real world

John Paul Mueller
Author of *Artificial Intelligence For Dummies*

Luca Massaron
Google Developer Expert in machine learning

Algorithms

2nd Edition

by John Paul Mueller and Luca Massaron

for dummies®
A Wiley Brand

Algorithms For Dummies®, 2nd Edition

Published by: **John Wiley & Sons, Inc.,** 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

Library of Congress Control Number: 2022934261

ISBN: 978-1-119-86998-6; 978-1-119-86999-3 (ebk); 978-1-119-87000-5 (ebk)

Contents at a Glance

Table of Contents

Introduction

You need to learn about algorithms for school or work. Yet, all the books you've tried on the subject end up being more along the lines of really good sleep-inducing aids rather than texts to teach you something. Assuming that you can get past the arcane symbols obviously written by a demented two-year-old with a penchant for squiggles, you end up having no idea of why you'd even want to know anything about them. Most math texts are boring! However, *Algorithms For Dummies,* 2nd Edition is different. The first thing you'll note is that this book has a definite lack of odd symbols (especially of the squiggly sort) floating about. Yes, you see a few (it is a math book, after all), but what you find instead are clear instructions for using algorithms that actually have names and a history behind them and that perform useful tasks. You'll encounter simple coding techniques to perform amazing tasks that will intrigue your friends. You can certainly make them jealous as you perform feats of math that they can't begin to understand. You get all this without having to strain your brain, even a little, and you won't even fall asleep (well, unless you really want to do so). New in this edition of the book are more details about how algorithms work, and you even get to create your own basic math package so that you know how to do it for that next job interview.

About This Book

Algorithms For Dummies, 2nd Edition is the math book that you wanted in college but didn't get. You discover, for example, that algorithms aren't new. After all, the Babylonians used algorithms to perform simple tasks as early as 1,600 BC. If the Babylonians could figure this stuff out, certainly you can, too! This book actually has three things that you won't find in most math books:

- Algorithms that have actual names and a historical basis so that you can remember the algorithm and know why someone took time to create it
- Simple explanations of how the algorithm performs awesome feats of data manipulation, data analysis, or probability prediction
- Code that shows how to use the algorithm without actually dealing with arcane symbols that no one without a math degree can understand

Part of the emphasis of this book is on using the right tools. This book uses Python to perform various tasks. Python has special features that make working with algorithms significantly easier. For example, Python provides access to a huge array of packages that let you do just about anything you can imagine, and more than a few that you can't. However, unlike many texts that use Python, this one doesn't bury you in packages. We use a select group of packages that provide great flexibility with a lot of functionality but don't require you to pay anything. You can go through this entire book without forking over a cent of your hard-earned money.

You also discover some interesting techniques in this book. The most important is that you don't just see the algorithms used to perform tasks; you also get an explanation of how the algorithms work. Unlike many other books, *Algorithms For Dummies*, 2nd Edition enables you to fully understand what you're doing, but without requiring you to have a PhD in math. Every one of the examples shows the expected output and tells you why that output is important. You aren't left with the feeling that something is missing.

Of course, you might still be worried about the whole programming environment issue, and this book doesn't leave you in the dark there, either. This book relies on Google Colab to provide a programming environment (although you can use Jupyter Notebook quite easily, too). Because you access Colab through a browser, you can program anywhere and at any time that you have access to a browser, even on your smartphone while at the dentist's office or possibly while standing on your head watching reruns of your favorite show.

To help you absorb the concepts, this book uses the following conventions:

- Text that you're meant to type just as it appears in the book is in **bold**. The exception is when you're working through a step list: Because each step is bold, the text to type is not bold.
- Words that we want you to type in that are also in *italics* are used as placeholders, which means that you need to replace them with something that works for you. For example, if you see "Type ***Your Name*** and press Enter," you need to replace *Your Name* with your actual name.
- We also use *italics* for terms we define. This means that you don't have to rely on other sources to provide the definitions you need.
- Web addresses and programming code appear in `monofont`. If you're reading a digital version of this book on a device connected to the Internet, you can click the live link to visit that website, like this: `http://www.dummies.com`.
- When you need to click command sequences, you see them separated by a special arrow, like this: File ➪ New File, which tells you to click File and then New File.

Foolish Assumptions

You might find it difficult to believe that we've assumed anything about you — after all, we haven't even met you yet! Although most assumptions are indeed foolish, we made certain assumptions to provide a starting point for the book.

The first assumption is that you're familiar with the platform you want to use, because the book doesn't provide any guidance in this regard. (Chapter 3 does, however, tell you how to access Google Colab from your browser and use it to work with the code examples in the book.) To give you the maximum information about Python with regard to algorithms, this book doesn't discuss any platform-specific issues. You really do need to know how to install applications, use applications, and generally work with your chosen platform before you begin working with this book.

This book isn't a math primer. Yes, you see lots of examples of complex math, but the emphasis is on helping you use Python to perform common tasks using algorithms rather than learning math theory. However, you do get explanations of many of the algorithms used in the book so that you can understand how the algorithms work. Chapters 1 and 2 guide you through a what you need to know in order to use this book successfully. Chapter 5 is a special chapter that discusses how to create your own math library, which significantly aids you in understanding how math works with code to create a reusable package. It also looks dandy on your resume to say that you've created your own math library.

This book also assumes that you can access items on the Internet. Sprinkled throughout are numerous references to online material that will enhance your learning experience. However, these added sources are useful only if you actually find and use them. You must also have Internet access to use Google Colab.

Icons Used in This Book

As you read this book, you encounter icons in the margins that indicate material of interest (or not, as the case may be). Here's what the icons mean:

TIP

Tips are nice because they help you save time or perform some task without a lot of extra work. The tips in this book are time-saving techniques or pointers to resources that you should try so that you can get the maximum benefit from Python, or in performing algorithm-related or data analysis–related tasks.

WARNING

We don't want to sound like angry parents or some kind of maniacs, but you should avoid doing anything that's marked with a Warning icon. Otherwise, you might find that your application fails to work as expected, you get incorrect answers from seemingly bulletproof algorithms, or (in the worst-case scenario) you lose data.

TECHNICAL STUFF

Whenever you see this icon, think advanced tip or technique. You might find these tidbits of useful information just too boring for words, or they could contain the solution you need to get a program running. Skip these bits of information whenever you like.

REMEMBER

If you don't get anything else out of a particular chapter or section, remember the material marked by this icon. This text usually contains an essential process or a bit of information that you must know to work with Python, or to perform algorithm-related or data analysis–related tasks successfully.

Beyond the Book

This book isn't the end of your Python or algorithm learning experience — it's really just the beginning. We provide online content to make this book more flexible and better able to meet your needs. That way, as we receive email from you, we can address questions and tell you how updates to Python, or its associated add-ons affect book content. In fact, you gain access to all these cool additions:

- **Cheat sheet:** You remember using crib notes in school to make a better mark on a test, don't you? You do? Well, a cheat sheet is sort of like that. It provides you with some special notes about tasks that you can do with Python, Google Colab, and algorithms that not every other person knows. To find the cheat sheet for this book, go to `www.dummies.com` and enter *Algorithms For Dummies, 2nd Edition Cheat Sheet* in the search box. The cheat sheet contains really neat information such as finding the algorithms that you commonly need to perform specific tasks.
- **Updates:** Sometimes changes happen. For example, we might not have seen an upcoming change when we looked into our crystal ball during the writing of this book. In the past, this possibility simply meant that the book became outdated and less useful, but you can now find updates to the book, if we make any, by going to `www.dummies.com` and entering *Algorithms For Dummies, 2nd Edition* in the search box.

  In addition to these updates, check out the blog posts with answers to reader questions and demonstrations of useful book-related techniques at `http://blog.johnmuellerbooks.com/`.

- **Companion files:** Hey! Who really wants to type all the code in the book and reconstruct all those plots manually? Most readers prefer to spend their time actually working with Python, performing tasks using algorithms, and seeing the interesting things they can do, rather than typing. Fortunately for you, the examples used in the book are available for download, so all you need to do is read the book to learn algorithm usage techniques. You can find these files by searching *Algorithms For Dummies,* 2nd Edition at `www.dummies.com` and scrolling down the left side of the page that opens. The source code is also at `http://www.johnmuellerbooks.com/source-code/`, and `https://github.com/lmassaron/algo4d_2ed`.

Where to Go from Here

It's time to start your algorithm learning adventure! If you're completely new to algorithms, you should start with Chapter 1 and progress through the book at a pace that allows you to absorb as much of the material as possible. Make sure to read about Python, because the book uses this language as needed for the examples.

If you're a novice who's in an absolute rush to get going with algorithms as quickly as possible, you can skip to Chapter 3 with the understanding that you may find some topics a bit confusing later.

Readers who have some exposure to Python, and have the appropriate language versions installed, can save reading time by moving directly to Chapter 5. You can always go back to earlier chapters as necessary when you have questions. However, you do need to understand how each technique works before moving to the next one. Every technique, coding example, and procedure has important lessons for you, and you could miss vital content if you start skipping too much information.

1

Getting Started with Algorithms

IN THIS PART . . .

Defining algorithms and their design

Using Google Colab to work with algorithms

Performing essential data manipulations

Building a matrix manipulation class

IN THIS CHAPTER

- » Defining what is meant by *algorithm*
- » Relying on computers to use algorithms to provide solutions
- » Determining how issues differ from solutions
- » Performing data manipulation so that you can find a solution

Chapter 1 Introducing Algorithms

If you're in the majority of people, you're likely confused as you open this book and begin your adventure with algorithms, because most texts never tell you what an algorithm is, much less why you'd want to use one. Hearing about algorithms is like being in school again with the teacher droning on; you're falling asleep from lack of interest because algorithms don't seem particularly useful to understand at the moment.

The first section of this chapter is dedicated to helping you understand precisely what the term *algorithm* means and why you benefit from knowing how to use algorithms. Far from being arcane, algorithms are actually used all over the place, and you have probably used or been helped by them for years without really knowing it. So, they're stealth knowledge! In truth, algorithms are becoming the spine that supports and regulates what is important in an increasingly complex and technological society like ours.

The second section of this chapter discusses how you use computers to create solutions to problems using algorithms, how to distinguish between issues and solutions, and what you need to do to manipulate data to discover a solution. The goal is to help you differentiate between algorithms and other tasks that people confuse with algorithms. In short, you discover why you really want to know about algorithms, as well as how to apply them to data.

The third section of the chapter discusses algorithms in a real-world manner, that is, by viewing the terminologies used to understand algorithms and to present algorithms in a way that shows that the real world is often less than perfect. Understanding how to describe an algorithm in a realistic manner also helps to temper expectations to reflect the realities of what an algorithm can actually do.

The final section of the chapter discusses data. The algorithms you work with in this book require data input in a specific form, which sometimes means changing the data to match the algorithm's requirements. Data manipulation doesn't change the content of the data. Instead, it changes the presentation and form of the data so that an algorithm can help you see new patterns that weren't apparent before (but were actually present in the data all along).

Describing Algorithms

Even though people have solved algorithms manually for thousands of years, doing so can consume huge amounts of time and require many numeric computations, depending on the complexity of the problem you want to solve. Algorithms are all about finding solutions, and the speedier and easier, the better. A huge gap exists between mathematical algorithms historically created by geniuses of their time, such as Euclid (`https://www.britannica.com/biography/Euclid-Greek-mathematician`), Sir Isaac Newton (`https://www.britannica.com/biography/Isaac-Newton`), or Carl Friedrich Gauss (`https://www.britannica.com/biography/Carl-Friedrich-Gauss`), and modern algorithms created in universities as well as private research and development laboratories. The main reason for this gap is the use of computers. Using computers to solve problems by employing the appropriate algorithm speeds up the task significantly. You may notice that more problem solutions appear quickly today, in part, because computer power is both cheap and constantly increasing.

When working with algorithms, you consider the inputs, desired outputs, and the process (a sequence of actions) used to obtain a desired output from a given input. However, you can get the terminology wrong and view algorithms in the wrong way because you haven't really considered how they work in a real-world setting.

Sources of information about algorithms often present them in a way that proves confusing because they're too sophisticated or even downright incorrect. Although you may find other definitions, this book uses the following definitions for terms that people often confuse with algorithms (but aren't):

- **Equation:** Numbers and symbols that, when taken as a whole, equate to a specific value. An equation always contains an equals sign so that you know that the numbers and symbols represent the specific value on the other side of the equals sign. Equations generally contain variable information presented as a symbol, but they're not required to use variables.
- **Formula:** A combination of numbers and symbols used to express information or ideas. Formulas normally present mathematical or logical concepts, such as defining the Greatest Common Divisor (GCD) of two integers (the video at `https://www.khanacademy.org/math/cc-sixth-grade-math/cc-6th-factors-and-multiples/cc-6th-gcf/v/greatest-common-divisor` tells how this works). Generally, they show the relationship between two or more variables.
- **Algorithm:** A sequence of steps used to solve a problem. The sequence presents a unique method of addressing an issue by providing a particular solution. An algorithm need not represent mathematical or logical concepts, even though the presentations in this book often do fall into those categories because people most commonly use algorithms in this manner. In order for a process to represent an algorithm, it must be:

REMEMBER

  - **Finite:** The algorithm must eventually solve the problem. This book discusses problems with a known solution so that you can evaluate whether an algorithm solves the problem correctly.
  - **Well-defined:** The series of steps must be precise and present steps that are understandable. Especially because computers are involved in algorithm use, the computer must be able to understand the steps to create a usable algorithm.
  - **Effective:** An algorithm must solve all cases of the problem for which someone defined it. An algorithm should always solve the problem it has to solve. Even though you should anticipate some failures, the incidence of failure is rare and occurs only in situations that are acceptable for the intended algorithm use.

With these definitions in mind, the following sections help to clarify the precise nature of algorithms. The goal isn't to provide a precise definition for algorithms, but rather to help you understand how algorithms fit into the grand scheme of things so that you can develop your own understanding of what algorithms are and why they're so important.

The right way to make toast: Defining algorithm uses

An algorithm always presents a series of steps and doesn't necessarily perform these steps to solve a math formula. The scope of algorithms is incredibly large. You can find algorithms that solve problems in science, medicine, finance, industrial production and supply, and communication. Algorithms provide support for all parts of a person's daily life. Anytime a sequence of actions achieving something in our life is finite, well-defined, and effective, you can view it as an algorithm. For example, you can turn even something as trivial and simple as making toast into an algorithm. In fact, the making toast procedure often appears in computer science classes, as discussed at `http://brianaspinall.com/now-thats-how-you-make-toast-using-computer-algorithms/`.

Unfortunately, the algorithm on the site is flawed. The instructor never removes the bread from the wrapper and never plugs the toaster in, so the result is damaged plain bread still in its wrapper stuffed into a nonfunctional toaster (see the discussion at `http://blog.johnmuellerbooks.com/2013/03/04/procedures-in-technical-writing/` for details). Even so, the idea is the correct one, yet it requires some slight, but essential, adjustments to make the algorithm finite and effective.

One of the most common uses of algorithms is as a means of solving formulas. For example, when working with the GCD of two integer values, you can perform the task manually by listing each of the factors for the two integers and then selecting the greatest factor that is common to both. For example, GCD (20, 25) is 5 because 5 is the largest number that divides evenly into both 20 and 25. However, processing every GCD manually is time consuming and error prone, so the Greek mathematician Euclid created a better algorithm to perform the task. You can see the Euclidean method demonstrated at `https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm`.

However, a single formula, which is a presentation of symbols and numbers used to express information or ideas, can have multiple solutions, each of which is an algorithm. In the case of GCD, another common algorithm is one created by Derrick Henry Lehmer (`https://www.imsc.res.in/~kapil/crypto/notes/node11.html`). Because you can solve any formula multiple ways, people spend a great deal of time comparing algorithms to determine which one works best in a given situation. (See a comparison of Euclid to Lehmer at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.693&rep=rep1&type=pdf`.)

Because our society and its accompanying technology are changing quickly, we need algorithms that can keep the pace. Scientific achievements such as sequencing the human genome were possible in our age because scientists found algorithms that run fast enough to complete the task. Measuring which algorithm is better in a given situation, or in an average usage situation, is really serious stuff and is a topic of discussion among computer scientists.

When it comes to computer science, the same algorithm can have multiple presentations; why do it one way when you can invent multiple methods just for fun? For example, you can *present* the Euclidean algorithm in both recursive and iterative forms, as explained at `http://cs.stackexchange.com/questions/1447/what-is-most-efficient-for-gcd`. In short, algorithms present a method of solving formulas, but it would be a mistake to say that just one acceptable algorithm exists for any given formula or that only one acceptable presentation of an algorithm exists. Using algorithms to solve problems of various sorts has a long history — it isn't something that has just happened.

Even if you limit your gaze to computer science, data science, artificial intelligence, and other technical areas, you find many kinds of algorithms — too many for a single book. For example, *The Art of Computer Programming*, by Donald E. Knuth (Addison-Wesley), spans 3,168 pages in four volumes (see `http://www.amazon.com/exec/obidos/ASIN/0321751043/datacservip0f-20/`) and still doesn't manage to cover the topic (the author intended to write more volumes). However, here are some interesting uses for you to consider:

» **Searching:** Locating information or verifying that the information you see is the information you want is an essential task. Without this ability, you couldn't perform many tasks online, such as finding the website on the Internet selling the perfect coffee pot for your office. These algorithms change constantly, as shown by Google's recent change in its algorithm (`https://www.youaretech.com/blog/2021/1/26/webpage-experience-a-major-google-algorithm-update-in-2021nbsp`).

» **Sorting:** Determining which order to use to present information is important because most people today suffer from information overload, and putting information in order is one way to reduce the onrush of data. Imagine going to Amazon, finding that more than a thousand coffee pots are for sale there, and yet not being able to sort them in order of price or the most positive review. Moreover, many complex algorithms require data in the proper order to work dependably, so ordering is an important requisite for solving more problems.

» **Transforming:** Converting one sort of data to another sort of data is critical to understanding and using the data effectively. For example, you might understand imperial weights just fine, but all your sources use the metric system. Converting between the two systems helps you understand the data.

- **Scheduling:** Making the use of resources fair to all concerned is another way in which algorithms make their presence known in a big way. For example, timing lights at intersections are no longer simple devices that count down the seconds between light changes. Modern devices consider all sorts of issues, such as the time of day, weather conditions, and flow of traffic.
- **Graph analysis:** Deciding on the shortest path between two points finds all sorts of uses. For example, in a routing problem, your GPS couldn't function without this particular algorithm because it could never direct you along city streets using the shortest route from point A to point B. And even then, your GPS might direct you to drive into a lake (`https://theweek.com/articles/464674/8-drivers-who-blindly-followed-gps-into-disaster`).
- **Cryptography:** Keeping data safe is an ongoing battle with hackers constantly attacking data sources. Algorithms make it possible to analyze data, put it into some other form, and then return it to its original form later.
- **Pseudorandom number generation:** Imagine playing games that never varied. You start at the same place; perform the same steps, in the same manner, every time you play. Without the capability to generate seemingly random numbers, many computer tasks become impossible.

REMEMBER

This list presents an incredibly short overview. People use algorithms for many different tasks and in many different ways, and constantly create new algorithms to solve both existing problems and new problems. The most important issue to consider when working with algorithms is that given a particular input, you should expect a specific output. Secondary issues include how many resources the algorithm requires to perform its task and how long it takes to complete the task. Depending on the kind of issue and the sort of algorithm used, you may also need to consider issues of accuracy and consistency.

Finding algorithms everywhere

The previous section mentions the toast algorithm for a specific reason. For some reason, making toast is probably the most popular algorithm ever created. Many grade-school children write their equivalent of the toast algorithm long before they can even solve the most basic math. It's not hard to imagine how many variations of the toast algorithm exist and what the precise output is of each of them. The results likely vary by individual and the level of creativity employed. There are also websites dedicated to telling children about algorithms, such as the one at `https://www.idtech.com/blog/algorithms-for-kids`. In short, algorithms appear in great variety and often in unexpected places.

Every task you perform on a computer involves algorithms. Some algorithms appear as part of the computer hardware. The very act of booting a computer involves the use of an algorithm. You also find algorithms in operating systems, applications, and every other piece of software. Even users rely on algorithms. Scripts help direct users to perform tasks in a specific way, but those same steps could appear as written instructions or as part of an organizational policy statement.

Daily routines often devolve into algorithms. Think about how you spend your day. If you're like most people, you perform essentially the same tasks every day in the same order, making your day an algorithm that solves the problem of how to live successfully while expending the least amount of energy possible. After all, that's what a routine does; it makes us efficient.

Throughout this book, you see the same three elements for every algorithm:

1. Describe the problem.
2. Create a series of steps to solve the problem (well defined).
3. Perform the steps to obtain a desired result (finite and effective).

Using Computers to Solve Problems

The term *computer* sounds quite technical and possibly a bit overwhelming to some people, but people today are neck deep (possibly even deeper) in computers. You wear at least one computer, your smartphone, most of the time. If you have any sort of special device, such as a pacemaker, it also includes a computer. A car can contain as many as 150 computers in the form of embedded microprocessors that regulate fuel consumption, engine combustion, transmission, steering, and stability (see `https://spectrum.ieee.org/software-eating-car` for details), provide Advanced Driver-Assist Systems (ADAS), and more lines of code than a jet fighter. A computer exists to solve problems quickly and with less effort than solving them manually. Consequently, it shouldn't surprise you that this book uses still more computers to help you understand algorithms better.

Computers vary in a number of ways. The computer in a watch is quite small; the one on a desktop quite large. Supercomputers are immense and contain many smaller computers all tasked to work together to solve complex issues, such as predicting tomorrow's weather. The most complex algorithms rely on special computer functionality to obtain solutions to the issues people design them to solve. Yes, you could use lesser resources to perform the task, but the trade-off is waiting a lot longer for an answer, or getting an answer that lacks sufficient accuracy to provide a useful solution. In some cases, you wait so long that the answer is no longer important. With the need for both speed and accuracy in mind, the following sections discuss some special computer features that can affect algorithms.

Getting the most out of modern CPUs and GPUs

General-purpose processors, CPUs, started out as a means to solve problems using algorithms. However, their general-purpose nature also means that a CPU can perform a great many other tasks, such as moving data around or interacting with external devices. A general-purpose processor does many things well, which means that it can perform the steps required to complete an algorithm, but not necessarily fast. Owners of early general-purpose processors could add math coprocessors (special math-specific chips) to their systems to gain a speed advantage (see `https://www.computerhope.com/jargon/m/mathcopr.htm` for details). Today, general-purpose processors have the math coprocessor embedded into them, so when you get an Intel i9 processor, you actually get multiple processors in a single package.

A *GPU* is a special-purpose processor with capabilities that lend themselves to faster algorithm execution. For most people, GPUs are supposed to take data, manipulate it in a special way, and then display a pretty picture onscreen. However, any computer hardware can serve more than one purpose. It turns out that GPUs are particularly adept at performing data transformations, which is a key task for solving algorithms in many cases. It shouldn't surprise you to discover that people who create algorithms spend a lot of time thinking outside the box, which means that they often see methods of solving issues in nontraditional approaches.

The point is that CPUs and GPUs form the most commonly used chips for performing algorithm-related tasks. The first performs general-purpose tasks quite well, and the second specializes in providing support for math-intensive tasks, especially those that involve data transformations. Using multiple cores makes parallel processing (performing more than one algorithmic step at a time) possible. Adding multiple chips increases the number of cores available. Having more cores adds speed, but a number of factors keeps the speed gain to a minimum. Using two i9 chips won't produce double the speed of just one i9 chip.