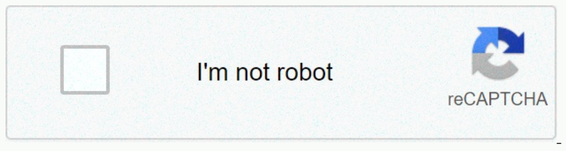
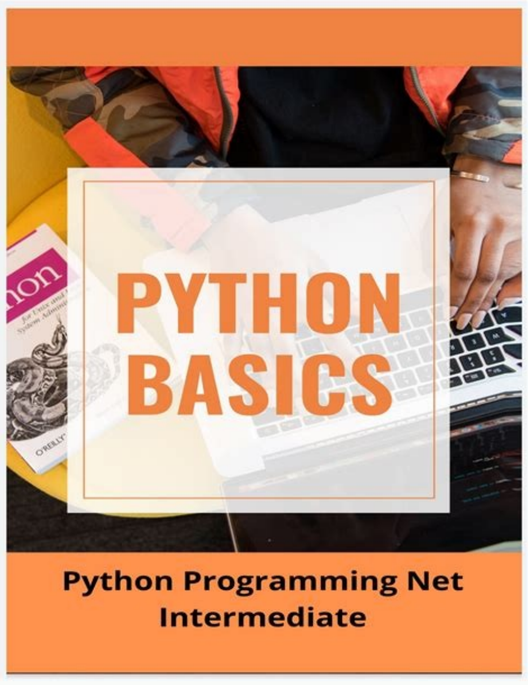


Continue

Python programming pdf free download

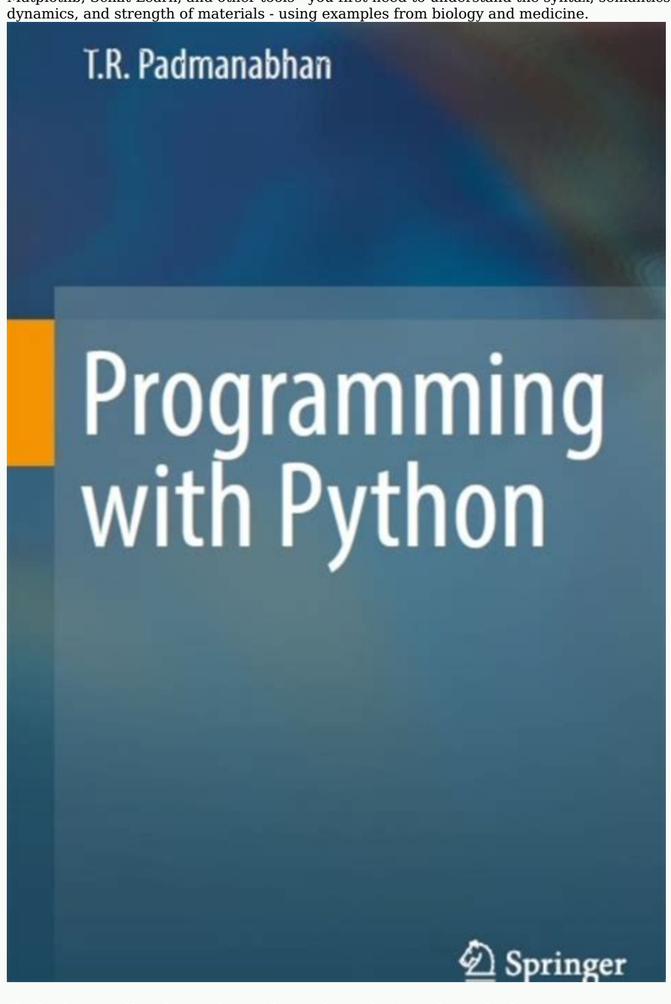
Python programming reema thareja pdf free download.

Python programming for beginners pdf free download. R nageswara rao core python programming pdf free download. A beginners guide to python programming pdf free download. Python programming questions and answers pdf free download. Python programming a modular approach pdf free download.



Python programming by sana rasheed download pdf free. Best book for python programming pdf free download. Python programming book pdf bangla free download. Python programming book in urdu pdf free download. Python programming in context 3rd edition pdf free download. Python programming a modern approach vamsi kurama pearson pdf free download. Python programming book in hindi pdf free download. Python programming notes pdf free download. Elements of programming interviews in python pdf free download.

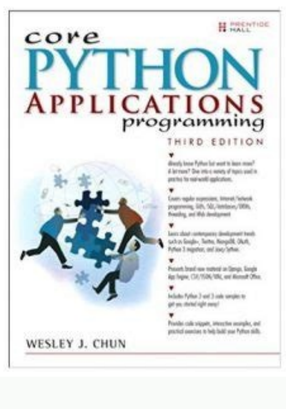
Free download Read online DescriptionTable of ContentsDetailsHashtagsReport an issue Guido van Rossum created the Python programming language in the late 1980s. In contrast to other popular languages such as C, C++ , Java, and C#, Python strives to provide a simple but powerful syntax.Python is used for software development at companies and organizations such as Google, Yahoo, Facebook, CERN, Industrial Light and Magic, and NASA. Experienced programmers can accomplish great things with Python, but Python's beauty is that it is accessible to beginning programmers and allows them to tackle interesting problems more quickly than many other, more complex languages that have a steeper learning curve.In late 2008, Python 3.0 was released. Commonly called Python 3, the current version of Python is incompatible with earlier versions of the language. Currently the Python world still is in transition between Python 2 and Python 3. Many existing published books cover Python 2, but more Python 3 resources now are becoming widely available.
The code in this book is based on Python 3.This book does not attempt to cover all the facets of the Python programming language. Experienced programmers should look elsewhere for books that cover Python in much more detail. The focus here is on introducing programming techniques and developing good habits. To that end, our approach avoids some of the more esoteric features of Python and concentrates on the programming basics that transfer directly to other imperative programming languages such as Java, C#, and C++ . We stick with the basics and explore more advanced features of Python only when necessary to handle the problem at hand.This open book is licensed under a Creative Commons License (CC BY). You can download Fundamentals of Python Programming ebook for free in PDF format (13.1 MB). Fundamentals of Computer Programming with C#This open book aims to provide novice programmers solid foundation of basic knowledge regardless of the programming language. This book covers the fundamentals of programming that have not changed significantly over the last 10 years. Educational content was developed by an authoritative author team led by Svetlin Nakov from the Software University...Fundamentals of C++ ProgrammingBjarne Stroustrup of AT&T Bell Labs created C++ in the mid 1980s. C++ is an extension of the programming language C, a product of AT&T Bell Labs from the early 1970s. C was developed to write the Unix operating system, and C is widely used for systems-level software and embedded systems development. C++ initially provided object-oriented...A Whirlwind Tour of PythonTo tap into the power of Python's open data science stack - including NumPy, Pandas, Matplotlib, Scikit-Learn, and other tools - you first need to understand the syntax, semantics, and patterns of the Python language. This report provides a brief yet comprehensive introduction to Python for engineers, researchers, and data scientists who are a...Fundamentals of BiomechanicsThis book integrates the classic fields of mechanics - statics, dynamics, and strength of materials - using examples from biology and medicine.



The book is excellent for teaching either undergraduates in biomedical engineering programs or health care professionals studying biomechanics at the graduate level. Extensively revised from a successful th...Python is one of those rare languages which can claim to be both simple and powerful. You will find yourself pleasantly surprised to see how easy it is to concentrate on the solution to the problem rather than the syntax and structure of the language you are programming in. A Byte of Python is a free book on programming using the Python language...Python Machine Learning ProjectsAs machine learning is increasingly leveraged to find patterns, conduct analysis, and make decisions -

sometimes without final input from humans who may be impacted by these findings - it is crucial to invest in bringing more stakeholders into the fold. This book of Python projects in machine learning tries to do just that: to equip the developers ... Page 1 Python Programming: An Introduction to Computer Science John M. Zelle, Ph.D. Preliminary Second Edition Fall 2009 This document was typeset by the author with LATEX 2ϱ. Contents 1 Computers and Programs 1 1.1 The Universal Machine . . . .

. . . . . .



. . .
. . . . . .
. . . . . 1 1.2 Program Power . . . .
. .
. . . . . .
.
. .
. . . .
. . . . . .
.
. . . . . 2 1.3 What is Computer Science? . . . . . .
. . . . . .
. . . . . .
.
. . . .
. 3 1.4 Hardware Basics . . . . .
. . . . . .
. . . . . .
. . . .
. . .
. . . 4 1.5 Programming Languages . . . . . .
. . . . . .
. 5 1.6 The Magic of Python .
. . . . . .
. . . . . .
. . . . .
. . .
. 7 1.7 Inside a Python Program . . . . . .
. . . .
. . . . . 12 1.8 Chaos and Computers . . . . . 14 1.9 Chapter Summary . . . . . .
. . . .
. . . . . .
. . . . .
. .
. . . 16 1.10 Exercises .
. . .
. . . . . .
. . . . . .
. . . . . .
. 17 2 Writing Simple Programs 21 2.1 The Software Development Process . . . . . .
. .
. . . . . .
. 21 2.2 Example Program: Temperature Converter . . . . . .
. . .
. . . . . 22 2.3 Elements of Programs . . . . . .
. . . . . 24 2.3.1 Names . . . . . .
. . . . . .
. . . . . .
. 24 2.3.2 Expressions . . . .
. . . . . .
. . . . . 25 2.4 Output Statements . . . . . .
.
. . . . . 27 2.5 Assignment Statements . . . . . .
. .
.
. . .
. . . 28 2.5.1 Simple Assignment . . .
. . . . .
.
. . . . . 28 2.5.2 Assigning Input . . . . . .
. . . .
. . . . . .
. . . . . 30 2.5.3 Simultaneous Assignment . . . .
.
. . .
. . . . . 32 2.6 Definite Loops . .
. . . . . .
. . .
. . .
. . . . . .
. . . . . 34 2.7 Example Program: Future Value .
. . . . . .
. . . . . .
.
. . . . . .
. 37 2.8 Chapter Summary . . . . . .
. .
. . . . . .
. . . . . .
. . . 39 2.9 Exercises . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . 40 3 Computing with Numbers 45 3.1 Numeric Data Types . .
.
. . . . . .
. . . . . .
. . . . . 45 3.2 Using the Math Library . .
. . . . . .
. . . . . .
. . . . . 49 3.3 Accumulating Results: Factorial . . . . . .
. . . . . .
. . . 51 i ii Contents 3.4 Limitations of Computer Arithmetic . . . . . .
. .
. . . . . .
. . . 54 3.5 Type Conversions and Rounding . .
. . . . . .
. . . .
.
. .
. . . . . 57 3.6 Chapter Summary . . . . . .
. . . . . 58 3.7 Exercises .
. . . . . .
. . . . . 59 4 Objects and Graphics 65 4.1 Overview . . . .
. . . . . .
. . . . . .
.
. .
. . . 65 4.2 The Object of Objects . . . . . .
. . . .
. . .
. . . . . .
. . 66 4.3 Simple Graphics Programming . . . . . .
. . . . . .
.
. . . . . 67 4.4 Using Graphical Objects .
. .
. . . . . .
. . . . . .
. . . 71 4.5 Graphing Future Value . . .
. . . .
. . .
.
. . . . . .
. . . . . .
75 4.6 Choosing Coordinates . . . . . .
. .
. .
. . . . . .
. . . . 81 4.7 Interactive Graphics . . . . . .
. . .
. . . . . 84 4.7.1 Getting Mouse Clicks . . . . . .
. . . . . .
.
. . . .
. . . . .
. . .
. . 84 4.7.2 Handling Textual Input . . . . . .
. . . . . .
. . . . . 86 4.8 Graphics Module Reference . . . . .
. . .
. . . .
. . . . . .
. . . . . .
. .
88 4.8.1 GraphWin Objects . . . . . .
. . . . .
. . . . . 89 4.8.2 Graphics Objects . . .
. . . . . .
. . . . . .
. . . . . 89 4.8.3 Entry Objects . . . . .
. . . . .
.
. . . . . .

. . . .
. . . . . 325 12.2.1 Candidate Objects and Methods . . . . . .
.
. . . .
. . .
. . . . . .
. . . . 325 12.2.2 Implementing SimStats . . . . . .
.
. . . . . .
. . . . . 327 12.2.3 Implementing RBallGame . . . . . .
. . . . . .
.
. . 328 12.2.4 Implementing Player .
. . . . . .
.
. . . . . .
331 12.2.5 The Complete Program . . . . . .
. . . . . .
. . . . . .
.
.
. . . . . 331 12.3 Case Study: Dice Poker . . . . . .
. . . . . .
. . . .
.
. . . . . 335 12.3.1 Program Specification . . . . . .
.
. . . . . .
. . . . . 335 12.3.2 Identifying Candidate Objects . . .
.
. . .
. . . .
. . . .
. . . . . 336 12.3.3 Implementing the Model . . . . . .
. . . .
. . 337 12.3.4 A Text-Based UI . . . . .
.
. . . . . .
.
. . . . . .
. .
. . . . . 341 12.3.5 Developing a GUI . . . . . .
. . . . . 343 vi Contents 12.4 OO Concepts . . . . . .
.
. .
. . . .
. . .
.
. . . . .
. . . . . .
. . . . . .
. 350 12.4.1 Encapsulation .
. . . . . .
.
. . . .
. . . . . .
. . 350 12.4.2 Polymorphism . .
. . .
. . . . .
. . . . . .
. . . . . .
. .
. . . . . .
. . . . . .
351 12.4.3 Inheritance . . . . . .
.
. . . . . .
. . . . . .
. . .
. . . . . .
. . . . . 351 12.5 Chapter Summary . . . . . .
. . . . . .
. . . 353 12.6 Exercises . . . . . .
.
. .
. . . . . .
. . .
.
. . . . . .
. . . . . 353 13 Algorithm Design and Recursion 357 13.1 Searching . . . . . .
. . . . . .
. . . . . .
. . . . . 357 13.1.1 A Simple Searching Problem . . . . . .
. . 358 13.1.2 Strategy 1: Linear Search . . . . . .
. . . . .
. . . . . .
.
. 359 13.1.3 Strategy 2: Binary Search . .
. . . . . .
. . . . . 359 13.1.4 Comparing Algorithms . . . . . .
. . . . . 360 13.2 Recursive Problem-Solving . . . .
.
. . . . . .
. . . .
. . . . .
. . . . . 362 13.2.1 Recursive Definitions . . . . . 363 13.2.2 Recursive Functions . . . . . .
. . . . . 364 13.2.3 Example: String Reversal . . . . . .
.
.
. . . . . .
. . . .
. . . . . 365 13.2.4 Example: Anagrams . . . . . .
. . 366 13.2.5 Example: Fast Exponentiation . . . . . .
. . .
. . . .
. . . . . .
. . .
. . . 367 13.2.6 Example: Binary Search . . . . . .
.
. . . . . 368 13.2.7 Recursion vs. Iteration . . . . . .
. . . . . 369 13.3 Sorting Algorithms . .
.
. . . . . 371 13.3.1 Naive Sorting: Selection Sort . . . . . .
. . . . . 371 13.3.2 Divide and Conquer: Merge Sort . . . . . .
. . . . . .
. . . . . 373 13.3.3 Comparing Sorts . . . . . .
. .
. . . .
. .
. . . . . .
. . . . . 375 13.4 Hard Problems .
. . .
. .
. . . . . 377 13.4.1 Towers of Hanoi . . . . . .
. . .
. . .
. . . . . 378 13.4.2 The Halting Problem . . . . . .
. . . . . .
. . . . . 382 13.4.3 Conclusion . . . . . .
. . . . . 384 13.5 Chapter Summary . . .
. .
. . . . . .
. . .
. . .
. . . . . 384 13.6 Exercises . . . . . .
. . . . . .
.
. . . . . 385 Chapter 1 Computers and Programs Objectives • To understand the respective roles of hardware and software in a computing system. • To learn what computer scientists study and the techniques that they use. • To understand the basic design of a modern computer. • To understand the form and function of computer programming languages. • To begin using the Python programming language.
• To learn about chaotic models and their implications for computing. 1.1 The Universal Machine Almost everyone has used a computer at one time or another. Perhaps you have played computer games or used a computer to write a paper or balance your checkbook. Computers are used to predict the weather, design airplanes, make movies, run businesses, perform financial transactions, and control factories. Have you ever stopped to wonder what exactly a computer is? How can one device perform so many different tasks? These basic questions are the starting point for learning about computers and computer programming. A modern computer can be defined as “a machine that stores and manipulates information under the control of a changeable program.” There are two key elements to this definition.
The first is that computers are devices for manipulating information. This means we can put information into a computer, and it can transform the information into new, useful forms, and then output or display the information for our interpretation. Computers are not the only machines that manipulate information. When you use a simple calculator to add up a column of numbers, you are entering information (the numbers) and the 1 2 Chapter 1. Computers and Programs calculator is processing the information to compute a running sum which is then displayed. Another simple example is a gas pump. As you fill your tank, the pump uses certain inputs: the current price of gas per gallon and signals from a sensor that reads the rate of gas flowing into your car. The pump transforms this input into information about how much gas you took and how much money you owe. We would not consider either the calculator or the gas pump as full-fledged computers, although modern versions of these devices may actually contain embedded computers. They are different from computers in that they are built to perform a single, specific task. This is where the second part of our definition comes into the picture: Computers operate under the control of a changeable program. What exactly does this mean? A computer program is a detailed, step-by-step set of instructions telling a computer exactly what to do. If we change the program, then the computer performs a different sequence of actions, and hence, performs a different task. It is this flexibility that allows your PC to be at one moment a word processor, at the next moment a financial planner, and later on, an arcade game. The machine stays the same, but the program controlling the machine changes. Every computer is just a machine for executing (carrying out) programs. There are many dif- ferent kinds of computers. You might be familiar with Macintoshes and PCs, but there are literally thousands of other kinds of computers both real and theoretical. One of the remarkable discoveries of computer science is the realization that all of these different computers have the same power; with suitable programming, each computer can basically do all the things that any other computer can do. In this sense, the PC that you might have sitting on your desk is really a universal machine. It can do anything you want it to do, provided you can describe the task to be accomplished in sufficient detail. Now that’s a powerful machine! 1.2 Program Power You have already learned an important lesson of computing: Software (programs) rules the hard- ware (the physical machine). It is the software that determines what any computer can do. Without software, computers would just be expensive paperweights. The process of creating software is called programming, and that is the main focus of this book. Computer programming is a challenging activity. Good programming requires an ability to see the big picture while paying attention to minute detail. Not everyone has the talent to become a first-class programmer, just as not everyone has the skills to be a professional athlete. However, virtually anyone can learn how to program computers. With some patience and effort on your part, this book will help you to become a programmer. There are lots of good reasons to learn programming. Programming is a fundamental part of computer science and is, therefore, important to anyone interested in becoming a computer profes- sional. But others can also benefit from the experience. Computers have become a commonplace tool in our society. Understanding the strengths and limitations of this tool requires an understand- ing of programming. Non-programmers often feel they are slaves of their computers. Programmers, however, are truly in control.
If you want to become a more intelligent user of computers, then this book is for you. 1.3. What is Computer Science? 3 Programming can also be loads of fun. It is an intellectually engaging activity that allows people to express themselves through useful and sometimes remarkably beautiful creations. Believe it or not, many people actually write computer programs as a hobby. Programming also develops valuable problem-solving skills, especially the ability to analyze complex systems by reducing them to interactions of understandable subsystems. As you probably know, programmers are in great demand.
More than a few liberal arts majors have turned a couple of computer programming classes into a lucrative career option. Comput- ers are so commonplace in the business world today that the ability to understand and program computers might just give you the edge over your competition, regardless of your occupation. 1.3 What is Computer Science? You might be surprised to learn that computer science is not the study of computers. A famous computer scientist named Edsger Dijkstra once quipped that computers are to computer science what telescopes are to astronomy. The computer is an important tool in computer science, but it is not itself the object of study. Since a computer can carry out any process that we can describe, the real question is What processes can we describe? Put another way, the fundamental question of computer science is simply What can be computed? Computer scientists use numerous techniques of investigation to answer this question. The three main ones are design, analysis, and experimentation. One way to demonstrate that a particular problem can be solved is to actually design a solution. That is, we develop a step-by-step process for achieving the desired result. Computer scientists call this an algorithm. That’s a fancy word that basically means “recipe.” The design of algorithms is one of the most important facets of computer science.
In this book you will find techniques for designing and implementing algorithms. One weakness of design is that it can only answer the question What is computable? in the positive. If I can devise an algorithm, then the problem is solvable. However, failing to find an algorithm does not mean that a problem is unsolvable. It may mean that I’m just not smart enough, or I haven’t hit upon the right idea yet. This is where analysis comes in. Analysis is the process of examining algorithms and problems mathematically. Computer scien- tists have shown that some seemingly simple problems are not solvable by any algorithm. Other problems are intractable. The algorithms that solve these problems take too long or require too much memory to be of practical value. Analysis of algorithms is an important part of computer science; throughout this book we will touch on some of the fundamental principles. Chapter 13 has examples of unsolvable and intractable problems. Some problems are too complex or ill-defined to lend themselves to analysis. In such cases, computer scientists rely on experimentation; they actually implement systems and then study the resulting behavior. Even when theoretical analysis is done, experimentation is often needed in order to verify and refine the analysis. For most problems, the bottom line is whether a working, reliable system can be built. Often we require empirical testing of the system to determine that this bottom-line has been met. As you begin writing your own programs, you will get plenty of opportunities to observe your solutions in action. I have defined computer science in terms of designing, analyzing, and evaluating algorithms, 4 Chapter 1.
Computers and Programs Input Devices CPU Secondary Memory Main Memory Output Devices Figure 1.1: Functional View of a Computer. and this is certainly the core of the academic discipline. These days, however, computer scien- tists are involved in far-flung activities, all of which fall under the general umbrella of computing. Some example areas include networking, human-computer interaction, artificial intelligence, com- putational science (using powerful computers to model scientific data), databases, software engi- neering, web and multimedia design, management information systems, and computer security. Wherever computing is done, the skills and knowledge of computer science are being applied. 1.4 Hardware Basics You don’t have to know all the details of how a computer works to be a successful programmer, but understanding the underlying principles will help you master the steps we go through to put our programs into action. It’s a bit like driving a car. Knowing a little about internal combustion engines helps to explain why you have to do things like fill the gas tank, start the engine, step on the accelerator, etc. You could learn to drive by just memorizing what to do, but a little more knowledge makes the whole process much more understandable. Let’s take a moment to “look under the hood” of your computer. Although different computers can vary significantly in specific details, at a higher level all mod- ern digital computers are remarkably similar. Figure 1.1 shows a functional view of a computer. The central processing unit (CPU) is the “brain” of the machine. This is where all the basic oper- ations of the computer are carried out. The CPU can perform simple arithmetic operations like adding two numbers and can also do logical operations like testing to see if two numbers are equal. The memory stores programs and data. The CPU can only directly access information that is stored in main memory (called RAM for Random Access Memory). Main memory is fast, but it is also volatile. That is, when the power is turned off, the information in the memory is lost. Thus, there must also be some secondary memory that provides more permanent storage. In a modern personal computer, this is usually some sort of magnetic medium such as a hard disk (also called a hard drive). Optical media such as CD (compact disc) and DVD (digital versatile disc) and flash memory devices such as USB memory “sticks” are also common. Humans interact with the computer through input and output devices. You are probably familiar with common devices such as a keyboard, mouse, and monitor (video screen). Information from 1.5. Programming Languages 5 input devices is processed by the CPU and may be shuffled off to the main or secondary memory. Similarly, when information needs to be displayed, the CPU sends it to one or more output devices. So what happens when you fire up your favorite game or word processing program? First, the instructions that comprise the program are copied from the (more) permanent secondary memory into the main memory of the computer. Once the instructions are loaded, the CPU starts executing the program. Technically the CPU follows a process called the fetch-execute cycle. The first instruction is retrieved from memory, decoded to figure out what it represents, and the appropriate action carried out. Then the next instruction is fetched, decoded and executed. The cycle continues, instruction after instruction. This is really all the computer does from the time that you turn it on until you turn it off again: fetch, decode, execute. It doesn’t seem very exciting, does it?
But the computer can execute this stream of simple instructions with blazing speed, zipping through millions of instructions each second.
Put enough simple instructions together in just the right way, and the computer does amazing things. 1.5 Programming Languages Remember that a program is just a sequence of instructions telling a computer what to do. Ob- viously, we need to provide those instructions in a language that a computer can understand. It would be nice if we could just tell a computer what to do using our native language, like they do in science fiction movies. (“Computer, how long will it take to reach planet Alphalpha at maximum warp?”) Unfortunately, despite the continuing efforts of many top-flight computer scientists (in- cluding your author), designing a computer to fully understand human language is still an unsolved problem. Even if computers could understand us, human languages are not very well suited for describing complex algorithms. Natural language is fraught with ambiguity and imprecision. For example, if I say: “I saw the man in the park with the telescope,” did I have the telescope, or did the man?
And who was in the park? We understand each other most of the time only because all humans share a vast store of common knowledge and experience. Even then, miscommunication is commonplace. Computer scientists have gotten around this problem by designing notations for expressing computations in an exact and unambiguous way. These special notations are called programming languages. Every structure in a programming language has a precise form (its syntax) and a precise meaning (its semantics). A programming language is something like a code for writing down the instructions that a computer will follow. In fact, programmers often refer to their programs as computer code, and the process of writing an algorithm in a programming language is called coding. Python is one example of a programming language. It is the language that we will use through- out this book.1 You may have heard of some other languages, such as C++, Java, Perl, Scheme, or BASIC.
Although these languages differ in many details, they all share the property of having well-defined, unambiguous syntax and semantics. Languages themselves tend to evolve over time. 1Specifically, the book was written using Python version 3.0. If you have an earlier version of Python installed on your computer, you should upgrade to the latest stable 3.x version to try out the examples. 6 Chapter 1. Computers and Programs (Program) Compiler Machine Code Running Inputs Outputs Source Code Program Figure 1.2: Compiling a High-Level Language All of the languages mentioned above are examples of high-level computer languages. Although they are precise, they are designed to be used and understood by humans. Strictly speaking, computer hardware can only understand a very low-level language known as machine language. Suppose we want the computer to add two numbers. The instructions that the CPU actually carries out might be something like this. load the number from memory location 2001 into the CPU load the number from memory location 2002 into the CPU add the two numbers in the CPU store the result into location 2003 This seems like a lot of work to add two numbers, doesn’t it? Actually, it’s even more complicated than this because the instructions and numbers are represented in binary notation (as sequences of 0s and 1s). In a high-level language like Python, the addition of two numbers can be expressed more natu- rally: c = a + b. That’s a lot easier for us to understand, but we need some way to translate the high-level language into the machine language that the computer can execute. There are two ways to do this: a high-level language can either be compiled or interpreted. A compiler is a complex computer program that takes another program written in a high-level language and translates it into an equivalent program in the machine language of some computer. Figure 1.2 shows a block diagram of the compiling process. The high-level program is called source code, and the resulting machine code is a program that the computer can directly execute. The dashed line in the diagram represents the execution of the machine code (aka “running the pro- gram”). An interpreter is a program that simulates a computer that understands a high-level language. Rather than translating the source program into a machine language equivalent, the interpreter analyzes and executes the source code instruction by instruction as necessary. Figure 1.3 illustrates the process. The difference between interpreting and compiling is that compiling is a one-shot translation; once a program is compiled, it may be run over and over again without further need for the compiler or the source code. In the interpreted case, the interpreter and the source are needed every time