Now that you're ready to become a software developer and have read the Introduction of this book, you need to become familiar with several key concepts. Your computer program will do exactly what you tell it to do—no more and no less. It will follow the programming rules that were defined by the operating system and programming language. Your program doesn't care if you are having a bad day or how many times you ask it to perform something. Often, what you think you've told your program to do and what it actually does are two different things.

**KEY TO SUCCESS:** If you haven’t already, take a few minutes to read the Introduction of this book. The Introduction shows you where to go to access the free webinars, forums, and YouTube videos that go with each chapter. Also, you’ll better understand why we are using the Alice programming environment and how to be successful in developing your iOS and Mac apps.

Depending on your background, working with something absolutely black and white may be frustrating. Many times, programming students have lamented, “That’s not what I wanted it to do!” As you begin to gain experience and confidence programming, you’ll begin to think like a programmer. You will understand software design and logic, and you will experience having your programs perform exactly as you want and the satisfaction associated with this.

**Thinking like a Developer**

Software development involves writing a computer program and then having a computer execute that program. A **computer program** is the set of instructions that we want the computer to perform. Before beginning to write a computer program, it is helpful to list the steps that we want our program to perform, in the order we want them accomplished. This step-by-step process is called an **algorithm**.
If we want to write a computer program to toast a piece of bread, we would first write an algorithm. This algorithm might look something like this:

1. Take the bread out of the bag.
2. Place the bread in the toaster.
3. Press the toast button.
4. Wait for the toast to pop up.
5. Remove the toast from the toaster.

At first glance, this algorithm seems to solve our problem. However, our algorithm leaves out many details and makes many assumptions. For example,

1. What kind of toast does the user want? Does the user want white bread, wheat, or some other kind of bread?
2. How does the user want the bread toasted? Light or dark?
3. What does the user want on the bread after it is toasted: butter, margarine, honey, or strawberry jam?
4. Does this algorithm work for all users in their cultures and languages? Some cultures may have another word for toast or not know what toast is.

Now, you might be thinking we are getting too detailed for just making a simple toast program. Over the years, software development has gained a reputation of taking too long, costing too much, and not being what the user wants. This reputation came to be because computer programmers often start writing their programs before they have really thought through their algorithms.

The key ingredients to making successful applications are design requirements. Design requirements can be very formal and detailed or as simple as a list on a piece of paper. Design requirements are important because they help the developer flush out what the application should do and not do when complete. Design requirements should not be completed in a programmer’s vacuum, but should be produced as the result of collaboration between developers, users, and customers.