science departments were experimenting with teaching C/C++. Finally, we are able to choose a language that is commonly taught and used in many contexts. Thus, it is likely that some of the students reading our text will already know Java and can contribute much to a class that uses our text.

Java provides many powerful libraries for building a graphical user interface and incorporating audio, video, and other media. If we were to discuss these libraries, students would become absorbed in programming tasks that have little or nothing to do with physics. For this reason our text uses the Open Source Physics library which makes it easy to write programs that are simpler and more graphically oriented than those that we wrote in True Basic. In addition, the Open Source Physics library is useful for other computational physics projects, which are not discussed in this text, as well as general programming tasks. This library provides for easy graphical input of parameters, tabular output of data, plots, visualizations and animations, and the numerical solution of ordinary differential equations. It also provides several useful data structures. The Open Source Physics library was developed by Wolfgang Christian, with the contributions and assistance of many others. The book *Open Source Physics: A User's Guide with Examples* by Wolfgang Christian is available separately and discusses the Open Source Physics library in much more detail. A CD that comes with the User's Guide contains the source code for the Open Source Physics library, the programs in this book, as well as ready-to-run versions of these programs. The source code and the library can also be downloaded freely from `<www.opensourcephysics.org/sip>`.

The ease of doing visualizations is a new and important aspect of Java and the Open Source Physics library, giving Java an advantage over other languages such as C++ and Fortran which do not have built-in graphics capabilities. For example, when debugging a program, it is frequently much quicker to detect when the program is not working by looking at a visual representation of the data rather than by scanning the data as lists of numbers. Also, it is easier to choose the appropriate values of the parameters by varying them and visualizing the results. Finally, more insight is likely to be gained by looking at a visualization than a list of numbers. Because animations and the continuous plotting of data usually cause a program to run more slowly, we have designed our programs so that the graphical output can be turned off or implemented infrequently during a simulation.

Java provides support for interacting with a program during runtime. The Open Source Physics library makes this interaction even easier, so that we can write programs that use a mouse to input data, such as the location of a charge, or toggle the value of a cell in a lattice. We also do not need to input how long a simulation should run and can stop the program at any time to change parameters.

As with our previous editions, we assume no background in computer programming. Much of the text can be understood by students with only a semester each of physics and calculus. Chapter 2 introduces Java and the Open Source Physics library. In Chapter 3 we discuss the concept of interfaces and how to use some of the important interfaces in the Open Source Physics library. Later chapters introduce more Java and Open Source Physics constructs as needed, but essentially all of the chapters after Chapter 3 can be studied independently and in any order. We include many topics that are sometimes considered too advanced for undergraduates, such as random walks, chaos, fractals, percolation, simulations of many particle systems, and topics in the theory of complexity, but we introduce these topics so that very little background is required. Other chapters discuss optics, electrodynamics, relativity, rigid body motion, and quantum mechanics, which require knowledge of the physics found in the corresponding standard undergraduate courses.

This text is written so that the physics drives the choice of algorithms and the programming syntax that we discuss. We believe that students can learn how to program more quickly with this approach because they have an immediate context, namely doing simulations, in which to hone their skills. In the beginning most of the programming tasks involve modifying the programs in the text. Students should then be given some assignments that require them to write their own programs by following the format of those in the text. The students may later develop their own style as they work on their projects.

Our text is most appropriately used in a project-oriented course that lets students with a wide variety of backgrounds and abilities work at their own pace. The courses that we have taught using this text have a laboratory component. From our experience we believe that active learning, where students are directly grappling with the material in this text, is the most efficient. In a laboratory context students who already know a programming language can help those who do not. Also, students can further contribute to a course by sharing their knowledge from various backgrounds in physics, chemistry, computer science, mathematics, biology, economics, and other subjects.

Although most of our text is at the undergraduate level, many of the topics are considered to be graduate level and thus would be of interest to graduate students. One of us regularly teaches a laboratory-based course on computer simulation with both undergraduate and graduate students. Because the course is project oriented, students can go at their own pace and work on different problems. In this context, graduate and undergraduate students can learn much from each other.

Some instructors who might consider using our text in a graduate-level context might think that our text is not sufficiently rigorous. For example, in the suggested problems we usually do not explicitly ask students to do an extensive data analysis. However, we do discuss how to estimate errors in Chapter 11. We encourage instructors to ask for a careful data analysis on at least one assignment, but we believe that it is more important for students to spend most of their time in an exploratory mode where the focus is on gaining physical insight and obtaining numerical results that are qualitatively correct.

There are four types of suggested student activities. The exercises, which are primarily found in the beginning of the text, are designed to help students learn specific programming techniques. The problems, which are scattered throughout each chapter, are open ended and require students to run, analyze, and modify programs given in the text or write new, but similar programs. Students will soon learn that the format for most of the programs is very similar. Starred problems require either significantly more background or work and may require the writing of a program from scratch. However, the programs for these problems still follow a similar format. The projects at the end of most of the chapters are usually more time consuming and would be appropriate for term projects or independent student research. Many new problems and projects have been added to this edition, while others have been improved and some have been eliminated. Instructors and students should view the problem descriptions and questions as starting points for thinking about the system of interest. It is important that students read the problems even if they do not plan to do them.

We encourage instructors to ask students to write laboratory reports for at least some of the problems. The Appendix to Chapter 1 provides guidance on what these reports should include. Part of the beauty and fun of doing computer simulations is that one is forced to think about the choice of algorithm, its implementation, the choice of parameters, what to measure, and the results. Do the results make sense? What happens if you change a parameter? What if you change the algorithm? Much physics can be learned in this way.