- 235U has a probability of decaying into two nuclei approximately half its size along with other particles.
- Radioactive decay is a random process; the exact timing of decay for a single nucleus cannot be predicted, only the probability or average time for decay can be estimated.
- It's important to emphasize that this numerical solution is an approximation and not the exact solution.
- The approach described, known as the Euler method, is a widely used algorithm for solving ordinary differential equations.
- While the Euler method is natural and commonly used, it's not the only algorithm available for such problems.
- Different methods for solving equations of this kind will be discussed in later chapters, along with detailed discussions on their strengths, weaknesses, and associated errors.
- One of the initial decisions in writing a program is the choice of programming language.
- While there are many programming languages suitable for the problems addressed in the book, it's impractical to provide example programs in all of them.
- Instead, the structure of a program is described in a general way using pseudocode.
- Pseudocode is not a precise programming language but rather a description of the essential steps in a program.
- It provides a high-level overview of the algorithm without getting bogged down in language-specific syntax.
- The use of pseudocode allows users of different programming languages to understand the algorithm's logic and adapt it to their preferred language.
- This approach facilitates communication and collaboration among programmers working with different languages.
- The chapter introduces the concept of expressing programming instructions in a generalized "conson" language, aiming to provide sufficient detail for readers to translate the instructions into their preferred programming language.
- While examples are provided only in pseudocode, the chapter also illustrates the translation process into two popular programming languages: Fortran and C.

- Programs for the problems discussed in the book are available in Fortran, C, and Python on the authors' website, facilitating practical implementation.
- The chapter emphasizes the importance of adopting recommended programming practices to ensure readability and understandability of code written by oneself and others.
- Before writing detailed code, it's essential to construct an outline of how the problem will be solved and identify necessary variables and parameters. Pseudocode often serves as a guideline for this.
- The structure of the program includes four basic steps: declaring necessary variables, initializing them, performing calculations, and storing results.
- Example 1.1 provides pseudocode for the main program portion of the radioactive decay problem, outlining the steps: declaring variables and arrays, initializing them, performing calculations, and storing results.
- The program begins with comment statements that provide information about the purpose of the program. In Fortran, comments are indicated by an exclamation mark (!) at the beginning of a line.
- The first line of the program "program decay" gives the name of the main program.
- Declarations such as "double precision Nu(100), t(100)" declare two arrays, "Nu" and "t", each capable of holding 100 values. These arrays will store the calculated values of the number of uranium nuclei and corresponding time values.
- The choice of arrays in the main program is because the subroutines that perform the calculations will interact with these arrays through the main program.
- The subroutine "initialize" sets the initial values of the variables.
- The subroutine "calculate" uses the Euler method to perform the computation.
- The program follows a modular structure with three subroutines: "initialize", "calculate", and "store", which are called sequentially.
- The names of the subroutines directly correspond to the general program outline mentioned earlier.
- Each subroutine includes statements describing the tasks it performs.
- The choice of subroutine names and array names aims to make the program easier to read and understand.
- This chapter illustrates examples in Fortran, but similar features are present in most programming languages.

- Future chapters will introduce more advanced programming concepts and may involve writing code in languages like Python and MATLAB.
- After debugging to resolve syntax issues, it's important to verify the accuracy of the program's results.
- Several guidelines are provided for verifying the correctness of program output.
- Firstly, assess whether the output seems reasonable by comparing it with your intuition and instincts about the problem. This helps improve overall understanding and ensures the results make sense.
- When sharing results with others, it's crucial to be able to convince them that the output is sensible.
- Additionally, if exact results are available for comparison, such as in the case of the radioactive decay problem where an analytic solution is known, compare the numerical values with the exact results.
- While exact comparisons may not always be feasible, physicists often aim to calculate results within an order of magnitude. This means anticipating answers with reasonable accuracy, even if exact values aren't available.
- Testing the program involves not only ensuring it runs without errors but also verifying that its output aligns with expectations and, when possible, with known exact solutions.Designing and selecting appropriate algorithms is a central concern in computer science related to numerical simulations.
- The book emphasizes practical applications rather than focusing solely on theoretical aspects.
- Despite the importance of numerical methods, the book prioritizes practical implementation over detailed theoretical discussions.
- The choice of algorithms, their numerical uncertainties, and their applicability are carefully considered.
- Numerical methods are discussed in relation to their relevance to the physics of the problem being studied.
- The appendix contains systematic discussions on various numerical methods covered in the book.
- Errors arise from finite numerical precision in programming languages, termed round-off errors, which are nearly always present.
- Modern computer languages typically employ a large number of significant digits to mitigate round-off errors.
- Double precision variables are recommended for minimizing numerical errors, although they may not always eliminate them entirely, particularly for certain types of problems sensitive to round-off errors.