Solve Problems With Computers

Everyday Programming

VOLUME I

Basics of Computer Programs

NOBEL KHANDAKER

Everyday Programming

Dr. Nobel Khandaker

Free Sample

Chapters 1 and 2

First Edition

Intramotev Press

PUBLISHED BY INTRAMOTEV PRESS

INTRAMOTEV.COM

This document contains Chapters 1 and 2 of *Everyday Programming*, made freely available as a sample of the complete book.

First printing, July 2026

Contents

Preface

Welcome, if you are a beginner wondering how to approach computer programming this book is for you.

about this sample. What follows is the opening of *Everyday Programming*: Chapter 1, *Mathematical Concepts*, and Chapter 2, *Problem Solving*. Together they make up Part I of the book, *Basic Concepts* — the two chapters that come before any Python is written.

That ordering is deliberate. Programming is the act of describing a solution precisely enough that a machine can carry it out, and both halves of that sentence need groundwork: a small vocabulary for describing things exactly (Chapter 1) and a habit of taking a problem apart before reaching for a keyboard (Chapter 2). The mathematics here goes no further than a tenth-grade classroom — logic, sets, remainders, variables, functions, sequences, counting, graphs, binary numbers — and every idea is introduced because a later chapter needs it, not for its own sake.

The complete book runs to twenty-one chapters and 445 *Find the Bug* exercises with full worked solutions. Cross-references in these pages that point past Chapter 2 refer to that larger work.

Part I

Basic Concepts

1

Mathematical Concepts

Computer programming does not require one to master complex mathematical concepts, nevertheless, an introduction to a few discrete mathematical concepts is useful. Discrete mathematics is the kind of math used for things you can count, list, organize, or decide. It is different from calculus, which studies continuous change. To start learning programming, you do not need advanced math like calculus. A strong understanding of logic, patterns, numbers, functions, and step-by-step thinking is much more important. Computer programming is built mostly on discrete ideas because computers work with separate values like `0` and `1`, `True` and `False`, `characters`, `lists`, and `steps`.

1.1 Logic

Logic is about deciding whether statements are `True` or `False`. In programming, this appears in if statements.

1.1.1 Examples

- ❑ `x > 5` is either `True` or `False`.
- ❑ `x > 5 and x < 10` is true only when both parts are true.
- ❑ `x == 3 or x == 4` is true when either part is true.

1.1.2 Common Logic Words

- ❑ and: both must be true
- ❑ or: at least one must be true
- ❑ not: reverses true/false

1.1.3 Example

```
age = 15
IF age >= 13 AND age <= 19:
    PRINT "Teenager"
```

1.2 Set

A set is a collection of different items. Sets help with searching, filtering, storing and retrieving data, tags, and removing duplicates.

1.2.1 Examples

Set of vowels: a, e, i, o, u Set of even numbers less than 10: 2, 4, 6, 8

1.2.2 Useful Set Ideas

- ❑ Union: items in either set
- ❑ Intersection: items in both sets
- ❑ Difference: items in one set but not the other

1.2.3 Example

```
A = {1, 2, 3}
B = {3, 4, 5}
Union: {1, 2, 3, 4, 5}
Intersection: {3}
```

1.3 Integers and Modular Arithmetic

Integers are the positive and negative counting numbers (and zero) `-2`, `-1`, `0`, `1`, `2`,. Modular arithmetic means "wrapping around" after reaching a certain number. Mod is used in clocks, game turns, repeating patterns, etc. The following example shows how the modulus can be used to determine if a number is odd or even.

```
number = 17
IF number MOD 2 == 0
    PRINT "Even"
ELSE
    PRINT "Odd"
```

1.3.1 Modulus Operation for a Clock

If it is `10` o'clock now, `5` hours later it is `3` o'clock. That is because `10 + 5 = 15`, and on a 12-hour clock, `15 mod 12 = 3`.

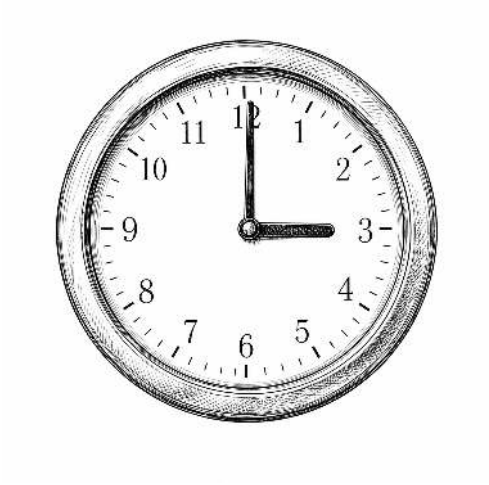

Figure 1.1: Wall Clock

1.4 Variables

A VARIABLE is a symbol or a letter that represents an unspecified object. Variables are used in mathematical equations and to capture abstract ideas. We will be using variables as inputs to the functions while discussing computer programs. Figure 1.2 shows the famous Pythagoras formula or function $a^2 + b^2 = c^2$ that can be used to calculate the length of a leg of a right triangle when we know the lengths of the other two sides. Here a and b are the legs, c is the hypotenuse, and all three are variables that can be assigned numerical values.

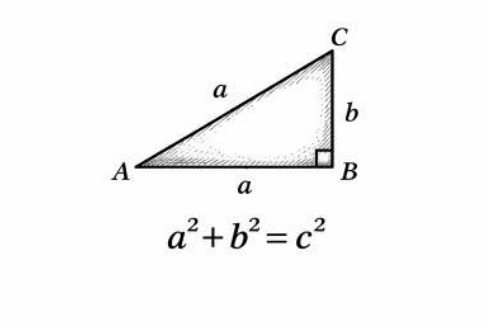

Figure 1.2: Pythagoras theorem

1.4.1 Functions

A FUNCTION f is defined as something that transforms or maps one set of elements X to another set of elements Y. Figure 1.3 shows a function as a mapping: $f : X \rightarrow Y$.

Notice that we can visualize a function as a process that takes input values, processes the input values, and produces output values (see figure 3.3 in chapter 3). For a complex process running inside a computer, the input could be a variety of things, numbers, letters, images, video, sound, etc. Although simplistic, we will build our understanding of computer programs and learn problem solving techniques using this easy to understand concept.

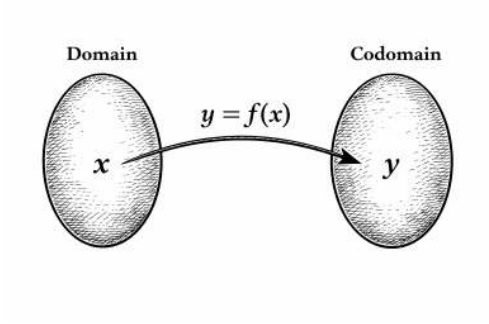

Figure 1.3: Function $y = f(x)$

Examples of functions:

- A straight line $y = mx + c$
- Root of a quadratic equation $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- Sine wave $y = \sin(x)$
- Temperature conversion $f(c) = c \times \frac{9}{5} + 32$, which maps a Celsius reading to its Fahrenheit value
- Absolute function

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

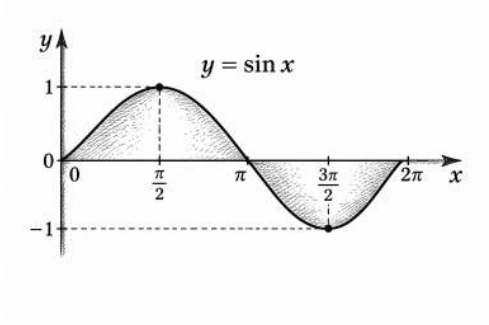

Figure 1.4: Sine wave

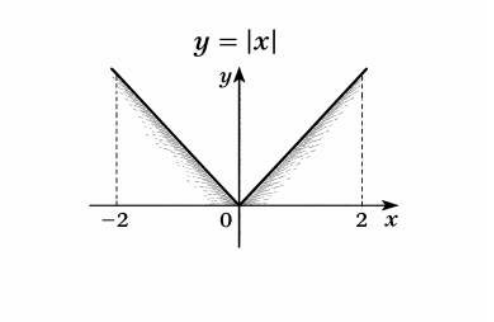

Figure 1.5: Absolute function

1.5 Composition

Function composition is the method of combining two functions so that the output of one function becomes the input of another. Think of it like a chain of machines: the first machine takes an input and produces an output, and then the second machine takes that output and processes it again. If we have two functions, $f(x)$ and $g(x)$, the composition is written as $f(g(x))$, which means you apply g first, then apply f to the result. The domain is the set of all possible starting inputs, and the codomain is the set of possible outputs. When composing functions, the output of g must fit into the domain of f. This idea is important because it lets us build more complex processes by combining simpler ones.

1.6 Sequences and Patterns

Programs often work with ordered data like lists, steps, scores, or daily temperatures. A sequence is an ordered list of numbers or objects.

1.6.1 Examples

```
2, 4, 6, 8, ...
1, 1, 2, 3, 5, 8, ... (Fibonacci sequence)
33.0, 36.5, 31.0, ... (Tokyo's daily high temperatures in Celsius)
```

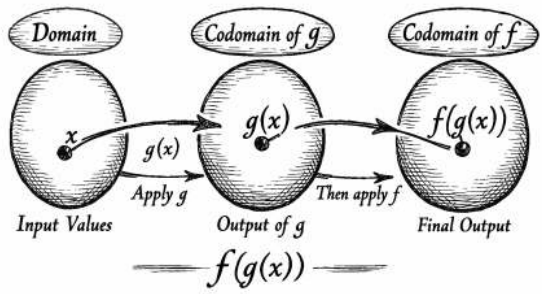

Figure 1.6: Function composition

1.7 Counting

Counting in discrete math is about finding how many ways something can happen. Counting helps in probability, game design, passwords, and algorithm efficiency.

1.7.1 Examples

- If you have `3` shirts and `2` pairs of pants, you can make `3 × 2 = 6` outfits.
- A 4-digit PIN number using digits `0` to `9` has `10 × 10 × 10 × 10 = 10,000` possibilities.

1.8 Algorithms

An algorithm is a step-by-step method for solving a problem. Programming is often really about writing algorithms. We will have a detailed description of algorithms in Section The Golden Principle of Al-Khwarizmi (Algorithm).

1.8.1 Example

How to find the largest number in a list:

- ❑ Start with the first number as the largest.
- ❑ Compare it to the next number.
- ❑ If the next number is bigger, replace it.
- ❑ Keep going until the end.

1.9 Graphs and Trees

A graph is a group of points connected by lines. Graphs and trees are used in maps, social networks, file folders, and search systems.

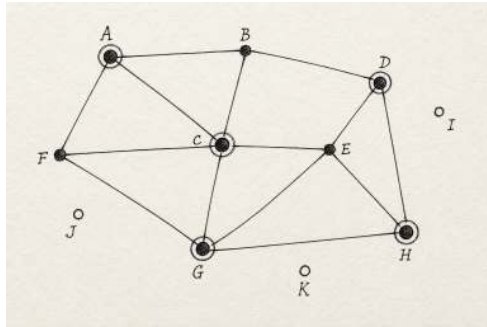

Figure 1.7: Graph

1.9.1 Examples

- ❑ Cities connected by roads
- ❑ People connected by friendships
- ❑ Web pages connected by links
- ❑ A tree is a special kind of graph that does not have any cycles (a path starting and ending at the same point).

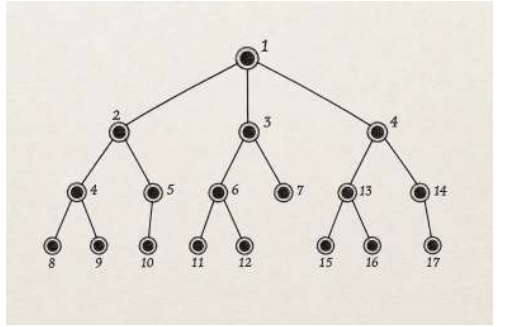

Figure 1.8: Tree

1.10 Binary Numbers

Computers store data using binary numbers, which is a series of values: `0` and `1`. For example, `1011`. The computer processor (chapter 3) performs the internal computations using the binary numbers. Decimal numbers are converted to binary by assigning each position of the binary number as a power of `2`. Figure 1.9 shows the binary representation of the decimal number `13`[1].

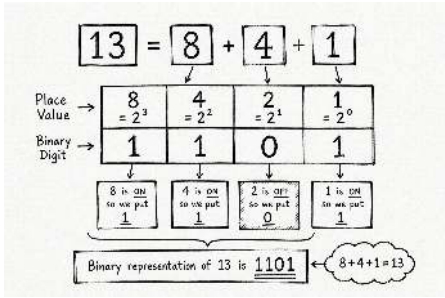

Figure 1.9: Binary Representation of a Decimal

All information in a computer is stored and computed using binary numbers - such information include text, image, numbers, sound, etc. For example a string of English characters is represented using decimal num-

[1]Negative decimal numbers are represented by setting 1 in one of the reserved bits - Two's complement

bers using the ASCII codes, then that decimal number is converted to a binary number using the powers of 2.

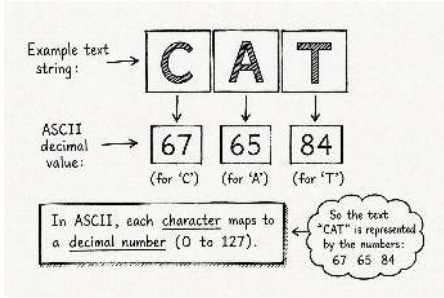

Figure 1.10: Representing a string using a decimal

1.11 Boolean Values

As we have seen in section 1.10, the numbers used by a computer to represent the variables and values are binary. Based on those variables, a computer program can check conditions and choose actions. A Boolean value is either: `True` or `False`. This is the foundation of decisions in programming. When you write code, you use:

- ❑ logic to make decisions
- ❑ variables and functions to organize work
- ❑ sequences to store ordered data
- ❑ counting to understand possibilities
- ❑ algorithms to solve problems
- ❑ binary and Booleans to understand how computers think

1.11.1 Simple Real-Life Example

Imagine making a login system:

- ❑ Logic: check if username and password are correct

- ❑ Boolean: logged in is True or False
- ❑ Sets: keep a list of allowed users
- ❑ Algorithms: search for a username
- ❑ Functions: make a function called login()

1.12 Hashing

Hashing is the idea of using a mathematical function, called a hash function, to turn data into a fixed-size value, often a number. You can think of it like giving information a short label based on its contents. For example, a string like ”apple” can be passed through a hash function and produce an integer such as 4372. The exact number depends on the hash function being used, but the key idea is that the same input will always produce the same output for that function. Hash functions help computers quickly compare, organize, and look up data instead of checking every character every time. You can think of the hash function as giving data a short numerical label based on its contents. For example, one simple way to hash a lowercase string is to use a polynomial hash with modulo a large prime:

$$h(s) = \left(c_0 \cdot 31^0 + c_1 \cdot 31^1 + c_2 \cdot 31^2 + \dots\right) \bmod 1{,}000{,}000{,}007$$

where $a = 1$, $b = 2$, $\dots$, $z = 26$. For instance, the string `"cat"` maps to

$$3 \cdot 31^0 + 1 \cdot 31^1 + 20 \cdot 31^2 = 3 + 31 + 19220 = 19254,$$

so its hash value is 19254. Hashing helps computers compare, organize, and look up data quickly.

1.13 Recursion

Recursion is a way of solving a problem by breaking it into smaller, similar versions of the same problem. Instead of solving everything at once, you define a rule that calls the same process again on a simpler case. There are two key parts:

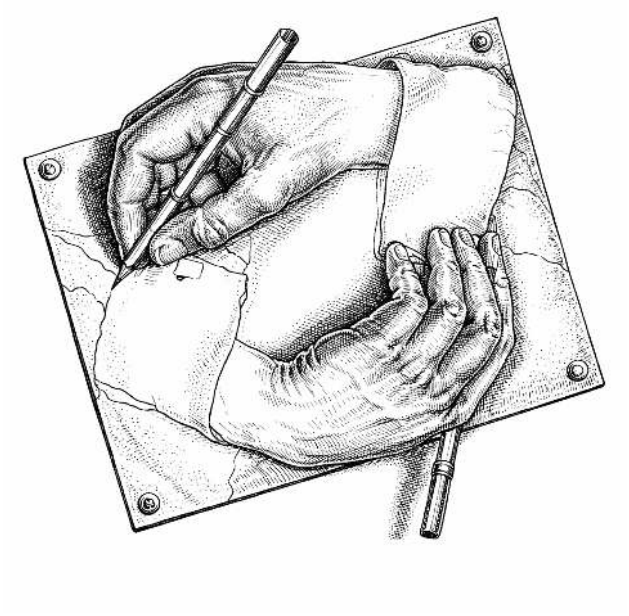

Figure 1.11: Two hands drawing one another

- ❑ Recursive step: the problem is reduced into a smaller version of itself.
- ❑ Base case: a simple situation where the answer is known and the process stops.

For example, to calculate a factorial (like 5!), you can think of it as: $factorial(5) = 5! = 5 \times 4 \times 3 \times 2 \times 1$. Using recursion, you define it as: $n! = n \times (n-1)!$ so we can write, $factorial(5) = 5 \times factorial(5-1)$. At each step of calculating the factorial value of a number, we can take that number and multiply it with the factorial of *one less than that number*. We stop when you reach 1! = 1 (the base case).

In other words, at each *recursive* step, we reduce the problem to a smaller problem until we reach the *base* case. Recursion is powerful because it helps solve complex problems in a clear, step-by-step way—but it must always have a base case, or it would keep going forever.

2

Problem Solving

Problem solving is a valuable skill that requires hard work to master. Although practice remains the only proven method to master problem solving, there are several methods and tools that can aid the reader in the learning process.

2.0.1 The Golden Principle of Al-Khwarizmi (Algorithm)

Al-Khwarizmi[1,2] provided a golden principle[3] for the world to create science with. The al-Khwarizmi principle states that all complex problems of science must be and can be solved by means of the following five simple steps:

- ❑ Break down each problem into a number of elemental or 'atomic' steps. An elemental or atomic step is one that cannot be simplified any further.

[1] wikipedia. *alKhwarizmi*. https://en.wikipedia.org/wiki/Al-Khwarizmi. Accessed: 2026-03-17. 2026

[2] Muhammad ibn Musa al-Khwarizmi (c.780 – c.850) was a mathematician active during the Islamic Golden Age, who produced Arabic-language works in mathematics, astronomy, and geography. Around 820, he worked at the House of Wisdom in Baghdad, the contemporary capital city of the Abbasid Caliphate. One of the most prominent scholars of the period, his works were widely influential on later authors, both in the Islamic world and Europe.

[3] kenatica. *goldenPrinciple*. https://kenatica.wordpress.com/2007/01/30/the-golden-principle-of-al-khwarizmi-algorithm/. Accessed: 2026-03-17. 2026

- ❑ Second, arrange all the elements or steps of the problem in an order or sequence, such that each element can be taken up and solved one at a time, without affecting other parts of the problem.
- ❑ Next, find a way to solve each of the elements separately. Because each element has been simplified as much as possible, it is very likely that the solution of an elemental step will itself be extremely simple or elemental making it available with relative ease.
- ❑ Then proceed to solve each element of the problem, either one at a time or several at a time, but in the correct order.
- ❑ Thus, when all steps are solved, the original problem itself has also been solved.

Khwarizmi used his own principle to solve many major problems of Algebra, Geometry, Astronomy, and other fields of science. Therefore he not only left us the golden rule but also many brilliant examples of how to use his rule. Figure 2.1 shows how we can break down and solve the problem *searching a word in the English dictionary* into a set of simple steps following the golden principle.

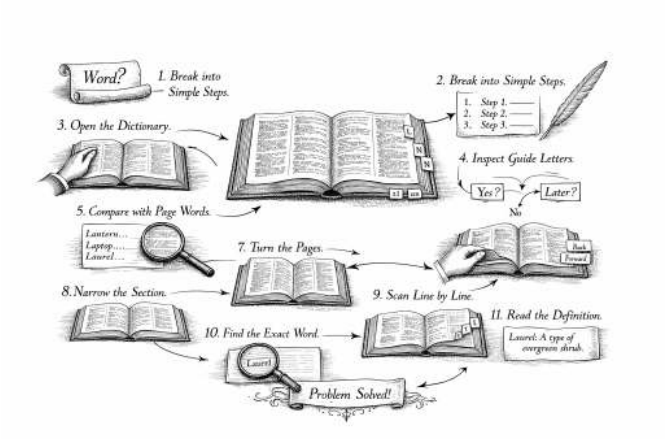

Figure 2.1: Steps to find a word in the dictionary

2.0.2 Algorithm

Algorithm[4] refers to a step-by-step procedure for solving a mathematical problem.

2.0.3 Example (gcd)

```python
def gcd_recursive(a, b):
    if b == 0:
        return abs(a)
    return gcd_recursive(b, a % b)

# Example usage
print(gcd_recursive(60, 48))  # Output: 12
```

2.1 Pseudocode

Pseudocode is a simple way to plan a program before writing it in a real programming language. It uses everyday language along with programming ideas such as input, output, decisions, and loops. This helps beginners focus on solving the problem step by step without worrying about punctuation or syntax errors. Pseudocode is useful because it helps you:

- ❑ understand the problem clearly
- ❑ organize your thoughts into steps
- ❑ find mistakes in your logic early
- ❑ translate your plan more easily into real code later

Most pseudocode uses three basic programming ideas:

- ❑ sequence: doing steps in order

[4] Al-Khwarizmi's Latinized name, *Algorismus*, became the name of the method used for computations and became the term *algorithm*

- ❑ selection: making decisions with IF and ELSE
- ❑ repetition: repeating steps with FOR or WHILE

For example, to add two numbers:

```
START
  INPUT first number
  INPUT second number
  SET sum = first number + second number
  OUTPUT sum
END
```

To check whether a number is even:

```
START
  INPUT number
  IF number mod 2 = 0 THEN
    OUTPUT "Even"
  ELSE
    OUTPUT "Odd"
  ENDIF
END
```

To print the numbers 1 to 5:

```
START
  FOR count = 1 TO 5
    OUTPUT count
  ENDFOR
END
```

Here is another example: calculating the greatest common divisor (GCD) of two numbers using Euclid's algorithm:

```
START
  INPUT a
```

```
  INPUT b

  WHILE b != 0
    SET temp = b
    SET b = a mod b
    SET a = temp
  ENDWHILE

  OUTPUT a
END
```

For example, if the inputs are 48 and 18:

```
48 mod 18 = 12
18 mod 12 = 6
12 mod 6 = 0
So the GCD is 6.
```

As another example, suppose we keep a weather log of daily temperature readings and want to find which city was the hottest. Breaking the problem into atomic steps gives:

```
START
  INPUT a list of (city, temperature) readings
  SET hottest = the first reading
  FOR each reading in the list
    IF reading temperature > hottest temperature
      SET hottest = reading
  OUTPUT hottest city
END
```

We will return to this weather log throughout the book, growing it from a single temperature into a small, tested program.

Pseudocode is a great beginner tool because it lets you practice problem-solving before coding. A good habit is to first read the problem, then write pseudocode, and finally convert it into code in a programming language

such as Python. That makes programming easier to understand and less overwhelming.

2.2 From Pseudocode to a Python Program

From our discussions we can create a clear process of solving any problem programmatically using a programming language like Python.

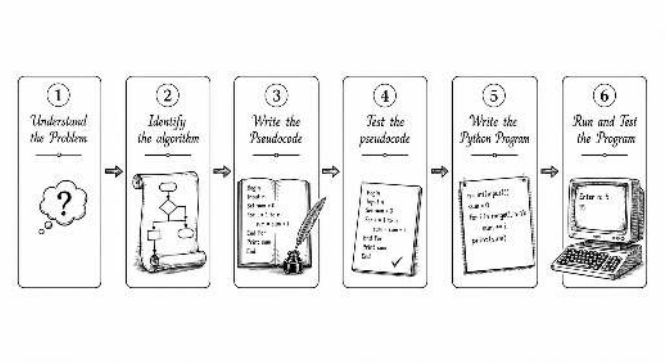

Figure 2.2: Problem Solving Process