VB.NET
from Zero to Production

A Complete, Practical Guide to Modern Visual Basic on .NET
From your first program to a tested desktop app, database, and web API — on .NET 10

First Edition

THENEXBOOKS

VB.NET from Zero to Production
A Complete, Practical Guide to Modern Visual Basic on .NET

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. .NET, Visual Basic, and the libraries and tools discussed in this book evolve; always consult the official documentation for the versions you use.

Microsoft, .NET, Visual Basic, Visual Studio, Visual Studio Code, Windows, Azure, and SQL Server are trademarks of the Microsoft group of companies. GitHub is a trademark of GitHub, Inc. Docker is a trademark of Docker, Inc. SQLite is in the public domain. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by Microsoft or the .NET Foundation.

Ashgrove Hardware, the business used as the running example in this book, is fictional, as are its stores, customers, products, prices, tax rate, and data.

Preface

Visual Basic has a long history of helping people turn ideas into working software. Generations of developers, analysts, and power users built their first real programs with it, and a remarkable amount of business software still runs on it today: order-entry screens, inventory tools, reporting utilities, and back-office applications that companies depend on every day.

Modern Visual Basic runs on modern .NET. The same language that powered classic desktop applications now targets .NET 10, with access to the same runtime, libraries, performance improvements, and tools as every other .NET language. You can write console tools, class libraries, Windows Forms and WPF applications, unit tests, and even ASP.NET Core Web API controllers in Visual Basic, and you can mix VB and C# projects in one solution when that makes sense.

This book teaches VB.NET the way it is used in practice. We start from zero and build one real application step by step, introducing each idea at the moment it solves a problem we actually have. By the end, a small console program has grown into a tested Windows Forms back-office application with a SQLite database, a Web API, and a release pipeline — and you will understand not only *how* to write Visual Basic, but *why* professional teams structure, test, and ship their code the way they do.

Who This Book Is For

- **Beginners** who want to learn programming with a readable, forgiving language and are ready to practice.
- **VBA and VB6 developers** who want to move their skills to modern .NET.
- **.NET Framework VB developers** who maintain existing applications and want to move them to .NET 10 with confidence.
- **Teams with a VB code base** who need a clear, honest picture of what VB does well today, and when to bring in C#.

What You Will Learn

- How .NET works, how to set up Visual Studio and the .NET SDK, and how to create, build, run, and debug VB projects.
- The Visual Basic language from the ground up: types, Option Strict, strings, control flow, procedures, classes, inheritance, interfaces, structures, enums, and nullable types.
- Modules, namespaces, extension methods, generics, collections, delegates, lambdas, and events.
- LINQ with Visual Basic's rich query syntax, XML literals, JSON, exceptions, files, CSV, and dates and times.
- Async and await, cancellation, unit testing with xUnit and MSTest, dependency injection, configuration, and logging.

- Windows Forms applications with data binding and validation, ADO.NET, Dapper, Entity Framework Core, and an ASP.NET Core Web API.
- Modernizing legacy VB6 and .NET Framework code, packaging and deployment, diagnostics, security, and CI.

How This Book Is Organized

Part I — Getting Started (Chapters 1–5) sets up .NET and covers the language fundamentals. **Part II — Object-Oriented Visual Basic** (Chapters 6–10) covers classes, inheritance, interfaces, structures, modules, generics, and collections. **Part III — Core Language and Libraries** (Chapters 11–15) covers delegates and events, LINQ, XML and JSON, error handling, and files and dates. **Part IV — Professional VB.NET** (Chapters 16–19) covers async programming, testing, the generic host, and solution structure. **Part V — Desktop, Data, and Web** (Chapters 20–24) builds the Ashgrove Windows Forms app, its database layer, and its Web API. **Part VI — Production** (Chapters 25–28) covers modernization, deployment, diagnostics and security, and a complete case study. The appendices provide a cheat sheet, templates, a production checklist, and a glossary.

Each chapter ends with exercises and a **Key Takeaways** box for quick review.

The Running Example: Ashgrove Hardware

Throughout the book we build the software for **Ashgrove Hardware**, a fictional family-owned chain of three hardware stores — Main Street and Brookside in the town of Ashgrove, and a newer store in Fernvale — that also serves contractors on trade accounts. It starts with a simple need: know what is in stock where. Over the chapters, a console tool grows to import product lists, track stock, price contractor orders, produce invoices and sales reports, and warn when shelves run low. In Part V it becomes a Windows Forms back-office application with a real database and a Web API, and in Part VI we ship it.

Versions and Conventions

Examples target **.NET 10**, a long-term support release, and **Visual Basic 16.9**, the current language version. Visual Basic is a stable language: Microsoft supports it on modern .NET and keeps it working well for its core scenarios, but it is not gaining major new language features. Most code in this book also runs on .NET 8 and .NET 9. Windows Forms chapters require Windows; everything else runs on Windows, macOS, or Linux. Where details may change between releases, a Note points you to the official documentation rather than guessing.

> **TIP**
>
> A recommended practice or a technique that makes your work easier.

> **NOTE**
>
> Background information, version differences, or a useful aside.

> **WARNING**
>
> A common mistake or something that can cause bugs, data loss, or security problems.

Code appears in `monospace`. Long lines are wrapped to fit the page. File names above code blocks, such as `src/Ashgrove.Core/Product.vb`, tell you where the code lives. All code assumes `Option Strict On`, which Chapter 3 explains.

Contents

Preface

PART I — GETTING STARTED

1 Why VB.NET Today? The Language, the Platform, and Your Setup

2 Your First Programs: Modules, Sub Main, Projects, and the dotnet CLI

3 Data Types, Variables, Operators, and Option Strict

4 Strings, Formatting, and Console Input and Output

5 Control Flow, Procedures, and Functions

PART II — OBJECT-ORIENTED VISUAL BASIC

6 Classes, Properties, and Constructors

7 Inheritance, Interfaces, and Polymorphism

8 Structures, Enumerations, Nothing, and Nullable Value Types

9 Modules, Namespaces, Shared Members, and Extension Methods

10 Generics and Collections

PART III — CORE LANGUAGE AND LIBRARIES

11 Delegates, Lambdas, and Events

12 LINQ with Visual Basic Query Syntax

13 XML Literals, LINQ to XML, and JSON

14 Exceptions, Error Handling, and Debugging

15 Files, Streams, CSV, and Dates and Times

PART IV — PROFESSIONAL VB.NET

16 Async and Await, Tasks, and Cancellation

17 Unit Testing with xUnit and MSTest

18 Dependency Injection, Configuration, and Logging with the Generic Host

19 Solutions, Class Libraries, NuGet, and Working Alongside C#

PART V — DESKTOP, DATA, AND WEB

20 Windows Forms Fundamentals

21 Building a Real Windows Forms App: Data Binding, Validation, and Layout

22 Data Access with ADO.NET, SQLite, and Dapper

23 Entity Framework Core with Visual Basic

24 Building and Calling a Web API with ASP.NET Core

PART VI — PRODUCTION

25 Modernizing Legacy Code: From VB6 and .NET Framework to Modern .NET

26 Packaging and Deployment: dotnet publish, Single-File, MSIX, and ClickOnce

27 Diagnostics, Performance, Security, and CI for VB.NET Applications

28 Case Study: Ashgrove Hardware from Console Tool to Production

APPENDICES

A VB.NET Cheat Sheet

B Templates and Reference

C Production Checklist

D Glossary

PART I

Getting Started

What .NET is and where Visual Basic fits today, setting up your tools, your first programs and projects, and the fundamentals every VB program is built from: types, Option Strict, operators, strings, console input and output, control flow, and procedures.

1 · Why VB.NET Today?

2 · Your First Programs

3 · Data Types and Option Strict

4 · Strings and Console I/O

5 · Control Flow and Procedures

CHAPTER 1 · PART I — GETTING STARTED

Why VB.NET Today? The Language, the Platform, and Your Setup

Before we write a real program, we need a clear picture of what Visual Basic on .NET actually is, where it fits in the modern Microsoft ecosystem, and how to install the tools. By the end of this chapter you will have a working .NET 10 setup, your first Visual Basic program running, and an introduction to Ashgrove Hardware, the business we will build software for throughout the book.

What You Are Getting Into

Visual Basic .NET (usually just called Visual Basic, or VB) is a general-purpose, object-oriented programming language that runs on .NET, Microsoft's open-source, cross-platform development platform. It is a language designed to be readable: keywords are English words, blocks end with explicit markers such as `End If` and `End Sub`, and the compiler is forgiving about capitalization. Those qualities made it the first language for a generation of professional and hobbyist developers, and they still make it a pleasant way to learn programming today.

This book takes you from zero to production. "Zero" means we assume no prior knowledge of VB.NET. If you have never programmed before, you will find every concept explained from first principles, with a runnable example you can type in and change. If you are coming from Office VBA or classic Visual Basic 6, you will recognize much of the syntax and we will point out the places where modern VB behaves differently. If you already maintain VB applications on the older .NET Framework, the later chapters show how to move them to .NET 10 and keep them healthy for years.

"Production" means we do not stop at toy programs. Over the course of the book we build a real application for a fictional business, test it, connect it to a database, put a desktop user interface and a small web API on top of it, and then package, deploy, and harden it. Along the way you will learn not just the language but the habits that professional .NET developers rely on: strict typing, clear project structure, automated tests, configuration, logging, and repeatable builds.

What .NET Actually Is

When people say ".NET" they usually mean several things at once. It helps to separate them, because each piece plays a different role when you build and run a program.

The runtime and the CLR

The **runtime** is the engine that executes your program. Its core is the **Common Language Runtime (CLR)**. The CLR loads your compiled code, manages memory for you through a **garbage collector** (so you never free memory by hand), enforces type safety, handles exceptions, and coordinates threads. Every .NET language, whether Visual Basic, C#, or F#, runs on the same CLR. That shared foundation is why a library written in C# can be used from VB and the other way around without any special glue.

Intermediate Language and the JIT compiler

The VB compiler does not translate your source code directly into machine instructions for a particular processor. Instead it produces **Intermediate Language (IL)**, a processor-independent instruction set, and packages it into an **assembly**, which is a `.dll` file (and, for an application, a small launcher `.exe` as well). When your program runs, the CLR's **just-in-time (JIT) compiler** translates IL into native machine code for the actual CPU, method by method, as each method is first called. The native code is kept in memory, so subsequent calls run at full speed.

The following diagram shows the journey from a VB source file to a running process.

From source code to running program

```
Program.vb  --->  VB compiler (vbc, run by the SDK)
                        |
                        v
            HelloAshgrove.dll   (IL + metadata)
                        |
                        v
        .NET runtime: CLR loads the assembly
                        |
                        v
        JIT compiler: IL ---> native x64/Arm64 code
                        |
                        v
                Your program runs
```

This two-step design has practical benefits. The same compiled assembly runs on Windows, Linux, and macOS, on Intel and Arm processors, as long as a compatible runtime is present. The JIT can also optimize for the exact machine it is running on. Modern .NET adds further techniques, such as tiered compilation and ahead-of-time options for some app types, but you do not need to understand them yet; Chapter 26 returns to them when we talk about deployment.

The Base Class Library

The **Base Class Library (BCL)** is the large set of ready-made types that ship with the runtime. It covers text, numbers, dates and times, collections, files, networking, JSON, XML, cryptography, threading, and much more. When you write `Console.WriteLine` or `Math.Round`, you are calling into the

BCL. A big part of becoming productive in any .NET language is learning what the BCL already provides so that you do not reinvent it. Because the BCL is shared, everything you learn about it in VB transfers directly to C# and back.

The SDK

The **.NET SDK** (Software Development Kit) is what developers install. It contains the runtime, the compilers for VB, C#, and F#, the MSBuild build engine, project templates, the NuGet package client, and the `dotnet` command-line tool that ties them together. End users who only *run* your application need just the runtime (or not even that, if you publish a self-contained app, as we do in Chapter 26). Developers need the SDK.

Piece	What it does	Who needs it
CLR	Loads code, manages memory, runs the JIT, handles exceptions	Everyone running a .NET app
BCL	Ready-made types for text, files, collections, networking, and more	Everyone (ships with the runtime)
Runtime	CLR plus BCL, installed per version	Users of framework-dependent apps
SDK	Runtime plus compilers, MSBuild, templates, NuGet, the `dotnet` CLI	Developers

One .NET, several generations

You will meet older names in documentation and job descriptions, so it is worth knowing how they relate. The original **.NET Framework** is Windows-only and is now in maintenance mode: it still ships with Windows and is supported, but it receives no new features. **.NET Core** was the cross-platform, open-source rewrite. Starting with .NET 5, Microsoft dropped the "Core" suffix, and the modern line is simply called .NET: .NET 5, 6, 7, 8, 9, and now .NET 10. When this book says ".NET" without qualification, it means this modern line. When we mean the older Windows-only product, we always say ".NET Framework."

A Short History of Visual Basic

Visual Basic descends from BASIC, a language created at Dartmouth College in the 1960s to make programming approachable for students. Microsoft's Visual Basic, launched in the early 1990s, paired a BASIC dialect with a drag-and-drop form designer and became one of the most popular ways to build Windows applications; Visual Basic 6 (VB6), its final classic version, powered countless business systems and also lived on as VBA inside Microsoft Office. When Microsoft introduced the .NET Framework in the early 2000s, it rebuilt the language as Visual Basic .NET: still recognizably Visual Basic, but fully object-oriented, compiled to IL, and running on the CLR alongside C#. That transition was a genuine break, since VB6 is a COM-based product and its code does not simply recompile as

.NET, which is why Chapter 25 treats VB6 migration as a guided rewrite. Since then, VB.NET has gained generics, LINQ, XML literals, async and await, and many other features, with language version 16.9 being the last to receive notable additions.

Where Visual Basic Fits Today

It is important to be honest and precise about VB's position, because it affects how you plan projects and careers.

Microsoft's stated strategy is that Visual Basic is **supported on modern .NET**, but the **language itself will not evolve** with major new features. Instead of chasing every new idea that appears in C#, VB stays stable. Microsoft focuses on keeping it working well for its core scenarios, namely Windows Forms, WPF, console applications, and class libraries, and on making sure VB code can *consume* some newer .NET APIs that are written in C#. Visual Basic 16.9 was the last version with notable language additions; one example is the ability to consume `init`-only properties defined in C# libraries.

What does "stable" mean in practice? It means the code you write today will keep compiling and running on future .NET versions with very little churn. You get the performance improvements, security fixes, and library additions that each new .NET release brings, because those live in the runtime and the BCL, not in the language. What you do not get are new syntax features. For many business applications, that stability is a feature rather than a drawback: the language you learn from this book is the language you will use for the life of the project.

> **NOTE**
>
> Support for a language and new features in a language are different things. VB applications on .NET 10 are fully supported by Microsoft for the app types this book covers. The language simply is not gaining new syntax. When a new .NET feature requires language changes that VB does not have, the usual answer is to put that piece in a small C# project and call it from VB, which Chapter 19 shows you how to do.

This position also explains who uses VB today. There is a very large amount of VB.NET code in production, especially internal business applications, desktop tools, and line-of-business systems. Organizations need developers who can maintain, extend, test, and modernize that code. There are also many people for whom VB's readability makes it the most comfortable way to automate and build things. If you are in either group, investing in modern VB skills is entirely reasonable, and because .NET is shared, nearly everything you learn about the platform carries over if you later work in C#.

What You Can Build with Visual Basic

The .NET SDK ships Visual Basic project templates for the most common app types. The table below summarizes what works well from VB and what needs C#.

App type	VB support	Where in this book
Console applications	Full, with a template (`console`)	Parts I–IV
Class libraries	Full, with a template (`classlib`)	From Chapter 2 on
Windows Forms	Full, with a template and the Visual Studio designer	Chapters 20–21
WPF	Full, with a template (`wpf`)	Mentioned only
Unit tests	xUnit, MSTest, and NUnit templates	Chapter 17
ASP.NET Core Web API	Controllers work from a VB project; no VB template	Chapter 24
Razor Pages, MVC views, Blazor	Not available in VB (Razor is C#-only)	Use C#
.NET MAUI mobile apps	No VB templates or support	Use C#

The ASP.NET Core row deserves a word of explanation. There is no `dotnet new webapi -lang VB` template. However, a VB project can use the web SDK by setting `<Project Sdk="Microsoft.NET.Sdk.Web">` in its project file, and API controllers written in VB work fine. Chapter 24 builds exactly that. Razor syntax, which mixes HTML with code, is defined only for C#, so web user interfaces are a C# job.

There is also a small set of language-level features that VB does not have and will not get. Examples include records, `required` members, primary constructors, pattern-matching `switch` expressions, and asynchronous streams. VB also restricts the use of `ref struct` types such as `Span(Of T)`, so performance-critical span-based code is usually written in C# and called from VB. None of these limits stop you from building complete, well-structured applications, and we point them out where they matter.

.NET Versions: LTS and STS

Modern .NET ships a new major version once a year, typically in November. Releases alternate between two support tracks:

- **LTS (Long-Term Support)** releases are the even-numbered versions such as .NET 8 and .NET 10. They receive patches for several years and are the usual choice for business applications.
- **STS (Standard-Term Support)** releases are the odd-numbered versions such as .NET 9. They have a shorter support window and suit teams that are happy to upgrade frequently.

Within a major version, Microsoft ships regular servicing updates (for example 10.0.1, 10.0.2, and so on) that contain security and reliability fixes. Those updates are drop-in compatible, and you should keep them current. Exact support durations have changed over time, so check Microsoft's official .NET support policy page when you plan an upgrade.

This book targets **.NET 10**, an LTS release, and **Visual Basic 16.9**. Most of the code also runs unchanged on .NET 8 and .NET 9; where something is specific to .NET 10, we say so. The Windows Forms chapters require Windows. Everything else, including the console application, class libraries, tests, and the web API, runs on Windows, Linux, and macOS.

> **TIP**
> If you work on a team, agree on one SDK version for a repository. A small `global.json` file in the repository root can pin the SDK version that `dotnet` commands use in that folder. Running `dotnet new globaljson` creates one for the SDK you currently have installed.

Installing Your Tools

You need two things: the .NET 10 SDK and an editor. On Windows, we strongly recommend Visual Studio, because it is the most complete environment for Visual Basic and the only one with the Windows Forms designer.

Option 1: Visual Studio on Windows (recommended)

1. Download Visual Studio from Microsoft's Visual Studio website. The free **Community** edition is fine for learning, open-source work, and small teams; check its license terms for your situation.
2. In the Visual Studio Installer, select the **.NET desktop development** workload. This installs the .NET SDK, the VB and C# compilers, the Windows Forms and WPF designers, and the debugger. If you plan to follow Chapter 24 inside Visual Studio as well, add the **ASP.NET and web development** workload.
3. Make sure the installed version supports .NET 10. A recent version of Visual Studio is required for each new .NET release, so let the installer update itself first.

Option 2: The SDK only, on any operating system

If you prefer working from a terminal, or you are on Linux or macOS, install just the .NET 10 SDK from Microsoft's .NET download page. On Windows you can also use the Windows Package Manager:

Terminal (Windows)

```
$ winget install Microsoft.DotNet.SDK.10
```

Package identifiers occasionally change, so if that command does not find the SDK, run `winget search dotnet` to see the current names. On Linux, Microsoft publishes packages for popular distributions, and many distributions include .NET in their own repositories. On macOS, use the installer package from the download page.

For editing, Visual Studio Code works on every platform. Be aware that its Visual Basic support is basic: you get syntax highlighting and can build and run through the terminal, but the rich editing experience that C# developers enjoy in VS Code is not the same for VB, and there is no Windows

Forms designer. For the console, library, test, and API chapters, VS Code plus the terminal is workable. For the Windows Forms chapters, use Visual Studio.

> **WARNING**
>
> Installing several SDKs side by side is normal and safe; .NET is designed for it. What causes confusion is assuming that the newest SDK is always the one in use. A `global.json` file in a parent folder can silently pin an older SDK. If a command behaves unexpectedly, run `dotnet --version` in that folder to see which SDK is actually active.

Checking Your Installation

Open a terminal (Windows Terminal, PowerShell, Command Prompt, or a Linux/macOS shell) and run the following commands. Throughout this book, a line starting with `$` is a command you type; the lines after it are the output. Do not type the `$` itself.

Terminal

```
$ dotnet --version
10.0.401
```

Your exact number will probably differ. What matters is that it starts with `10.0`. The first two parts are the .NET version; the last part identifies the SDK feature band and patch level. If you see an older version, either the .NET 10 SDK is not installed or a `global.json` file is pinning an older one.

To see every SDK and runtime on the machine, use `dotnet --list-sdks` and `dotnet --list-runtimes`. For a full report, use `dotnet --info`. Its output is long; here is an abbreviated version from a Windows machine:

Terminal

```
$ dotnet --info
.NET SDK:
 Version:           10.0.401
 ...

Runtime Environment:
 OS Name:     Windows
 OS Version:  10.0.22631
 OS Platform: Windows
 RID:         win-x64
 Base Path:   C:\Program Files\dotnet\sdk\10.0.401\
 ...

.NET runtimes installed:
  Microsoft.AspNetCore.App 10.0.12 [C:\Program Files\dotnet\shared\...]
  Microsoft.NETCore.App 10.0.12 [C:\Program Files\dotnet\shared\...]
  Microsoft.WindowsDesktop.App 10.0.12 [C:\Program Files\dotnet\shared\...]
 ...
```

Three runtimes appear. `Microsoft.NETCore.App` is the base runtime every .NET app needs. `Microsoft.AspNetCore.App` adds the web stack used in Chapter 24. `Microsoft.WindowsDesktop.App`, present only on Windows, adds Windows Forms and WPF. The **RID** (runtime identifier) `win-x64` describes the operating system and processor; you will meet RIDs again when publishing self-contained apps.

Finally, confirm that the Visual Basic templates are available:

Terminal

```
$ dotnet new list --language VB
These templates matched your input: --language='VB'

Template Name                 Short Name          Language  Tags
----------------------------  ------------------  --------  --------------------
Class Library                 classlib            C#,F#,VB  Common/Library
Console App                   console             C#,F#,VB  Common/Console
MSTest Test Project           mstest              C#,F#,VB  Test/MSTest/Desktop/Web
NUnit Test Project            nunit               C#,F#,VB  Test/NUnit/Desktop/Web
Windows Forms App             winforms            C#,VB     Common/WinForms
WPF Application               wpf                 C#,VB     Common/WPF
xUnit Test Project            xunit               C#,F#,VB  Test/xUnit/Desktop/Web
...
```

The list is abbreviated here. You may see additional entries, such as item templates or library variants, and more if you have installed optional workloads. The important ones for this book are `console`, `classlib`, `winforms`, and the three test templates.

Your First Visual Basic Program

Let's make sure everything works end to end. Create a folder for your book exercises, open a terminal in it, and run:

Terminal

```
$ dotnet new console -lang VB -o HelloAshgrove
The template "Console App" was created successfully.
...
$ cd HelloAshgrove
$ dotnet run
Hello World!
```

The `-lang VB` option is essential. Without it, `dotnet new` creates a C# project, because C# is the default language. The `-o` option names the output folder, and the folder name also becomes the project name. `dotnet run` restores any dependencies, builds the project, and runs it, all in one step.

Now open `Program.vb` in your editor and replace its contents with the following listing. Do not worry about every detail; Chapter 2 explains the structure of a VB program line by line.

HelloAshgrove/Program.vb

```
Imports System.Runtime.InteropServices

Module Program
    Sub Main(args As String())
        Console.WriteLine("Welcome to Ashgrove Hardware!")
        Console.WriteLine($"Running on {RuntimeInformation.FrameworkDescription}")
        Console.WriteLine($"Operating system: {RuntimeInformation.OSDescription}")
        Console.WriteLine(
            $"Process architecture: {RuntimeInformation.ProcessArchitecture}")
    End Sub
End Module
```

Run it again:

Terminal

```
$ dotnet run
Welcome to Ashgrove Hardware!
Running on .NET 10.0.12
Operating system: Microsoft Windows 10.0.22631
Process architecture: X64
```

The version, operating system, and architecture lines will reflect your machine. On an Arm-based Mac you would see `Arm64`, and on Linux the operating system line names your distribution. The important thing is that the program was compiled and run by the .NET 10 runtime.

A few things are already visible in this short listing. `Imports` at the top brings a namespace into scope so that we can write `RuntimeInformation` instead of its full name. `Module Program` and `End Module` form a container for code. `Sub Main` is the entry point where execution starts. The `$"..."` syntax is an **interpolated string**, in which expressions inside braces are evaluated and inserted into the text. Chapter 4 covers interpolation and formatting in depth.

A first taste of business logic

One more small program gives you a flavor of the kind of code this book is about. Ashgrove Hardware gives contractor customers 10% off the shelf price. Here is that calculation for the store's best-selling claw hammer:

FirstTaste/Program.vb

```
Module Program
    Sub Main(args As String())
        Dim productName As String = "Claw Hammer 16 oz"
        Dim retailPrice As Decimal = 19.99D
        Dim contractorPrice As Decimal =
            Math.Round(retailPrice * 0.9D, 2, MidpointRounding.AwayFromZero)

        Console.WriteLine($"Product:          {productName}")
        Console.WriteLine($"Retail price:     {retailPrice}")
```

```
        Console.WriteLine($"Contractor price: {contractorPrice}")
    End Sub
End Module
```

Terminal

```
$ dotnet run
Product:          Claw Hammer 16 oz
Retail price:     19.99
Contractor price: 17.99
```

Notice the `D` suffix on `19.99D` and `0.9D`. It tells the compiler that these numbers are `Decimal` values, the type .NET provides for exact base-10 arithmetic. Money must always be stored in `Decimal`, never in floating-point types like `Double`, which cannot represent most cents values exactly. The calculation 19.99 × 0.9 gives 17.991, which `Math.Round` rounds to two decimal places. Chapter 3 explains numeric types, suffixes, and rounding in detail; for now, just notice that even a one-line business rule involves deliberate choices.

Meet Ashgrove Hardware

Learning a language through disconnected snippets makes it hard to see how the pieces fit together. So, throughout this book, we build software for one fictional business: **Ashgrove Hardware**, a family-owned chain of three hardware stores. Ashgrove sells to the public and also runs trade accounts for contractors, such as builders, plumbers, and electricians, who buy regularly and receive a discount.

The stores

Id	Code	Name	City
1	ASH	Main Street	Ashgrove
2	BRK	Brookside	Ashgrove
3	FRV	Fernvale	Fernvale

The product catalog

Every product has a SKU (stock-keeping unit) in the form `HW-10001`, a name, a category, a unit of measure, and a price. Here is the core catalog we will use from Chapter 2 to Chapter 28:

Id	Sku	Name	Category	Unit	Price
1	HW-10001	Claw Hammer 16 oz	Tools	Each	19.99
2	HW-10002	Cordless Drill 18V	PowerTools	Each	89.00
3	HW-10003	Wood Screws #8 x 1-1/4 in, 100 pack	Fasteners	Pack	7.49
4	HW-10004	Interior Paint Eggshell White 1 gal	Paint	Gallon	34.50

5	HW-10005	Garden Hose 50 ft	Garden	Each	27.95
6	HW-10006	LED Bulb A19, 4 pack	Electrical	Pack	12.99
7	HW-10007	PVC Pipe 1/2 in x 10 ft	Plumbing	Each	6.25

Customers and business rules

Ashgrove has three customers in our sample data: Dana Whitfield (account `C-1001`, a contractor with a 5,000.00 credit limit), Luis Moreno (`C-1002`, a retail customer), and Grace Kim (`C-1003`, a contractor with a 2,500.00 credit limit). The business follows a handful of simple rules that our code must implement exactly:

- **Contractor pricing:** customers marked as contractors get 10% off the unit price, rounded to cents.
- **Sales tax:** 7.25% of the invoice subtotal, rounded to cents. This is a sample rate for the fictional town of Ashgrove, not a real tax rate. The total is the subtotal plus tax.
- **Low stock:** a product is low at a store when its quantity is at or below its reorder point.
- **Invoice numbers:** the format `INV-yyyy-nnnn`, numbered sequentially within each year, starting with `INV-2026-0001`.

Those rules lead to a standard example that we will compute, test, and print many times. Invoice `INV-2026-0001` is for Dana Whitfield at the Main Street store: two claw hammers at the contractor price of 17.99, one cordless drill at 80.10, and three packs of wood screws at 6.74. The subtotal is 35.98 + 80.10 + 20.22 = 136.30. Tax at 7.25% is 9.881750, which rounds to 9.88, giving a total of 146.18. When a later chapter's code produces 146.18 for this invoice, you will know the pieces are working together correctly.

How the application grows

The software grows with the book, part by part:

The Ashgrove roadmap

```
Part I   Getting Started        console "ashgrove" tool + Ashgrove.Core library
Part II  Object-Oriented VB     Product, Store, Customer, PricingService,
                                InvoiceCalculator, repositories
Part III Core Language          events, LINQ reports, XML/JSON export,
                                error handling, CSV import
Part IV  Professional VB.NET    async, unit tests, generic host, DI, config,
                                logging, multi-project solution
Part V   Desktop, Data, Web     Windows Forms back office, SQLite database,
                                EF Core, ASP.NET Core Web API
Part VI  Production             modernization, packaging, diagnostics, CI,
                                final case study
```

By the end, the solution, `Ashgrove.sln`, contains a domain library (`Ashgrove.Core`), a command-line tool (`Ashgrove.Cli`, run as `ashgrove`), a data access library (`Ashgrove.Data`), a Windows Forms back-office application (`Ashgrove.Desktop`), a web API (`Ashgrove.Api`), and a test project (`Ashgrove.Core.Tests`). Each project is introduced when we need it, so you never face a wall of unexplained structure.

How to Use This Book

Each chapter builds on the previous ones, so reading in order works best, especially in Parts I and II. If you already know some VB, you can skim the early chapters, but do look at the "Ashgrove" sections, because later chapters rely on the code built there.

A few conventions apply throughout:

- A label above a code block shows the file it belongs to, such as `src/Ashgrove.Core/Product.vb`, or `Terminal` for commands and output.
- Commands are shown with a `$` prompt. They work in PowerShell, Command Prompt, and Unix shells unless we say otherwise. Paths use forward slashes, which the `dotnet` tool accepts on Windows too.
- Program output in this book was produced by actually running the code. If your output differs, the difference usually comes from your machine's culture settings (number and date formats), the .NET patch version, or a typo worth hunting down.
- Every project uses `Option Strict On`, which makes the compiler reject risky implicit conversions. Chapter 3 explains why this matters so much in VB.

Most importantly, type the code in and change it. Reading a program teaches you what it does; breaking it and fixing it teaches you why. Each chapter ends with a short "Try It Yourself" list for exactly that purpose.

Try It Yourself

1. Run `dotnet --list-sdks` and `dotnet --list-runtimes`. Note which .NET versions you have and which runtimes (base, ASP.NET Core, Windows Desktop) are present.
2. In a new empty folder, run `dotnet new globaljson`, open the generated file, and run `dotnet --version` there. Then delete the file and check again.
3. Change `FirstTaste` to compute the contractor price for the Cordless Drill 18V (89.00). You should get 80.10.
4. Run `dotnet new console -o CSharpPeek` without `-lang VB` and compare the generated file with the VB version. Which language is the default?
5. Add a line to `HelloAshgrove` that prints `Environment.MachineName`, and another that prints `DateTime.Now`.