The Bare Minimum

Windows Edition

Computer Skills You Need Before Starting The Data Scientist or the Certificate in Python for Finance

Dr. Yves J. Hilpisch[1]

September 19, 2026

[1]Get in touch: `https://linktr.ee/dyjh`. Web page: `https://thedatascientist.dev`. Research, structuring, drafting, and visualizations were supported by large language models (LLMs) as co-writing tools under human direction. Comments and feedback are welcome.

Preface

Many people who would enjoy learning Python, data science, or quantitative finance hesitate for one practical reason: they are unsure whether their computer skills are sufficient. That uncertainty is unnecessary, and removing it takes an afternoon.

This is not a programming course, and it will not turn you into a computer expert. It covers the small set of everyday skills that serious technical training quietly assumes: working with files and folders, editing plain text, and typing a few commands into a terminal. That is genuinely all the computer literacy you need before you begin *The Data Scientist* (TDS)[2] or the *Certificate in Python for Finance* (CPF)[3].

The same skills are also the technical prerequisites for *The AI Engineer* and *The Crypto Engineer*. Those two programs go further: they expect Python and data-science fluency at the level of *The Data Scientist*, so the natural path runs through that program first.

Who This Is For

You will benefit most if you are:

- a complete beginner considering *The Data Scientist* or the *Certificate in Python for Finance*,
- a reader who is comfortable with ideas but not with tooling,
- or anyone who has never opened a terminal and would like that to stop feeling like a barrier.

The guide comes in two editions. This is the **Windows edition**; a macOS edition covers the same concepts with macOS-specific instructions. The conceptual material is identical, because the concepts are the same on every computer.

Read actively. Whenever a page suggests a small action — create a folder, save a file, type a command — actually do it. The core guide and its essential exercises can be worked through in roughly an hour; the optional excursions and the Readiness Gym provide additional practice if you want it. If anything on your screen looks different from what is described here, ask a large language model (LLM) such as ChatGPT or Claude; the vocabulary you are about to learn is exactly what you need to formulate good questions.

By the end, you should be able to say: files, folders, text editors, and the terminal are ordinary tools, and I am ready to start learning Python.

[2] `https://thedatascientist.dev`
[3] `https://cpfprogram.com`

Contents

List of Figures

1 You Need Much Less Than You Think

We start with a widespread misconception: that learning Python or data science requires extensive technical preparation. It does not. The entry barrier is much smaller than most beginners assume, and a few pages from now you will have a realistic picture of how small it really is.

1.1 What You Do Not Need

A beginner does **not** need to understand any of the following before starting *The Data Scientist* or the *Certificate in Python for Finance*:

- programming or Python,
- algorithms or data structures,
- databases,
- cloud computing,
- artificial intelligence,
- Linux,
- networking,
- computer architecture.

All of these subjects belong to later learning. Some of them will appear naturally as you progress; others you may never need at all. Treating them as prerequisites is the single most common way beginners talk themselves out of starting.

1.2 What You Actually Need

The practical starting point is much simpler. You should be reasonably comfortable:

- using a keyboard and a mouse or trackpad,
- launching applications,
- creating and organizing folders,
- opening and saving files,
- editing text,
- opening a terminal,

- entering a simple command and pressing Enter.

None of these is a programming skill. They are basic computer-working skills, and you can close any gaps in an hour or two of experimentation — starting right here.

1.3 The Entry Barrier, Visualized

Figure 1.1 shows where these skills sit relative to everything that follows. The first two steps are deliberately small: they are what you are practicing right now, and they are learnable in an afternoon. Python and everything beyond it is the actual journey, and it begins only after these small steps.

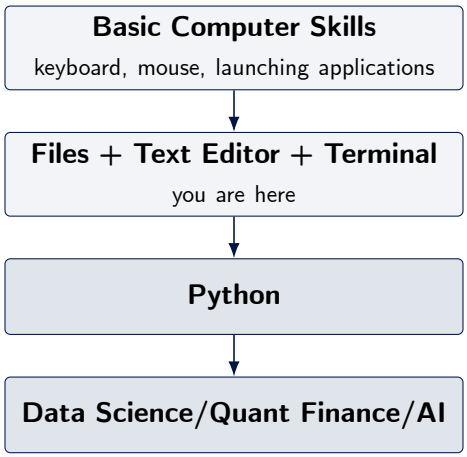

Figure 1.1: The entry barrier is much smaller than the learning journey that follows.

The Central Point

The technical prerequisites for getting started are minimal, practical, and easy to acquire. If you can work with files, edit text, and type a few commands into a terminal, you are technically ready to begin.

1.4 Where We Are Heading Next

Before we can talk about files and terminals concretely, we need one shared mental model: what an operating system is and where files, applications, and terminals fit inside it. Once that model is in place, everything else becomes concrete.

Try in 60 Seconds

Say the following sentence out loud: "To start *The Data Scientist*, I need to be comfortable with files, text editing, and a few terminal commands — nothing more." If it still feels implausible, keep reading — the proof is hands-on and starts immediately.

2 The Operating System and the File System

Four ideas carry almost everything that follows: what an operating system does, what a file is, how folders organize files, and how paths describe locations. These ideas appear constantly in Python work, so we take them slowly and concretely.

2.1 What Is an Operating System?

A computer contains hardware: a processor, memory, storage, a screen, a keyboard, and network interfaces. The **operating system** is the layer of software that sits between that hardware and the applications you use. Windows, macOS, and Linux are all operating systems.

The operating system quietly handles tasks such as storing and retrieving files, launching programs, allocating memory, communicating with devices, connecting to networks, and managing users and permissions. You never have to think about how it does these things; you only need to know that it provides the environment in which everything else happens.

A light analogy helps: if the computer hardware is a building, the operating system is the infrastructure that makes the building usable — elevators, electricity, doors, room numbers, security, and utilities. Applications such as a browser, a text editor, or the Python interpreter then operate inside this environment. Figure 2.1 shows the three layers.

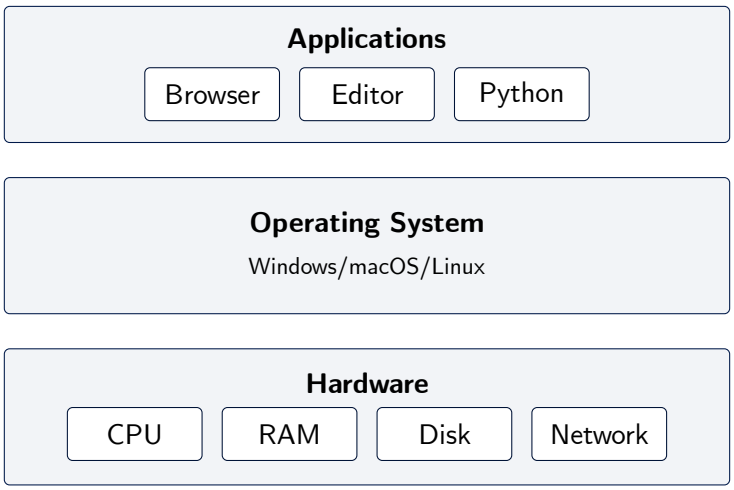

Figure 2.1: The operating system sits between the hardware and the applications you use.

Keep this picture in mind. It will reappear when we look at the terminal, which is simply another application running inside the same environment.

2.2 Files: How Computers Store Your Work

A **file** is a named collection of information stored on a computer. A photograph, a PDF, a Word document, a spreadsheet, a Python program, a text document, and a JSON data file are all files.

For practical purposes, think of a file in terms of three things, summarized in Figure 2.2: its filename, its location, and its contents. Many filenames also include an extension such as `.txt`, `.py`, or `.pdf`. Consider:

```
analysis.py
```

Here the filename is `analysis.py`: `analysis` is the main part, and `.py` is the **filename extension**, a suffix of the filename that helps humans and software recognize what kind of file it is. Operating systems sometimes hide extensions by default; we will change that setting soon, because seeing extensions is genuinely useful when learning programming.

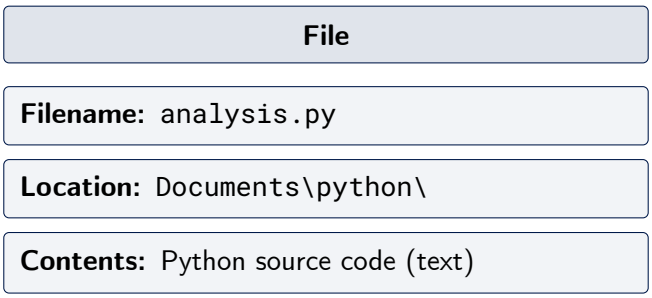

Figure 2.2: A file has a filename, a location, and contents; many filenames include an extension such as `.py`.

An Important Beginner Concept

A file is not "inside Word," "inside Python," or "inside Excel." A file exists independently on the computer's storage, and different applications can open or manipulate the same file. This distinction matters later, when the same `.csv` file might be inspected in a text editor, opened in Excel, and read by a Python program.

2.3 Files Live in a File System

Files are organized through a **file system**: a structured way to organize files, folders, locations, and names. The familiar graphical folder view in File Explorer is simply a visual representation of the file system.

Folders can contain files, other folders, or both, which produces a hierarchy:

```
Documents\
    python\
        first_project\
            analysis.py
            data.csv
            results.txt
```

Figure 2.3 shows the same idea as a tree. The computer sits at the top; folders branch into subfolders; files live at the ends of the branches. This picture is worth a long look; it is the mental model that everything else uses.

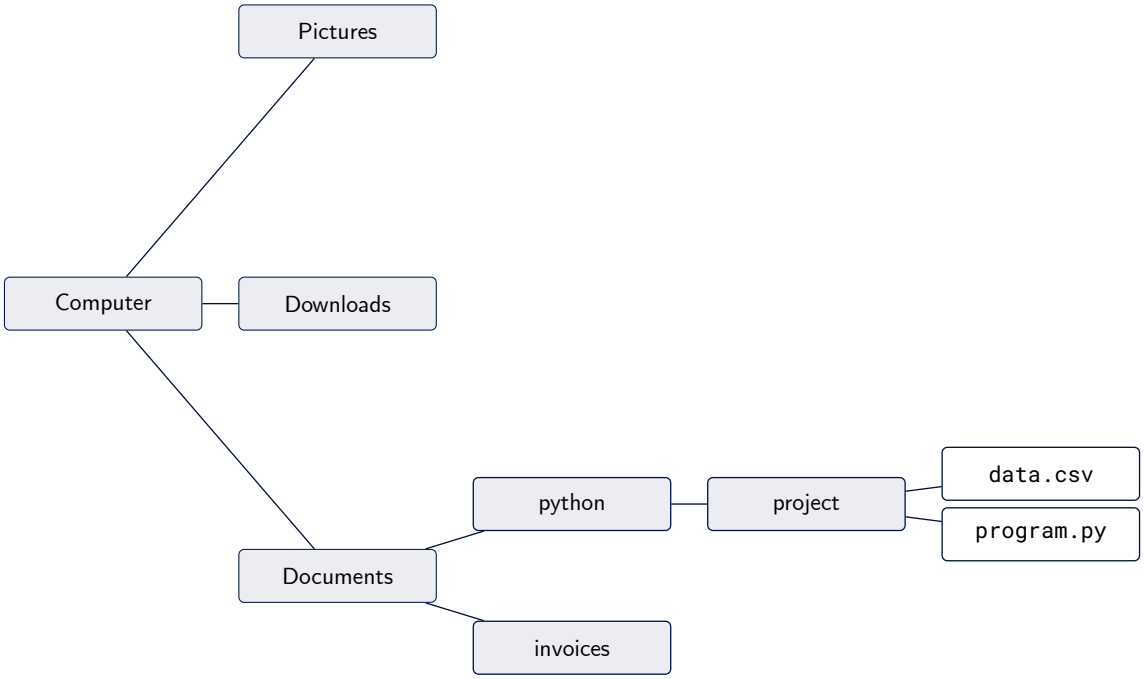

Figure 2.3: The file system organizes folders inside folders, with files at the leaves.

Folders and Directories

You will encounter two words for the same thing. Graphical interfaces usually say **folder**; programming documentation and terminals usually say **directory**. In this context they mean exactly the same thing. We use both words deliberately from here on, so that neither one surprises you later.

2.4 Paths: The Address of a File

A **path** describes where a file or folder is located within the file system. A path is the computer equivalent of a postal address: it names every stop along the route, from the top of the file system down to one specific file.

On Windows, a typical path looks like this:

```
C:\Users\yves\Documents\python\project\data.csv
```

On some Windows computers, Documents is managed by OneDrive and appears under a path such as `C:\Users \<username>\OneDrive \Documents` . Use the actual location shown by File Explorer. In the examples below, `Documents` means the Documents folder on your own computer.

You do not need to memorize path syntax. Just notice the ingredients: the leading `C:\` names the drive and marks the very top of the file system, called the **root**; the folder names along the way are separated by further `\` characters; and the last element is the filename. Figure 2.4 draws the same path as a route from the root down to one specific file.

Python programs constantly need paths to locate data or save results, which is why this concept is worth meeting now, in a calm setting.

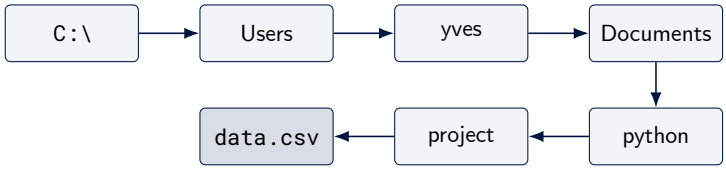

Figure 2.4: A path is the route from the top of the file system to one specific file.

Absolute and Relative Paths

You will soon hear two related terms. An **absolute path** describes the full location, starting from the root `C:\`, like the example above. A **relative path** describes a location relative to the folder currently being used:

```
data\data.csv
```

That is all you need for now. The distinction will become concrete the first time a Python program reads a data file.

2.5 File Types

Computers store many different kinds of information, and extensions are the usual shorthand for the kind at hand. Table 2.1 lists the types you will meet most often in technical training.

Table 2.1: Common file types you will encounter early on.

Extension	Typical meaning
`.jpg`	image
`.pdf`	document
`.docx`	Word document
`.xlsx`	Excel workbook
`.txt`	plain text
`.py`	Python source code
`.json`	structured text data
`.csv`	tabular text data

The important distinction for programming is between specialized document formats and plain-text-based formats. Formats such as images or Excel workbooks require software that understands their internal representation before anything sensible can be shown. Text files, by contrast, contain characters that you can usually open and inspect directly, in almost any editor, on almost any computer. Why that distinction matters so much deserves its own careful look — and gets it next.

You now have the core mental model: an operating system, a file system of folders inside folders, files with names and extensions, and paths as addresses. One kind of file matters more for programming than all others combined — the plain text file — and it earns a closer look.

Try in 60 Seconds

Open File Explorer, navigate to your `Documents` folder, and identify one file and one folder inside it. For the file, say its name, extension, and location out loud — for example, "name: report, extension: `.pdf`, location: `Documents`." That is the anatomy of a file, applied to your own computer.

3 Text Files and Text Editors

Much of programming is built on plain text files. Once that clicks, many things that look different — source code, data files, configuration, documentation — turn out to be variations of the same simple medium. That realization is worth more than any single command you could memorize.

3.1 Why Text Files Matter So Much

A remarkable amount of programming is based on **text files**: source code, configuration files, data files, markup, documentation, web pages, shell scripts, and log files are all, at bottom, text.

Text is so useful because it is readable and editable by humans, readable and generatable by programs, searchable, easy to copy, portable across operating systems, and perfectly suited to version-control systems such as Git. There is also a modern reason: text works extremely well with AI systems. When you work with an LLM, what you receive — Python code, JSON, shell commands, configuration files — is simply text, which you can paste directly into a file. Figure 3.1 puts text at the center of the map.

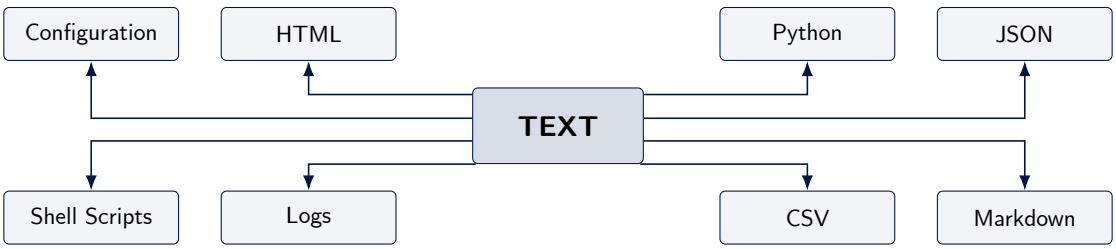

Figure 3.1: Different purposes, same fundamental medium: text.

3.2 Three Text Files That Look Different but Are the Same

Here is a small "aha" moment. Consider three files side by side.

The first is an ordinary note, `notes.txt`:

```
My first Python project
Created on Monday
```

This is simply text, with no special structure imposed by the format.

The second is a tiny Python script, `hello.py`:

```python
name = "Python"
print(f"Hello, {name}!")
```

This is also a text file. The `.py` extension signals that its text should be interpreted as Python source code, and Python itself reads this text and executes the instructions.

The third is a JSON file, `person.json`:

```
{
    "name": "Alice",
    "age": 32,
    "city": "London"
}
```

Again plain text — but the text follows the structural rules of JSON, so software can reliably interpret the contents as structured data.

The Key Insight

`notes.txt`, `hello.py`, and `person.json` are all fundamentally files containing text. What differs is the filename extension, the conventions used inside the file, and what software does with the contents. Figure 3.2 makes this visible.

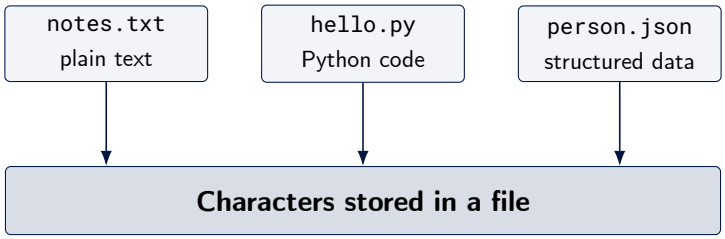

Figure 3.2: Different extensions and conventions, but all three files simply store text.

3.3 What Is a Text Editor?

A **text editor** is an application designed primarily to edit plain text. On Windows you already have Notepad; other popular choices include Visual Studio Code and Notepad++. The specific editor is not important at this stage.

A text editor is not a word processor. Microsoft Word is designed for formatted documents: fonts, page layouts, headings, embedded graphics, and printing. A text editor is designed around characters:

```
price = 100
returns = 0.05
```

Programming normally uses text editors rather than word processors, because source code must remain plain text. A word processor would invisibly add formatting information that breaks the code.

A Deliberately Simple Exercise

Do this now; it establishes the essential workflow.

1. Launch Notepad (find it via the Start menu). Notepad is always plain text; that is its whole job.
2. Type a few lines of anything.
3. Save the file as `hello.txt` in `Documents`.
4. Close the editor.
5. Find the file in File Explorer.
6. Reopen it, change the text, and save it again.

You have now performed the complete create–save–reopen–edit cycle that programming uses every day.

3.4 Applications and File Associations

Double-clicking a file usually causes the operating system to choose an application based on the file type: `report.pdf` opens in a PDF viewer, `photo.jpg` in an image viewer, `notes.txt` in a text editor. But the file itself remains separate from the application.

You can always choose a different application via *Open With* (right-click a file in File Explorer). This matters for source code in particular: the same `.py` file can be opened in a text editor, opened in an integrated development environment (IDE), or executed by Python. Those are three different operations involving one and the same file.

So far everything happened through the graphical interface: File Explorer, menus, double-clicks. There is a second interface to the very same file system — the terminal — and it is neither separate nor scary.

Try in 60 Seconds

Create a new text file containing only the line `print("Hello")` and save it as `hello.txt`. Reopen it to check the contents, then rename the file to `hello.py`. The contents did not change — only the filename did. Because the text already happens to be valid Python source code, the renamed file can now conventionally be treated as a Python script.

4 The Terminal

The terminal has an undeserved reputation for being technical and intimidating. In reality it is simply another way of interacting with the same computer you already know. A few minutes from now, you will have used it to create a real folder structure on your own machine.

4.1 The Graphical Interface and the Command Line

You already interact with the file system graphically. To reach a project folder you might open File Explorer, open `Documents`, open `python`, and open `project`. The terminal does essentially the same thing using text commands:

```
PS C:\Users\yves> cd Documents
PS C:\Users\yves\Documents> cd python
PS C:\Users\yves\Documents\python> cd project
PS C:\Users\yves\Documents\python\project>
```

Keep one message in mind: the terminal is **not a separate world**. It is another interface to the same files, the same folders, the same applications, and the same operating system, as shown in Figure 4.1.

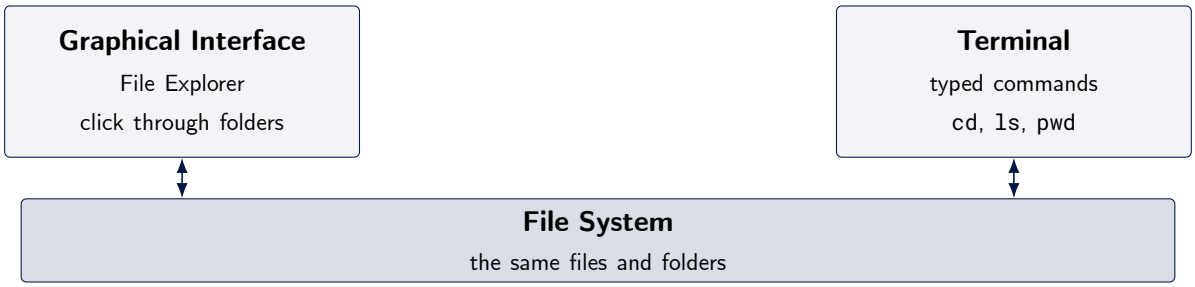

Figure 4.1: File Explorer and the terminal are two interfaces to the same file system.

4.2 What Is a Terminal?

A **terminal** is an application in which you interact with the computer by typing commands as text. On Windows you can use Windows Terminal or Windows PowerShell; this guide uses a PowerShell prompt. On Windows 10 the application may simply be listed as Windows PowerShell. You can find it in the Start menu. The underlying command environment is a *shell* called `PowerShell`. You do not need to know anything about shells beyond the word.

When you open the terminal you see a prompt:

```
PS C:\Users\yves>
```

The prompt tells you where you are: the current folder, written in Windows notation as a path starting at the `C:\` drive — here your home folder, `C:\Users \yves` . The leading `PS` marks the shell, PowerShell, and the `>` sign at the end means: *the computer is waiting for a command.* You type, press Enter, the computer responds, and a new prompt appears. That is the entire interaction pattern. Your own prompt will show your own username; the details vary, the pattern does not.

4.3 Your First Commands

You need perhaps five commands to make the terminal feel familiar. All examples here are for Windows with its default PowerShell prompt: what you type follows the `>` sign, and the computer's response appears underneath. If your prompt looks slightly different from these transcripts, that is normal — the commands and their results are what matter.

Where Am I?

`pwd` (*print working directory*) tells you which folder the terminal is currently in:

```
PS C:\Users\yves> pwd

Path
----
C:\Users\yves

PS C:\Users\yves>
```

What Is Here?

`ls` (*list*) shows the contents of the current folder:

```
PS C:\Users\yves> ls

    Directory: C:\Users\yves

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        9/14/2026   8:12 AM                Desktop
d-----        9/14/2026   7:45 AM                Documents
d-----         9/1/2026   6:03 PM                Downloads
d-----        8/30/2026   5:41 PM                Pictures

PS C:\Users\yves>
```

The classic Windows command for the same job is `dir`; PowerShell accepts both names. We use `ls` because it matches the terminal tradition you will meet in Python training.

Move Into a Folder

`cd` (*change directory*) moves the terminal into another folder. Notice how the prompt changes to reflect where you are:

```
PS C:\Users\yves> cd Documents
PS C:\Users\yves\Documents>
```

Move Up One Level

`cd ..` moves back up to the enclosing folder:

```
PS C:\Users\yves\Documents> cd ..
PS C:\Users\yves>
```

Create a Folder

`mkdir` (*make directory*) creates a new folder. PowerShell confirms the new folder with a short listing:

```
PS C:\Users\yves> mkdir first_project

    Directory: C:\Users\yves

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        9/14/2026   9:00 AM                first_project

PS C:\Users\yves>
```

Optional: Clear the Screen

`cls` tidies the terminal window without changing anything else:

```
PS C:\Users\yves> cls
PS C:\Users\yves>
```

That is the whole vocabulary for now. We deliberately skip flags, pipes, wildcards, permissions, and environment variables — none of them is needed for readiness.

4.4 A Five-Minute Terminal Exercise

Now turn the commands into a tiny success experience. The goal is to build this folder structure:

If your Documents folder is elsewhere, for example under OneDrive, use its actual path in place of `Documents` when you work through the commands below.

```
Documents\
    python\
        first_project\
```

Work through the following session, one line at a time:

```
PS C:\Users\yves> pwd

Path
----
C:\Users\yves

PS C:\Users\yves> cd Documents
PS C:\Users\yves\Documents> mkdir python

    Directory: C:\Users\yves\Documents
```

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        9/14/2026   9:02 AM                python

PS C:\Users\yves\Documents> cd python
PS C:\Users\yves\Documents\python> mkdir first_project

    Directory: C:\Users\yves\Documents\python

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        9/14/2026   9:02 AM                first_project

PS C:\Users\yves\Documents\python> cd first_project
PS C:\Users\yves\Documents\python\first_project> ls

PS C:\Users\yves\Documents\python\first_project>
```

Line by line: `pwd` confirms where you start, and the answer `C:\Users \yves` is your home folder. `cd Documents` moves into `Documents`. `mkdir python` creates the `python` folder, and `cd python` enters it. The same two steps create and enter `first_project`. Finally `ls` lists the contents — it prints nothing, which is correct for a brand-new folder.

What You Just Did

You told your computer what to do using typed commands, and it did it. That is the whole idea of the command line. Everything else you will ever type into a terminal — running Python, installing packages, using Git — is the same pattern with different words.

4.5 Graphical and Terminal Views Are the Same Reality

Now for the reinforcement step. Open File Explorer and navigate to `Documents`, then `python`, then `first_project`. The folders you created from the terminal are right there, looking completely ordinary.

This works because both interfaces manipulate the same underlying file system, as Figure 4.2 illustrates. A `mkdir` command creates a real folder on disk; File Explorer simply displays what is on disk. Nothing about the folder knows or cares which interface created it.

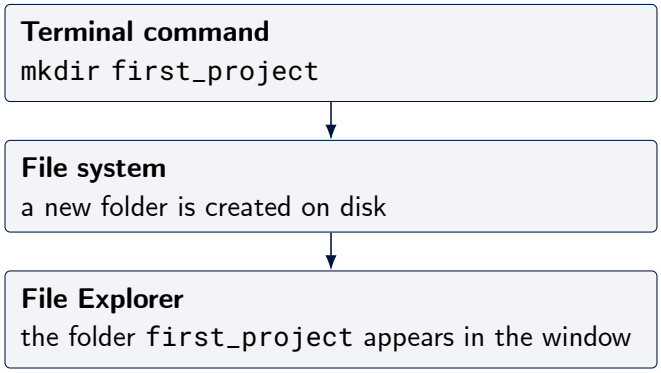

Figure 4.2: A folder created in the terminal shows up in File Explorer, because both work on the same file system.

Keep this equivalence in mind whenever a tutorial mixes terminal commands and graphical

steps. They are two views of one reality.

You can now navigate and create folders from the terminal. Time to combine everything so far — folders, text editing, file types, and terminal use — into your first small project folder.

Try in 60 Seconds

In the terminal, type `cd ..` until `pwd` shows your home folder `C:\Users \yves` , then type `ls`. You have just navigated the file system without touching the mouse. If you get lost, `pwd` always tells you where you are.

5 Your First Project

Now everything so far comes together into one concrete artifact: a small project folder containing the three representative text-file types. Along the way, a few everyday file-management habits make the difference between calm technical work and chaos.

5.1 Create Your First Project

In programming, the word **project** is used casually: a project is often simply a folder containing related files. Nothing more formal is implied. Create the following structure, using either the terminal or File Explorer for the folders and a text editor for the files:

```
first_project\
    README.txt
    hello.py
    person.json
```

If you completed the five-minute terminal exercise, your `Documents\python` folder already contains `first_project` — use that one. The three files go inside it.

The first file, `README.txt`, is a plain note:

```
My First Project
================

This is my first small project folder.
```

The second, `hello.py`, is a Python script:

```
name = "World"
print(f"Hello, {name}!")
```

You do not need Python installed for this, and you do not need to run the file. It is enough to know what it would do: define a piece of text and print a greeting containing it.

The third, `person.json`, is structured data as text:

```
{
    "name": "Alice",
    "city": "London"
}
```

You have now manually created the three representative text-file types you met earlier: a note, source code, and structured data.

5.2 File Extensions Matter

Here is a classic beginner problem. You intend to create a file called `hello.py` but actually end up with `hello.py.txt` — and everything looks fine, because Windows hides filename extensions by default. The two names are not equivalent: the first filename identifies the file conventionally as Python source code, the second identifies it as a `.txt` file whose name happens to contain `.py`.

The fix is a one-time configuration change, and one of the few settings we explicitly recommend. In File Explorer, open *View* → *Show* → *File name extensions* (Windows 11) or select the checkbox *View* → *File name extensions* (Windows 10). From now on you will always see the real, full filename.

Whenever you save a file for programming purposes, glance at the name and confirm that the extension is exactly what you intended.

5.3 Saving Files Where You Can Find Them

Beginners frequently save files without noticing where they are being stored, and then lose twenty minutes hunting for their own work. The cure is a deliberate structure. The exact layout does not matter; something like this is plenty:

```
Documents\
    python\
        exercises\
        projects\
        data\
```

The important habit is simply: *know where your files are.* Every time you save, pay attention to three things — the filename, the extension, and the destination folder. Those three seconds of attention save hours over a training program.

5.4 Copy, Move, Rename, Delete

Four operations cover most day-to-day file housekeeping, and File Explorer is entirely sufficient for all of them:

- **copy** — duplicate a file, for example before modifying an example,
- **move** — relocate a file, for example a downloaded dataset into your project,
- **rename** — give a script a clearer name,
- **delete** — remove obsolete files.

In programming you will duplicate example files, rename scripts, reorganize project folders, move downloaded datasets, and delete obsolete files constantly. This is basic housekeeping, not programming — but doing it confidently makes the programming parts much calmer.

5.5 Downloads and ZIP Files

Two practical interactions come up in every course. The first is the `Downloads` folder, where your browser puts everything it fetches. Know how to find a downloaded file there, recognize its filename and extension, and move it to where it belongs.

The second is the ZIP archive. A file like `project.zip` packages one or more files and folders into a single compressed archive, which is why courses and colleagues distribute material as ZIP files. The required skill is simple: download the ZIP, locate it in `Downloads`, right-click it, choose *Extract All*, confirm the destination, and inspect the resulting folder. On Windows the extracted folder sits next to the archive, and the folder is what you actually work with.

You can now create, name, find, and organize files — the complete practical toolkit. The natural next questions: what actually happens when a program runs, and do notebooks and cloud environments change any of it? The answers are reassuring.

Try in 60 Seconds

In File Explorer, copy `hello.txt` (select it, press `Ctrl-C` then `Ctrl-V`), rename the copy to `backup.txt` (press `F2`), and move it into your `first_project` folder. Copy, rename, move — three of the four housekeeping operations in under a minute.

6 Programs, Notebooks, and the Cloud

Everything you have learned so far — files, folders, text, paths — remains the foundation when programs run, when notebooks appear, and when work moves into the cloud. No Python yet; just a look at why your new skills keep working.

6.1 What Happens When You Run a Program?

Take the tiny script you have already met:

```
print("Hello")
```

Running it involves five steps. First, the source code is stored in a text file, say `hello.py`. Second, Python reads that text. Third, Python interprets the instructions it finds. Fourth, the computer performs the requested operations. Fifth, output appears. Figure 6.1 shows the chain.

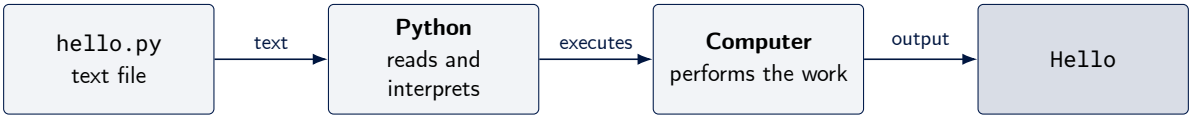

Figure 6.1: Running a program: Python reads the text file and the computer carries out the instructions.

Notice what is *not* in this picture: no compilation ceremony, no special file format, no magic. A text file goes in, Python interprets it, and text comes out. This is the bridge from computer skills to programming.

6.2 What About Jupyter Notebooks?

Early in *The Data Scientist* you will meet **Jupyter notebooks**. A notebook provides an interactive way to combine code, text, output, and graphics in one document, usually inside the browser. Instead of running a whole script at once, you work cell by cell and see results immediately.

Underneath, the same fundamental concepts remain: notebooks are files, they live in folders, they contain text, and they execute programs. Your *The Data Scientist* teaches notebooks properly; for now it is enough that the word no longer sounds alien.

6.3 What About Google Colab and Cloud Environments?

Many courses, including parts of *The Data Scientist*, use Google Colab or similar browser-based environments. It would be easy to conclude that file-system knowledge is obsolete there. It is not.

Even when the computer doing the work sits in a data center, the same ideas apply: files exist, directories exist, paths exist, source code is text, commands can be entered, and programs read and write data. The environment runs on another machine, but the concepts transfer essentially unchanged — which is exactly why they are worth learning once, properly.

6.4 What Is the Command Line Actually Good For?

As motivation, here are commands you will type later in your training. You are **not** expected to understand them now; they are examples of future use, nothing more. Run a Python program, install a software package, check the state of a Git repository:

```
PS C:\Users\yves\Documents\python\first_project> python analysis.py
PS C:\Users\yves\Documents\python\first_project> pip install pandas
PS C:\Users\yves\Documents\python\first_project> git status
PS C:\Users\yves\Documents\python\first_project>
```

The pattern should look familiar: a prompt, a word, maybe a few more words, Enter. You already did this with `pwd`, `ls`, `cd`, and `mkdir`. The important message is that you do not need to know these commands now; you merely need to be comfortable with the idea of entering commands into a terminal — and after your five-minute exercise, you are.

The concepts are all in place. What remains is confidence: making peace with errors, recruiting an LLM as a technical companion, and checking off the concrete skills that make you ready to begin.

Try in 60 Seconds

Look at the three future commands above once more and, for each, say what you think its first word refers to — `python`, `pip`, `git`. Three programs, invoked by name from the terminal. That observation alone puts the command line in its proper place: a way to start programs.

7 Readiness

Knowledge becomes readiness through a few last steps: making peace with the small mistakes everyone makes, recruiting an LLM as a technical companion, ignoring what can safely be ignored, and proving the essentials to yourself with a checklist and a ten-minute practical test.

7.1 Errors Are Normal

Type a command that cannot work and the terminal simply reports the problem and shows the next prompt:

```
PS C:\Users\yves> cd nonexistent_folder
cd : Cannot find path 'C:\Users\yves\nonexistent_folder'
because it does not exist.
PS C:\Users\yves>
```

Mistype a filename and you will get a similar message. Try to open something that does not exist and, again, the computer reports the problem. An error message is usually just information from the computer — not a verdict on you, and rarely a sign that anything is broken. The exact wording can differ slightly between PowerShell versions; the pattern of a short description followed by a new prompt does not.

When an error appears, follow a calm routine: read the message, look at the command you typed, check the spelling and the path, try again, and if it still fails, ask an LLM. That routine is not a beginner workaround; it is what experienced programmers do, quickly and without emotion.

A Working Principle

Technical work consists partly of telling computers what to do and partly of understanding what they tell you back. Error messages are the second half of that conversation.

7.2 Your LLM Is a Technical Companion

You no longer need to memorize every command. What you need is enough terminology to formulate useful questions — and you now have exactly that vocabulary. Questions like these get excellent answers from any modern LLM:

- How do I create a folder from the Windows terminal?
- What does “current working directory” mean?
- How do I show filename extensions in Windows File Explorer?
- Why does this command say “Cannot find path”?

- Explain the difference between a folder and a directory.

Without terminology, it is difficult to ask precise questions. With basic concepts such as *file*, *directory*, *path*, *terminal*, *command*, and *extension*, an LLM can fill in almost any missing detail on demand. Learn the concepts; look up the details when needed.

7.3 Things You Explicitly Do Not Need to Know Yet

To further lower the pressure: before beginning *The Data Scientist* or the *Certificate in Python for Finance*, it is **not** necessary to understand Python syntax, object-oriented programming, algorithms, Git, GitHub, package managers, virtual environments, Docker, Linux administration, cloud computing, APIs, databases, machine learning, mathematics beyond the program's stated requirements, or command-line scripting.

Some of these will appear later in your training, at the moment they become useful. Others you may never need. None of them is an entry requirement. And because *The AI Engineer* and *The Crypto Engineer* build on Python and data-science fluency at the level of *The Data Scientist*, the same minimal foundation serves all four programs.

7.4 The Minimum Skills Checklist

Here is the practical summary of this entire guide. Read each statement and check whether you can honestly say yes.

Basic Computer Use

- I can launch an application.
- I can use File Explorer.
- I can create a folder and a subfolder.
- I can rename, move, copy, and delete a file.
- I can locate something I downloaded.

Files

- I understand that files have names and locations.
- I know what a filename extension is.
- I can distinguish `.txt`, `.py`, `.json`, `.csv`, and `.pdf` at a high level.
- I understand that many programming-related files are simply text files.

Text Editing

- I can open a text editor.
- I can type text and save the file.
- I can reopen the file and modify it.

Terminal

- I can open the Terminal application and get a PowerShell prompt.
- I am comfortable typing a command and pressing Enter.
- I can determine my current directory with `pwd`.
- I can list files with `ls`.
- I can change directory with `cd`.
- I can create a directory with `mkdir`.

If these statements are broadly true, you know enough about computers to start learning Python.

7.5 The Ten-Minute Readiness Test

Finish with a compact practical self-test. Do not worry about speed; if you get stuck, ask an LLM and continue. The task is to create the following structure:

```
Documents\
    python_readiness\
        hello.txt
        hello.py
```

Work through these steps:

1. Create the `python_readiness` folder inside `Documents`, using the terminal or File Explorer.
2. Open a text editor.
3. Create the file `hello.txt` and type one line into it: `Hello from my first project.` Then save it inside `python_readiness`.
4. Create `hello.py` containing the line `print("Hello from Python")` and save it in the same folder.
5. Open the terminal.
6. Navigate to `python_readiness` with `cd`.
7. List the files with `ls`.
8. Verify that both files appear.

Optionally, open the same folder in File Explorer and confirm that the same two files are visible there — the two interfaces showing one reality, as always.

The Result

If you can complete this exercise — even with help from an LLM — you possess all the basic computer skills this guide assumes you need to begin *The Data Scientist*.

For more repetitions of exactly these moves, Chapter D offers five short workouts that train each skill in isolation, plus a bonus workout for the curious.

The final chapter is optional in the best sense: it offers a slightly deeper look at a few concepts for curious readers, and then closes the guide with the transition into Python itself.

Try in 60 Seconds

Pick one item from the checklist that felt least certain and redo it once, right now. Sixty seconds of repetition is worth more than another page of reading.

8 Looking One Level Deeper

A handful of terms you will soon hear, two optional excursions for a deeper appreciation of text and data, and then the handover to Python: that is all that remains. Nothing here is required knowledge — familiarity is the whole point.

8.1 Concepts You Will Soon Encounter

Each of the following terms gets one or two sentences, deliberately. You do not need to master any of them; having heard them is enough.

Current working directory
: The folder the terminal or a program is currently "working in." You met it as the answer to `pwd`.

Executable/program
: Something the operating system can run, such as Terminal, Python, or your browser.

Interpreter
: Software that reads and executes programs written in a language such as Python. Unlike languages with a separate user-visible compilation step, Python normally lets you run source code directly.

Source code
: Human-readable instructions written by a programmer — the text inside files such as `hello.py`.

Data file
: A file primarily containing information rather than instructions, such as `data.csv`.

Configuration file
: A text file that tells software how it should behave.

Environment
: A broad term for the system and settings under which software runs. You will hear about "virtual environments" early in your Python training.

8.2 Optional Excursion: Everything Is Ultimately Data

Computers fundamentally store information numerically. Text, images, audio, Python code, and JSON are all represented internally as data. For text specifically, Unicode assigns numbers called code points to characters, and encodings such as UTF-8 represent those code points as bytes.

The conceptual chain looks like this:

```
"Hello"  ->  characters  ->  UTF-8 encoding  ->  bytes  ->  storage
```

You do not need binary arithmetic or the details of Unicode. The takeaway is simply that the friendly text in your editor and the numbers on the disk are two views of the same thing — another instance of the theme running through everything you have practiced: two views of one reality. If you would like to see this chain worked out byte by byte for a concrete file, Chapter C walks through exactly that.

8.3 Optional Excursion: Why Programmers Like Plain Text

Programming culture revolves around plain text for good reasons. Plain text is simple, transparent, portable, searchable, easy to generate, easy to automate, and equally easy for humans and machines to process. Small changes between two versions of a text file can be compared line by line, which is why version-control tools such as Git are exceptionally effective: source code is text.

Modern LLMs add a new twist to this old story. They naturally produce and manipulate text, which makes text-based development workflows — code, configuration, documentation, data — particularly powerful in an AI-assisted world. The medium you have been practicing is the medium of both traditional and AI-assisted programming.

8.4 From Here to Python

Stepping back, you now understand the full chain shown in Figure 8.1: the operating system hosts a file system; the file system holds folders and files; many of those files are text; some of that text is source code; the terminal lets you act on all of it; and Python is where instructions turn into results.

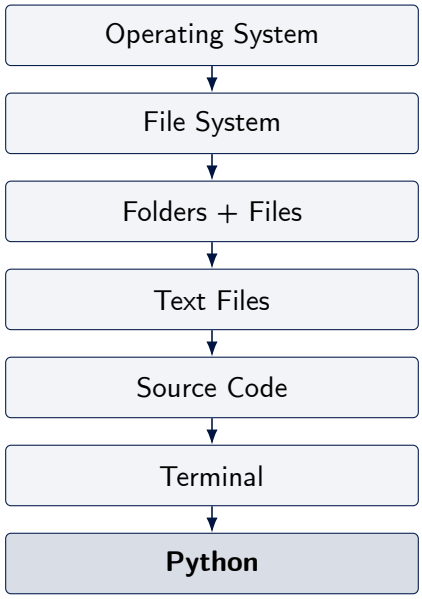

Figure 8.1: Everything you have practiced leads to one starting point: Python.

The next step is no longer learning how the computer itself is organized. The next step is learning how to give it useful instructions. That is where Python begins.

8.5 Final Message

Learning Python does not require becoming a computer expert first. You do not need extensive technical preparation. If you can organize files, edit text, and enter a few terminal commands,

you have the technical foundation needed to begin.

Everything else can be learned progressively. And whenever terminology or commands are unfamiliar, modern tools — including LLMs — make it easier than ever to obtain explanations exactly when they are needed.

Welcome to the starting line. The bare minimum turns out to be enough.

8.6 Where We Are Heading Next

Onward to *The Data Scientist* (`https://thedatascientist.dev`) or the *Certificate in Python for Finance* (`https://cpfprogram.com`). If *The AI Engineer* or *The Crypto Engineer* is your destination, the path runs through *The Data Scientist* first — and you are ready for it. If you work on both Windows and macOS, the companion *macOS Edition* translates every exercise to that platform.

Try in 60 Seconds

Close this PDF, open your terminal, and run `pwd` and `ls` once more — this time from memory. If that felt ordinary, you are ready.

A Windows Quick Reference

The Windows-specific facts in one place, condensed into Table A.1 — a single reference table meant for looking things up, not for reading.

Table A.1: Windows quick reference for everyday file and terminal skills.

Task	**Windows**
File manager	File Explorer
Basic text editor	Notepad
Terminal application	Windows Terminal or PowerShell
Home folder	`C:\Users \<username>`
Show current location	`pwd`
List files	`ls` (or `dir`)
Change directory	`cd folder`
Go up one level	`cd ..`
Go home	`cd ~`
Create directory	`mkdir folder`
Clear terminal	`cls`
Show all extensions	File Explorer → View → Show → File name extensions (Windows 11); on Windows 10, View → File name extensions

The miniature map in Figure A.1 recalls where the folders you created live.

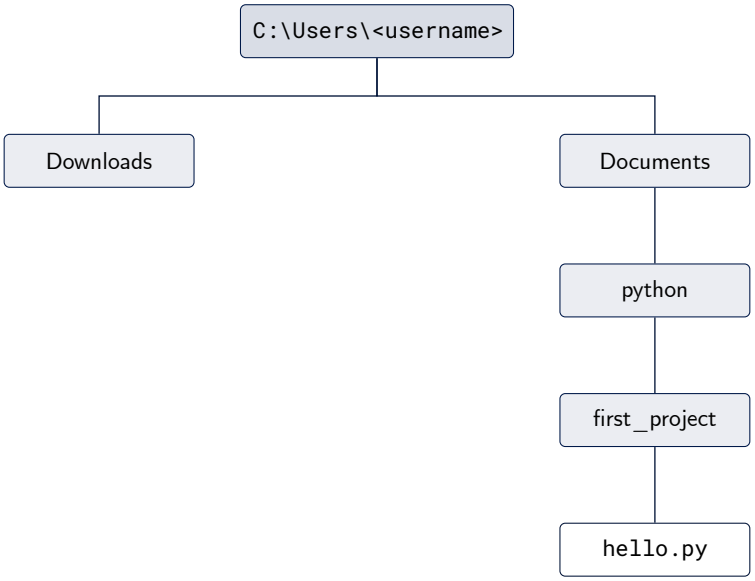

Figure A.1: Your home folder and the folders you created along the way.

B Vocabulary Cheat Sheet

Here is the key vocabulary in alphabetical order. The definitions deliberately restate the main ideas in compressed form, so that every term gets a second, independent explanation path.

Application A program used to perform a task, such as File Explorer, Notepad, or Terminal. Applications run on top of the operating system.

Command A textual instruction entered into a terminal, such as `ls` or `cd Documents`. You type it, press Enter, and the computer responds.

Directory Another word for folder. Graphical interfaces say *folder*; terminals and programming documentation usually say *directory*. They mean the same thing.

Extension The suffix of a filename, such as `.txt` or `.py`, that indicates what kind of file it is. Enable *File name extensions* in File Explorer to always see it.

File A named collection of stored information with a filename, a location, and contents. Many filenames include an extension that indicates the conventional file type. Files exist independently of the applications that open them.

File system The structure an operating system uses to organize files and directories: folders inside folders, with files at the leaves. File Explorer and the terminal are two views of the same file system.

Folder A container used to organize files and other folders. See *directory*.

Operating system The core software environment managing a computer — storing files, launching programs, and connecting devices. Windows, macOS, and Linux are operating systems.

Path A description of where a file or directory is located within the file system, for example `C:\Users \yves \Documents` followed by the filename. A path is the postal address of a file.

Plain text Text stored without document-specific formatting: just characters. Source code, JSON, CSV, and configuration files are all plain text.

Python script A text file containing Python source code, conventionally named with the `.py` extension. Python reads the text and executes the instructions.

Shell The command environment running inside a terminal — on Windows, usually `PowerShell`. It interprets the commands you type.

Terminal An application for interacting with the computer through typed commands. It is another interface to the same files and folders you see in File Explorer.

Working directory The directory in which a terminal or program is currently operating; the answer you get when you type `pwd`.

C How a Text File Is Really Stored

Welcome to an optional excursion, one level deeper than everything before. What does it *actually* mean that a text file is stored on disk? We follow one tiny, concrete file all the way from characters to bytes on storage and back. Nothing here is required for starting Python — but every step is fully understandable, and understanding it removes the last bit of magic from the file system.

C.1 The Example File

Take the smallest useful example: a file called `hello.txt`, saved in `Documents` with a text editor, containing exactly five characters and nothing else:

```
Hello
```

No title, no formatting, no invisible extras. Just the five characters `H`, `e`, `l`, `l`, `o`. The question is: what is physically stored on the disk when you press Save?

C.2 Step 1: Characters Become Numbers

Computers store numbers, not characters. Bridging the two takes two agreements. First, **Unicode** assigns a number, called a *code point*, to every character. Second, **UTF-8**, the encoding used almost everywhere today, defines how those numbers are represented as bytes. For ordinary English letters, each character uses exactly one byte, and the value agrees with the older ASCII table.

Table C.1: The five characters of `hello.txt` as Unicode code points and as UTF-8 bytes in hexadecimal notation.

Character	Code point (decimal)	Byte (hexadecimal)	Byte (binary)
`H`	72	48	01001000
`e`	101	65	01100101
`l`	108	6C	01101100
`l`	108	6C	01101100
`o`	111	6F	01101111

C.3 Step 2: Numbers Become Bytes

For these five characters, each Unicode code point is encoded as one **byte** in UTF-8: a group of eight bits, each bit being a 0 or a 1, giving 256 possible values per byte. Because long rows of binary digits are tedious to read, programmers normally write bytes in **hexadecimal** (base sixteen), where the digits run `0` to `9` and then `A` to `F`. One byte is always exactly two hexadecimal

digits, so the number 72 becomes `48` in hexadecimal. Table C.1 shows all three views side by side; you only ever need to recognize the hexadecimal one.

So the contents of `hello.txt`, as actually stored, are five bytes:

```
48 65 6C 6C 6F
```

That is the whole file. Five bytes.

C.4 Step 3: Bytes Are Written into a Block

Storage devices are organized in fixed-size **blocks**, typically 4096 bytes each. A file's bytes are written into one or more of these blocks. Our five bytes occupy only the very beginning of one block; the rest of the block stays unused. Figure C.1 shows the situation.

Modern file systems such as NTFS are more sophisticated internally — with extents, caching, and compression — but this simplified model captures the idea you need: the file system associates a filename and some bookkeeping with the storage locations that contain the file's bytes.

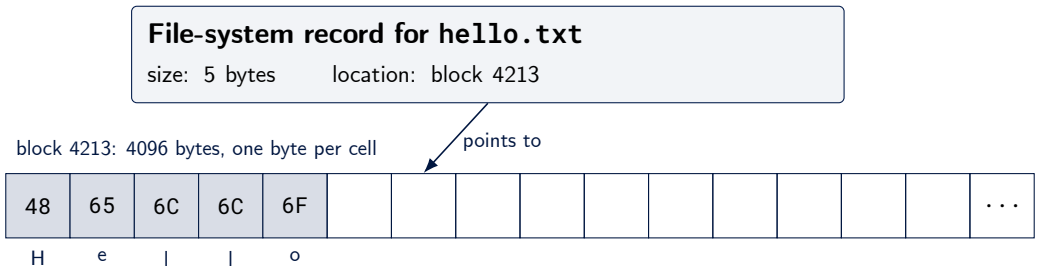

Figure C.1: The file system's record links the name `hello.txt` to a storage block; the block holds the five bytes of `Hello` (shown in hexadecimal) at its beginning.

C.5 Step 4: The File System Keeps the Bookkeeping

How does the computer later find those five bytes? The file system keeps an administrative record for every file — separate from the file's contents — noting at least the name, the size, and which blocks hold the data. The folder `Documents` is itself little more than a list that connects the name `hello.txt` to this record.

The Deepest Insight of All

Nothing inside the file says that it is "text." The five bytes `48 65 6C 6C 6F` carry no type information at all. The `.txt` extension is only a naming convention for humans and for default applications. A text file is simply bytes plus an agreed character encoding — UTF-8 in our example.

C.6 Step 5: Reading the File Back

Opening the file reverses the journey. You double-click `hello.txt` in File Explorer. The operating system looks up the name in the folder's listing, finds the record, learns the size and the block location, and reads the five bytes from the block. The text editor then decodes the bytes using UTF-8 — `48` becomes `H`, `65` becomes `e`, and so on — and you see `Hello` on the screen. Save, store, look up, read, decode: that is the complete life of a text file.

C.7 Looking at the Bytes Yourself

You can inspect the real bytes with one terminal command. Navigate to the folder containing `hello.txt` and run `Format-Hex`:

```
PS C:\Users\yves\Documents> dir hello.txt

    Directory: C:\Users\yves\Documents

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        9/14/2026   9:23 AM              5 hello.txt

PS C:\Users\yves\Documents> Format-Hex -Path hello.txt

   FilePath: C:\Users\yves\Documents\hello.txt

          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   48 65 6C 6C 6F                                   Hello

PS C:\Users\yves\Documents>
```

Two things to notice. First, `dir` reports the size (`Length`) as `5` — five bytes, exactly as expected. Second, `Format-Hex` shows three things: the position in the file (left, in hexadecimal), the bytes themselves (middle, in hexadecimal), and the bytes decoded as characters (right). The position column starts at `00000000`, because the first byte is at position zero. Your five characters really are the five bytes from Table C.1. The exact layout of the output can differ slightly between PowerShell versions; the three parts are what matter.

Try It Yourself

Save `Hello` as `hello.txt` with your text editor, open the terminal, `cd` into the folder, and run `Format-Hex -Path hello.txt`. Then change the file to `Hello!`, save, and run the command again — a sixth byte, `21`, appears. You are watching text become bytes. If you pressed Enter after typing `Hello`, you will also see two extra bytes, `0D 0A` — the invisible characters that Windows uses to mark the end of a line. A bonus discovery, not an error.

C.8 Why This Matters

None of this changes what you need to know. But the last piece of mystery is now a mechanism: a text file is bytes on a block, plus a record that remembers where they are, plus an encoding that turns them back into characters. That is why any editor can open any text file, why `hello.py` and `hello.txt` are the same kind of object, and why renaming a file changes its name but never its contents. One level deeper — and still completely understandable.

D Readiness Gym

Skills become reliable only through repetition. The five workouts here are short, concrete practice loops for exactly the moves this guide has taught: navigating folders, handling text files, keeping extensions honest, managing downloads, and rebuilding a small project from memory. Each workout takes five to ten minutes and needs nothing but your Windows PC.

Every workout states a goal, the steps, and a success check that lets you verify the result yourself. Solutions are deliberately not included: if you get stuck, re-read the linked section or ask an LLM, then continue. Struggling briefly and recovering on your own is precisely the training effect you are after.

D.1 Workout 1: The Navigation Drill

Goal. Move around your file system in the terminal without getting lost.
Steps. Work entirely in the terminal:

1. Open the terminal and run `pwd` to see where you are.
2. Create a folder `gym` inside `Documents`.
3. Inside `gym`, create three folders: `one`, `two`, and `three`.
4. Enter `one`, confirm your location with `pwd`, go back up, then visit `two` and `three` the same way.
5. From inside `three`, return to your home folder with a single command.

Success check. Running `ls` inside `gym` shows exactly `one`, `two`, and `three` — and `pwd` at the very end prints your home folder, `C:\Users \<username>`.
Hint. Whenever you feel lost, `pwd` tells you where you are, `cd ..` moves one level up, and `cd ~` brings you straight home.
Review if needed. Section 4.3 and Section 4.4 for the commands, Section 2.4 for what the addresses mean.

D.2 Workout 2: The Editor Loop

Goal. Run the create, save, close, reopen, and edit cycle with a plain text file — the loop you will repeat hundreds of times in a training program.
Steps. Work in your editor and in File Explorer:

1. Open Notepad. It is always plain text; that is its whole job.
2. Type one line: `Practice makes permanent.`
3. Save the file as `notes.txt` inside `Documents\gym` .
4. Close the editor window completely.

5. Reopen `notes.txt` from File Explorer, add a second line, and save again.

Success check. In the terminal, `ls` inside `Documents\gym` lists `notes.txt`, and reopening the file shows both lines.
Hint. Notepad needs no configuration. If you use a word processor such as Word instead, the saved document is not plain text; switch to Notepad.
Review if needed. Section 3.3 for the editor, Section 3.2 for why plain text is the point.

D.3 Workout 3: The Extension Detective

Goal. Prove that a file you create really carries the extension you intended — and not a hidden second one.
Steps. One setting, one file, two inspections:

1. Make sure File Explorer shows all filename extensions (the one-time setting).
2. In your editor, create a file containing the single line `print("gym")` and save it as `tiny.py` inside `Documents\gym` .
3. Read the file's full name in File Explorer.
4. List the same folder in the terminal with `ls` and compare the two names.

Success check. Both views agree: the file is called `tiny.py`, not `tiny.py.txt` or anything similar.
Hint. If the two views disagree, trust the terminal — it always shows the real, full name.
Review if needed. Section 5.2 for the extension trap, Section 4.5 for why File Explorer and the terminal show the same file.

D.4 Workout 4: The Download Tidy-Up

Goal. Take a file from the `Downloads` folder to its proper place — the most common housekeeping move in any course.
Steps. Start in your browser, finish in File Explorer:

1. Download any small file with your browser (or duplicate an existing file into `Downloads`).
2. Open `Downloads` in File Explorer and identify the file by name and extension.
3. If it is a ZIP archive, right-click it, choose *Extract All*, and work with the extracted folder from here on.
4. Move the file or folder into `Documents\gym \three` .
5. Remove the original download once it lives in its new place.

Success check. The file sits inside `Documents\gym \three` , visible both in File Explorer and via `ls`, and `Downloads` no longer contains it.
Hint. Your browser's download list has a "Show in folder" option that jumps straight to the downloaded file.
Review if needed. Section 5.5 for Downloads and ZIP files, Section 5.4 for copy, move, rename, delete.

D.5 Workout 5: The Full Rebuild

Goal. Recreate the complete project structure from the ten-minute readiness test — from memory, without looking anything up.
Steps. Use whichever tool you like, but work from memory:

1. From memory, build a folder `Documents\python _gym` containing a text file `hello.txt` and a Python file `hello.py`, each with one line of your choice.
2. Navigate to the folder in the terminal and list its contents.
3. Open the same folder in File Explorer and compare what you see.
4. Delete the whole `python_gym` folder, then build it once more, faster.

Success check. The second build takes noticeably less time than the first, both files pass the extension check from Workout 3, and File Explorer and the terminal agree on what exists.
Hint. Look up a step only after you have tried it from memory; the small struggle is what makes the skill stick.
Review if needed. Section 2.3 for the big picture, Section 4.4 for the terminal moves, and Section 5.4 for deleting safely.

D.6 Bonus Workout: The Byte Inspector

Goal. Watch your own text file appear as bytes.

Save a five-character text file with your editor and inspect its real bytes in the terminal; Section C.7 walks through the two commands involved. The success check is simple: the number of bytes matches the number of characters, and you recognize the hexadecimal values. Seeing your own file this way — and understanding what you see — is the surest sign that the file system no longer holds any mystery for you.

Ready

If all five workouts feel routine, nothing in this guide can surprise you anymore — and the basic file, editor, and terminal operations assumed in the first weeks of *The Data Scientist* or the *Certificate in Python for Finance* will no longer surprise you either.

Contact

The Bare Minimum

Windows Edition

Computer Skills You Need Before Starting The Data Scientist or the Certificate in Python for Finance

Get in touch:

linktr.ee/dyjh

thedatascientist.dev