

Python Programming Foundations for Data Science

Module 1: Python Programming Foundations

Dr. Yves J. Hilpisch¹

March 27, 2026



¹Get in touch: <https://linktr.ee/dyjh>. Web page: <https://thedata scientist.dev>. Research, structuring, drafting, and visualizations were assisted by GPT 5.x as a co-writing tool under human direction. Comments and feedback are welcome.

Preface

This first core book builds the Python habits that every later data-science module depends on. The focus is not just syntax. The focus is clear problem-solving, clean decomposition, file and text handling, and the ability to explain code confidently enough that it can become a visible artifact.

The primer [1] handles orientation: notebooks, local environments, terminal basics, Git, and the surrounding workflow. This volume, *Python Programming Foundations for Data Science* [2], moves the center of attention to the language itself. That is an important shift. The reader should no longer be asking mainly “Where do I work?” The more useful question becomes “How do I use Python to make small tasks clear, correct, and repeatable?”

Positioning

Module 1 begins where the primer ends. Setup and tooling are assumed to be stable enough that the reader can focus on Python itself: values, flow, functions, files, and small structured tasks.

That shift is worth stating plainly. The book is not trying to turn the reader into a software engineer in one jump. It is trying to make Python feel usable. Usable means that a short script, a small notebook, or a tidy function no longer looks like a wall of symbols. It starts to look like a sequence of decisions that can be followed, checked, and improved.

This book is written for absolute beginners and early-stage returners. It does not assume that confidence exists already. Instead, it tries to build confidence through repetition, clarity, and visible completion.

One theme runs through the whole module: a small finished example is worth more than a larger half-understood one. For that reason, the early chapters stay close to simple values, simple control flow, and short artifacts. The goal is not to avoid ambition forever. The goal is to create a stable base on which later ambition can rest.

By the end of the book, the reader should be able to:

- read and write small Python scripts,
- work comfortably with the main built-in data types,
- create basic reusable functions,
- handle strings and files,
- reason through small coding tasks step by step,
- and package the result in a first coherent portfolio artifact.

How to Use This Book

Treat each chapter as something to work through, not just something to read. Run the examples, change at least one input, and save the result in a notebook or script. The habit of producing small visible artifacts matters as much as the content itself.

The module also has a deliberate scale. Many examples are intentionally small: counting, cleaning a short string, scanning a list, writing a helper function, reading a text file. That is not because the ambitions are low. It is because small examples reveal structure. Later data work becomes easier when these building blocks no longer consume all of the reader's attention.

That is a meaningful outcome. It is also the right foundation for the next stage, where Python fluency becomes data fluency. The next volume, *Python Data Analysis, Visualization, and Storytelling* [3], builds directly on this foundation.

Contents

Preface	i
I Python Foundations	1
1 Introduction and Learning Goals	3
1.1 What This Module Is Trying to Build	4
1.2 Why Python Matters for Data Work	4
1.3 What Counts as a Useful Artifact	4
1.4 The Shape of the Module	5
1.5 Why the Module Starts Small	5
1.6 What Good Progress Looks Like	5
1.7 How to Work Through the Module	6
1.8 Why Simplicity Is a Strength Here	6
1.9 Where Module 1 Leads	6
1.10 Where We Are Heading Next	6
2 Core Syntax and Data Types	8
2.1 Values Come Before Workflows	9
2.2 Variables and Assignment	9
2.3 Expressions Turn Values into New Values	9
2.4 Strings, Numbers, and Booleans	9
2.4.1 Strings	10
2.4.2 Numbers	10
2.4.3 Booleans	10
2.5 Inspecting What You Have	11
2.6 Collections: Why One Value Is Often Not Enough	11
2.7 Lists	11
2.7.1 Mutability Matters	12
2.8 Tuples	12
2.9 Dictionaries	12
2.10 Sets	13
2.11 How to Choose Between the Four	13
2.12 Indexing and Slicing	14
2.13 Small Transformations Build Confidence	14
2.14 A Tiny Worked Example	14
2.15 Why This Matters for Data Science	15
2.16 Where We Are Heading Next	15
3 Control Flow and Functions	16
3.1 From Stored Values to Active Logic	17
3.2 Conditionals Express Decisions	17

3.3	Loops Express Repetition	17
3.4	Why State Matters in Loops	18
3.5	A Note on While Loops	18
3.6	Functions Package Thought	18
3.7	Parameters and Return Values	19
3.8	Printing Is Not the Same as Returning	19
3.9	Decomposition Makes Small Problems Smaller	19
3.10	A Worked Example	20
3.11	Why This Matters for Data Science	20
3.12	Where We Are Heading Next	20
4	Strings, Files, and Notebook Habits	21
4.1	Titanic as a String-Only Warmup	22
4.2	Strings Are Data Too	22
4.3	Inspecting and Cleaning Text	22
4.4	Splitting and Joining	23
4.5	Small String Methods Add Up	23
4.6	Files Turn Work into Artifacts	23
4.7	Reading a Text File	23
4.8	From File Content to Structured Pieces	24
4.9	Writing a Simple Output File	24
4.10	Paths Deserve Attention	24
4.11	A Safer Path Habit with Pathlib	25
4.12	A First Debugging Habit for File Errors	25
4.13	Notebook Habits Matter	25
4.14	When to Use a Script and When to Use a Notebook	26
4.15	A Worked Example	26
4.16	Why This Matters for Data Science	26
4.17	Where We Are Heading Next	27
II	Pythonic Problem Solving	28
5	Pythonic Thinking and Idioms	30
5.1	What Pythonic Really Means	31
5.2	Readability Is a Technical Choice	31
5.3	From Mechanical to Expressive	32
5.4	Truthiness Simplifies Common Checks	32
5.5	Enumerate Makes Position Visible	32
5.6	Plain Loop First, Idiom Second	33
5.7	Zip Connects Parallel Sequences	33
5.8	Comprehensions Compress a Familiar Loop Pattern	33
5.9	A Bad Comprehension Versus a Good Loop	34
5.10	Filtering Is Another Common Idiom	34
5.11	Sorted Is Better Than Reinventing Small Sort Logic	34
5.12	A Worked Example in Pythonic Cleanup	35
5.13	A Slightly Larger Worked Example	35
5.14	Why This Matters for Data Science	36
5.15	Where We Are Heading Next	36
6	Object-Oriented Programming Basics	37
6.1	Why OOP Appears Here at All	38

6.2	Class and Object	38
6.3	A Human Analogy	38
6.4	A First Class	39
6.5	From Dictionary to Class	39
6.6	What <code>init</code> Does	40
6.7	What <code>Self</code> Means	40
6.8	Methods Are Functions Attached to an Object	40
6.9	When a Class Helps	40
6.10	Multiple Objects from One Class	41
6.11	When a Class Does Not Help	41
6.12	A Small Worked Example	41
6.13	A Realistic Reading Goal	42
6.14	Why This Matters for Data Science	42
6.15	Where We Are Heading Next	42
7	Standard Algorithms from Computer Science	43
7.1	Why Standard Algorithms Matter	44
7.2	A Useful Analogy	44
7.3	Linear Search	44
7.4	Counting Frequencies	45
7.5	Membership Is a Related Question	45
7.6	Sorting Clarifies Structure	46
7.7	Binary Search as an Idea	46
7.8	Lookup Tables with Dictionaries	46
7.9	A First Runtime Intuition	47
7.10	Choosing the Right Move	47
7.11	A Worked Example	47
7.12	A Slightly Richer Example	47
7.13	Why This Matters for Data Science	48
7.14	Where We Are Heading Next	48
8	Capstone Brief	49
8.1	What the Capstone Should Prove	50
8.2	Recommended Scope	50
8.3	Recommended Deliverables	50
8.4	Notebook and Repository Expectations	50
8.5	If You Use a Helper Module	51
8.6	What Good Looks Like	51
8.7	A Suggested Project Shape	51
8.8	A Minimal Notebook Plus Helper Flow	52
8.9	What the Reader Should Demonstrate	52
8.10	What to Avoid	52
8.11	A Capstone Prompt	53
8.12	Why This Matters for the Portfolio	53
8.13	Where We Are Heading Next	53
	Glossary	55
	Epilogue	59

List of Figures

1.1	Module 1 moves from raw values to reusable logic and visible artifacts.	5
2.1	The four core built-in container ideas at a glance.	11
3.1	A simple problem-decomposition pattern for beginner Python tasks.	19
4.1	A beginner-friendly text and file workflow: inspect first, then clean, then save.	22
5.1	A safe beginner path into Pythonic idioms: understand the plain version first, then adopt the clearer pattern.	31
6.1	A beginner map of object-oriented basics: a class defines state and behavior, and objects are concrete instances.	38
6.2	A human analogy for classes and instances: one class describes a kind of thing, while Peter, Paul, and Mary are separate instances of that kind.	39
7.1	A small algorithm map: different question types naturally suggest different solution patterns.	44

Contact

Python Programming Foundations for Data Science

Module 1: Python Programming Foundations



Get in touch:

linktr.ee/dyjh

thedata scientist.dev