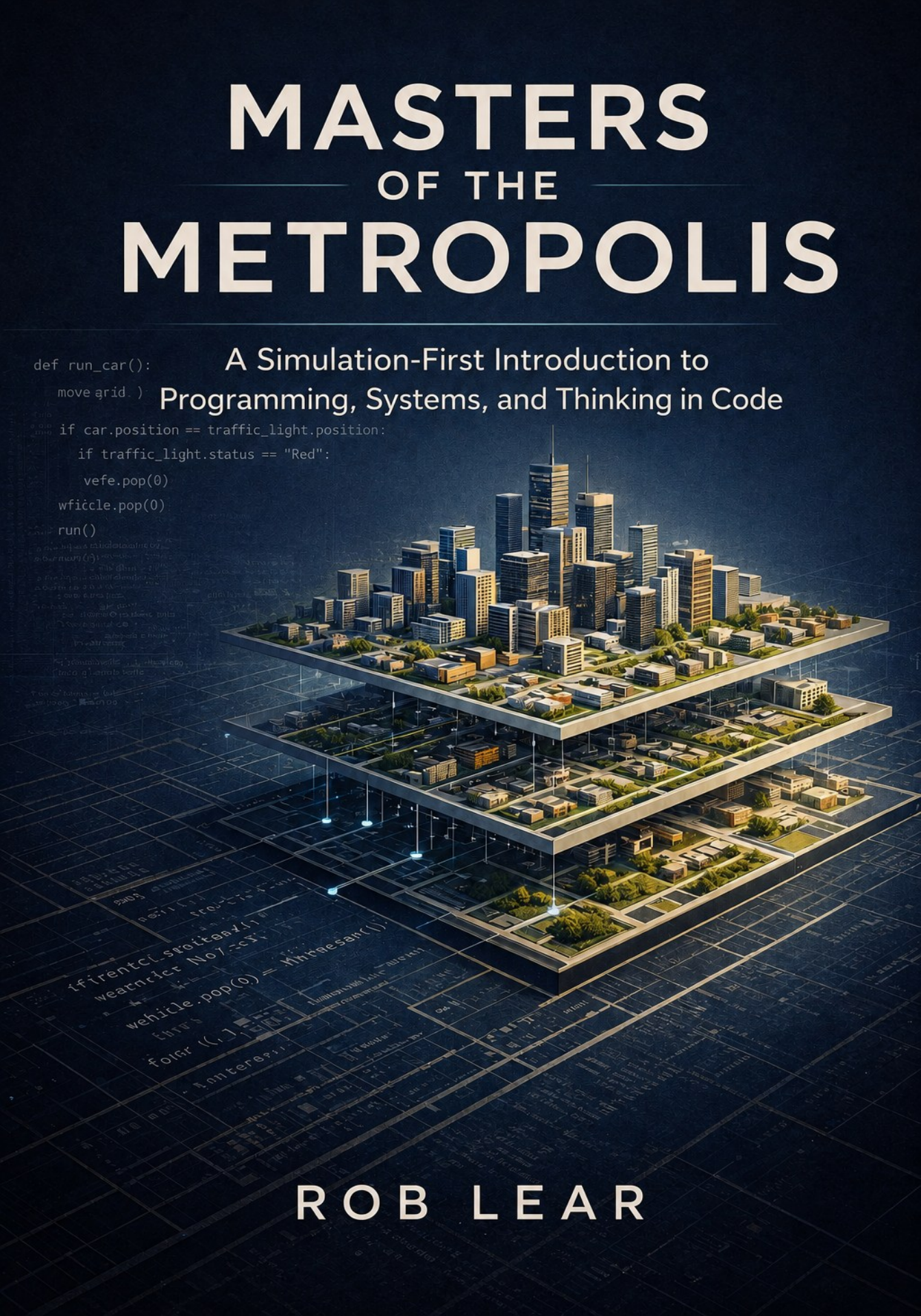
MASTERS OF THE METROPOLIS

A Simulation-First Introduction to Programming, Systems, and Thinking in Code



ROB LEAR

Masters of the Metropolis
A Simulation-First Introduction to Programming, Systems, and Thinking in Code

Rob Lear

Copyright & Legal

Disclaimer

The author is not responsible if your machine becomes sentient, achieves economic dominance, or establishes an independent space agency.

Author's Note

This book exists because learning to program is often taught backwards.

Beginners are shown syntax before meaning, tools before purpose, fragments before systems. They learn how to write lines of code long before they understand what those lines do together. As a result, many people learn to code without ever learning how to think in code.

This book takes a different approach.

Instead of starting with language features, we start with a world — a city — and grow it piece by piece. Every concept you encounter exists because the simulation needs it. Every abstraction earns its place. Complexity is not avoided; it is introduced only when it becomes unavoidable.

This is not a book about memorising Python.
It is a book about building understanding.

If you finish this book, you will not just have written a program. You will have built a system, watched it evolve, tested it, broken it, repaired it, and refactored it. You will learn that code does not exist in isolation — it interacts, accumulates, and changes over time.

That is the real lesson.

To Claire, who knows how “this” system works.

Masters of the Metropolis

TABLE OF CONTENTS

0 - The City Is the Teacher

> *"The only way to learn programing is to program. The only way to learn about complex systems is to build one."*

0.1 What This Book Is

This book will teach you Python by having you build a city simulator from scratch.

By the time you reach the final chapter, you will have a working simulation running in your terminal: buildings that age and degrade, citizens with names and jobs and morale scores, a seasonal economy where wages are paid and taxes collected, a mayor whose approval rating rises and falls with every decision, an infrastructure network of roads and utilities that slowly fails unless you maintain it, and a game layer complete with win conditions, achievements, and a City Chronicle that records the history of every run. All of it rendered as a live, colour-coded dashboard in the terminal. All of it written in Python, by you.

But that is not the whole description of what this book is. It is also a book about *how to think* when you program. About how to grow a system across many weeks without losing track of its shape. About why some code is easy to change and other code resists every modification. About how to test what you build, measure what you optimise, and read a program you wrote six months ago without needing to reconstruct it from first principles. Those lessons do not fit in a syntax reference. They have to be earned by doing something real over a long arc.

The city is the vehicle for all of that. It is the something real.

0.2 Where the Idea Came From

The inspiration for this book is a category of game that reached its peak in the 1990s and early 2000s: the management simulation, think championship manager (do a quick search if you feel the need). You do not play a character directly. You sit one level above the action, make decisions, and watch a simulation engine translate those decisions into results over time. These games are not famous for their graphics. They are famous for the way they make you care.

Picture this: you approved a new residential block in year two because the treasury looked healthy. By year four the roads serving it are at thirty percent condition, commute times have doubled, and three citizens are considering leaving. You do not blame bad luck. You trace the decision: should you have invested in road maintenance first? Was the infrastructure report clear enough? Good simulations generate consequences, and consequences generate stories, and stories generate care.

This book uses the same structure. The simulation runs forward. Events fire. Seasons change. Citizens get promoted and retire and leave if conditions are bad enough. The city accumulates a history. The economic decisions you make in Chapter 7 still matter in Chapter 14. The infrastructure you neglected in Chapter 11 starts failing in Chapter 16. Every chapter adds something, and the something it adds interacts with everything that came before.

That accumulation is intentional. Programming is taught most often as a series of isolated topics: here is a loop, here is a function, here is a class. The connections between those topics — why you would reach for a class instead of a dictionary, when a generator is better than a list, how a caching decision in one part of the code creates a correctness obligation in another — are harder to demonstrate in isolation. They require a system. The city is the system.

0.3 The Philosophy of Simulation

PHILOSOPHY: What makes a simulation a simulation

A simulation is not a model of what things are. It is a model of what things do over time. A spreadsheet can tell you that the treasury holds $50,000. A simulation tells you whether that $50,000 will still be there in three seasons, given the current maintenance costs, the current population growth rate, the current debt interest schedule, and the probability distribution of incoming crisis events.

The gap between those two statements is where simulation earns its value. Spreadsheets answer the question you asked. Simulations surface the questions you did not know to ask.

A simulation has three ingredients. It has

state — the set of values that describe the system at a given moment. The treasury balance, the health of each infrastructure node, the morale of each citizen, the day counter, the season. Everything the simulation needs to know to continue from where it is.

Rules — the logic that transforms state over time. Each day, building condition decreases by 0.1. Each season, happiness changes by the season's delta. Each tick, wages are paid before taxes are collected. Rules are deterministic: given the same state, they produce the same result. Rules are also composable: the interaction between the condition-decay rule and the maintenance-cost rule and the citizen-morale rule produces emergent behaviour that was not written anywhere explicitly.

Events — the stochastic element that makes every simulation run different. A fire breaks out. An investor arrives. A citizen gets promoted unexpectedly. Events are selected by probability, weighted by context, handled by consequence. They prevent the simulation from settling into a predictable cycle and force the player to respond to novelty.

The reason cities make good simulation subjects is that they genuinely have all three. A real city has state: population, infrastructure condition, economic health, political mood. It has rules: maintenance costs are real, population growth follows patterns, economic feedback loops are well-documented. And it has events: disasters, windfalls, elections, epidemics. The simulation is not a metaphor for city management. It is a simplified model of the actual dynamics.

Understanding simulation is also, quietly, understanding software architecture. The same three ingredients — state, rules, events — appear in every non-trivial software system. A web application has state (user records, session data), rules (authentication logic, business rules), and events (requests, webhooks, background jobs). A game has state (entity positions, health, inventory), rules (physics, collision, AI), and events (player input, timers, network messages). The mental model you build working through this book transfers.

0.4 What Simulation Teaches That Other Approaches Do Not

Most programming education works by decomposition: here is topic A, here is topic B, here are some exercises. The implicit promise is that once you have collected enough topics, you will be able to assemble them into something real. The problem is that assembly is itself a skill, and it is one that decomposition-based teaching almost never practises.

When you are three chapters into this book and your tick_day() function is not updating happiness correctly, you cannot look up “how

to debug a simulation". You have to reason about state. You have to trace what changed and when and why. You have to understand the difference between a bug that causes a crash — which announces itself — and a bug that produces the wrong numbers quietly, which requires you to know what the right numbers should be.

That is harder, and it is more useful. The debugging skill you develop on a simulation is closer to the debugging skill you will need in a professional codebase than anything you can learn from a collection of isolated exercises. Real codebases have history. They have decisions that made sense at the time and create friction now. They have interactions between components that were not designed to interact. A simulation that grows across eighteen chapters has all of that.

Simulation also makes certain abstract concepts viscerally concrete. Consider feedback loops. The maintenance cost of a building increases when its condition falls below 50%. Lower condition means higher costs; higher costs drain the treasury; a depleted treasury makes repairs harder to afford; deferred repairs accelerate condition decay. That is a negative feedback loop, and describing it in words is less instructive than watching it happen to your city while you try to stop it. The concept is not new when you encounter it in a systems-thinking textbook later. You have already felt it.

PHILOSOPHY: Emergence

The most important thing a simulation can teach you is emergence: the property of complex systems where the behaviour of the whole is qualitatively different from the behaviour of any individual part.

No single rule in this simulation says "cities in the late game tend to face infrastructure crises." That is an emergent consequence of the

interaction between the decay rate, the maintenance cost formula, the event probability distribution, and the player's natural tendency to defer maintenance when the treasury is under pressure. Emergent behaviour is not designed. It appears. The quality of a simulation is measured largely by how much interesting emergent behaviour it produces per rule written.

0.5 Who This Book Is For

This book assumes you can write a short Python script. You know what a variable is. You have seen a for loop. You may have written a function. Beyond that, everything is introduced when it is needed.

The book is not for complete beginners who have never opened a text editor. Chapter 1 moves quickly. The first code block appears on the second page.

It is also not for experienced Python developers who want a reference. It is for the reader who has done some Python and wants to do more, but finds tutorial exercises too small and professional codebases too large. The city simulator is the right size: complex enough to require real architectural thinking, small enough to hold entirely in one file for most of the book, and concrete enough that every design decision has an obvious real-world analogy.

The ideal reader is someone who has completed a Python basics course and found themselves wondering what to do next. The answer this book gives is: *build something that grows*. Not a to-do list. Not another temperature converter. A simulation with citizens who have names and morale scores and life stories, running in a live terminal dashboard, with win conditions and elections and a debt spiral if you are not careful. Something with stakes.

0.6 The Terminal as Canvas

Most Python learning material targets the web or desktop GUI frameworks. This book targets the terminal deliberately. The terminal is universal — it runs identically on macOS, Linux, and Windows without installation beyond Python itself. It is also, with the right library, genuinely expressive. The `Rich` library turns the terminal from a log into a canvas: it supports colour, bold, italic, tables, panels, progress bars, live-updating layouts, and full-screen mode. By Chapter 8, the simulation runs inside a five-region live dashboard that updates in real time: colour-coded regions, live sparklines, and keyboard controls, all rendered entirely in the terminal.

The choice of Rich is also pedagogical. When you write `console.print("[bold cyan]Treasury:[/bold cyan] $50,000")`, you are doing two things at once: learning string formatting, and producing output that looks intentional rather than incidental. The visual feedback of a well-formatted terminal panel is immediate and satisfying in a way that plain `print()` output is not. Presentation matters because it creates the sense that you are building something real, not completing an exercise.

The terminal is also honest. There is nowhere to hide. If your dashboard renders wrong, you see it immediately. If your spacing is off, the table breaks. The terminal does not smooth over errors with default styling. That honesty is a teaching asset: the feedback loop between code and result is as short as it can possibly be.

0.7 How This Book Is Structured

The book is divided into four parts. Each part corresponds to a phase in the simulation's development and a level in Python's conceptual stack.

PART I — FOUNDATIONS (Chapters 1–4)

Chapters 1 through 4 build the city's skeleton. A list of building dictionaries becomes the foundation. A Citizen class brings object-oriented programming into the picture. By Chapter 4, the terminal shows a colour-coded building registry alongside a citizen feed that updates in real time. The Python topics are: variables, types, lists, loops, functions, conditionals, dictionaries, and classes.

Every concept introduced in Part I is introduced through the city. You do not learn about dictionaries in the abstract and then apply them to buildings. You discover that a building is not a name — it is a collection of named facts — and dictionaries are the natural way to express that.

PART II — THE SIMULATION ENGINE (Chapters 5–8)

Chapters 5 through 8 add time, events, economy, and the full live dashboard. Time flows through seasons and years, triggering boundary effects at each transition. The event engine fires random occurrences drawn from a weighted catalogue. The economy processes wages, taxes, rent, and maintenance each tick. Chapter 8 assembles everything into the Championship Manager dashboard: five live regions, keyboard controls, speed modes, and the first taste of the simulation running as a real-time system.

The Python topics are: dataclasses, decorators, random.choices(), the Observer pattern, JSON-style data design, and Rich's Layout and Live systems.

PART III — DEPTH SYSTEMS (Chapters 9–16)

Chapters 9 through 16 deepen the simulation with systems that interact across time. The mayor accumulates political capital and faces elections. Citizens age through life stages, form relationships, and experience life events that emerge from probability rather than script. Save and load gives the simulation persistent memory. Buildings gain an upgrade tree with branching paths and delayed consequences. Analytics surfaces trends the player would otherwise miss. Testing and infrastructure deep-dive chapters complete the set.

The Python topics span from @classmethod constructors and @dataclass to pytest fixtures, Dijkstra's algorithm, heapq, and graph theory.

PART IV — MASTERY (Chapters 17–18)

Chapter 17 optimises what has been built: generators replace list comprehensions in hot paths, cProfile locates real bottlenecks, lru_cache and dirty-flag caching reduce redundant computation, __slots__ tightens memory, and asyncio adds cooperative multitasking for non-blocking saves.

Chapter 18 is the final project: difficulty settings, win conditions, a narrative game-over screen, achievements, and the City Chronicle — an append-only record of significant events that makes every playthrough unique.

Here is the full map of the book at a glance:

Ch	Title	What you learn	What the city gains
1	The Empty Plot	Variables, strings, integers, lists, for loops, Rich Table	City name, treasury, buildings list, styled terminal output
2	Daily Routines	Functions, parameters, return values, if/elif/else, range()	tick_day() heartbeat, building status, Rich progress bar
3	Building Profiles	Dictionaries, .get(), list-of-dicts, conditional styling	Full building registry with colour-coded Rich Table
4	Citizens	Classes, __init__, self, methods, deque, Live display	Named citizens with arcs, live citizen feed
5	Nested Time	@property, while loop, break/continue, state machines	Seasons, years, TimeState, seasonal effects, speed controls
6	Events	random.choices(), Observer pattern, EventBus, Layout	Weighted event catalogue, live two-column display

Ch	Title	What you learn	What the city gains
7	The Economy	Classes with history, sparklines, feedback loops	CityFinances, Market, wages, taxes, rent, bankruptcy
8	The Full Dashboard	Layout regions, screen=True, keypress handling	Live five-region Championship Manager dashboard
9	The Mayor's Office	@dataclass, PendingEffect, IntPrompt, elections	Political capital, mayoral actions, four-year elections
10	Citizens as Characters	Threshold tables, survivor pattern, life events, lambda	Life stages, relationships, procedural citizen arcs
11	Infrastructure & Transport	Adjacency dicts, weighted graphs, two-tier thresholds	Six utility systems, road network, commute effects
12	Save, Load & City History	json, to_dict/from_dict, @classmethod, SAVE_VERSION	Versioned saves, autosave, save browser, city chronicle
13	Building Upgrades	Inheritance, super(), __getitem__, Rich Tree	Upgrade tree, ResearchCentre, TradeHub, prestige system

Ch	Title	What you learn	What the city gains
14	Data & Analytics	Statistics from scratch, bar charts, rolling means, matplotlib	Analytics dashboard, sparklines, PNG chart export
15	Testing	pytest, fixtures, parametrize, patch(), coverage	Full test suite, conftest.py, Rich test report
16	Infrastructure Deep Dive	Dijkstra, heapq, edge conditions, congestion, repair tiers	Per-edge road health, shortest-path routing, city map
17	Optimisation & Performance	SimulationState, generators, cProfile, lru_cache, __slots__	Measurably faster simulation, asyncio autosave
18	Final Project	Integration, architecture, craftsmanship	Win conditions, difficulty, game-over, City Chronicle

0.8 How to Use This Book

There is no answer key. This is not an oversight.

Every exercise in this book asks you to build part of the city simulator. Later chapters show one way to implement what you built — not the only way. Your working implementation is the right implementation. If your code produces a running city, you are ready for the next chapter regardless of how it compares to the version shown later. The gap

between your first attempt and a cleaner subsequent version is where most of the learning happens. Treat that gap as a code review, not a correction.

The starred exercises

Each chapter ends with four exercises. The fourth is starred (★) and substantially larger than the first three — it is an integration exercise that wires the chapter's new concepts into the running simulation. The starred exercise has a **"Next chapter needs"** note at the bottom that states exactly what the following chapter requires. Everything in the starred exercise beyond that minimum is optional enrichment. Do the minimum, move forward, and come back to the enrichment when you want the practice.

The carry-forward contract

Every chapter after the first opens with a short note stating what it assumes from the previous chapter. If your implementation differs from the version shown — different field names, different class structure, different approach — that note will tell you what the new chapter actually needs so you can adapt rather than rewrite.

Three chapters — 13, 16, and 17 — are architectural pivot points where the simulation's structure changes significantly. Each of those chapters has a **"If your implementation differs"** note in the relevant section, explaining how to apply the chapter's pattern to whatever you have built rather than to the specific version shown.

Running the code

Every code block in this book is runnable. Create a file called `city.py` and add to it as you work through each chapter. The simulation is one file for most of the book — not because one file is always the right architecture, but because it is the right architecture for a growing simulation that you can read from top to bottom and understand completely. By Chapter 12 you will have a `tests/` directory and a

`saves/` directory alongside `city.py`, but the core simulation remains in a single importable module throughout.

From Chapter 1 onward, the label above each code block tells you what to do with it. TYPE THIS means add the code to the current file, usually city.py, and run it when the surrounding text says to run. EXAMPLE means read and try it if you want, but do not carry it forward unless the text says so. TERMINAL means run the command in your terminal, not inside Python. REFERENCE means keep the pattern nearby for comparison while you build the surrounding feature.

Run the file at every step. Do not write ten sections and run once. Write a section, run it, see what it produces, understand the output, then continue. The terminal is your map. *Treat it as one.*

The pace

Part I and Part II are denser than later chapters because they establish foundations everything else depends on. Part III is slower — each chapter adds one depth system to a running simulation, and the pace of new concepts decreases as the pace of integration increases. Part IV is more conceptual than constructional: you are revisiting code you already understand and making deliberate improvements.

If you finish a chapter and the concepts feel settled, move forward. If you finish a chapter and something feels uncertain, the exercises at the end are the right place to pressure-test that uncertainty rather than re-reading the section.

0.9 What You Will Need

Python 3.10 or later. Most features used in this book work on 3.8, but some syntax — particularly the structural pattern matching in later chapters and certain type annotation styles — requires 3.10. Install it from python.org or your system's package manager.

A text editor or IDE. Visual Studio Code with the Python extension is the most commonly used option and the one the book's illustrations

assume. PyCharm, Neovim, and any other editor that understands Python will also work. The specific editor does not matter. What matters is that you can run `python city.py` from a terminal — either the editor's integrated terminal or a standalone one.

The Rich library. Install it before starting Chapter 1:

TERMINAL — RUN IN TERMINAL — installing Rich

```
# Windows users Run this in your terminal — not in Python

pip install rich

# Mac users use this command

pip3 install rich

# mac users  If you see a permissions error, try:

pip3 install --user rich

# Windows users verify it installed correctly

python -c "from rich.console import Console;
Console().print('[green]Ready.[/green]')"

# Mac users verify it installed correctly

python3 -c "from rich.console import Console;
Console().print('[green]Ready.[/green]')"
```

Chapter 14 optionally uses `matplotlib` for PNG chart export. Chapter 15 uses `pytest`. Both are introduced when needed with installation instructions at the point of first use.

0.10 A Note on the City

In the walkthrough, the city is called New Pythonville. Its founding citizens are Alice Chen, Bob Marsh, Mia Torres, Tom Wick, Elena Kosta, and Sam Okafor. You can use these names exactly if you want a

guided path through the book, or replace them when the code block tells you a name is yours to choose. Tom is 63 in Chapter 4. By Chapter 10 he retires. By Chapter 12 his retirement is recorded in the City Chronicle. By Chapter 14 a sparkline shows the morale trend across the years he worked.

They are not real people. They are named variables with richer state than most named variables. But naming matters. When the simulation surfaces that Elena could not cover her rent last Tuesday, you notice in a way you would not if the output said "Citizen 4 rent shortfall." Names create the minimum viable fiction required for a simulation to generate the feeling of consequence. And consequence is what makes a simulation worth building.

The city you build can also be called whatever you call it. Its citizens can have whatever names you give them. If you feel more comfortable following the book exactly, keep New Pythonville and the founding cast. If you want to make the city your own, change the names where the code invites you to do so. The simulation does not enforce New Pythonville. That is the point: the systems you build are yours, and the city that runs inside them will have its own history, its own crises, its own moments worth remembering.

0.11 What You Will Have at the End

A Python program you wrote, from first variable to final test. A simulation that runs in real time in your terminal, with a live dashboard that would look at home in a 1990s game developer's portfolio and an architecture that would not embarrass a 2020s software engineer.

More importantly: a mental model of how complex systems are built. Not by designing the whole thing upfront and implementing it — nobody does that, and it does not work. By starting with the minimum viable piece, making it run, then extending it one well-chosen layer at

a time until it becomes something you could not have designed at the start.

That is how the best simulations are built. It is how most good software is built. It is also, it turns out, how most good cities are built — one plot of land, one decision, at a time.

Yours starts in Chapter 1.

QUICK START — BEFORE CHAPTER 1

1. Install Python 3.10 or later from python.org
2. Install Rich: `pip install rich` or `pip3 install rich`
3. Create a file called `city.py` in a new folder
4. Open a terminal in that folder
5. Turn to Chapter 1