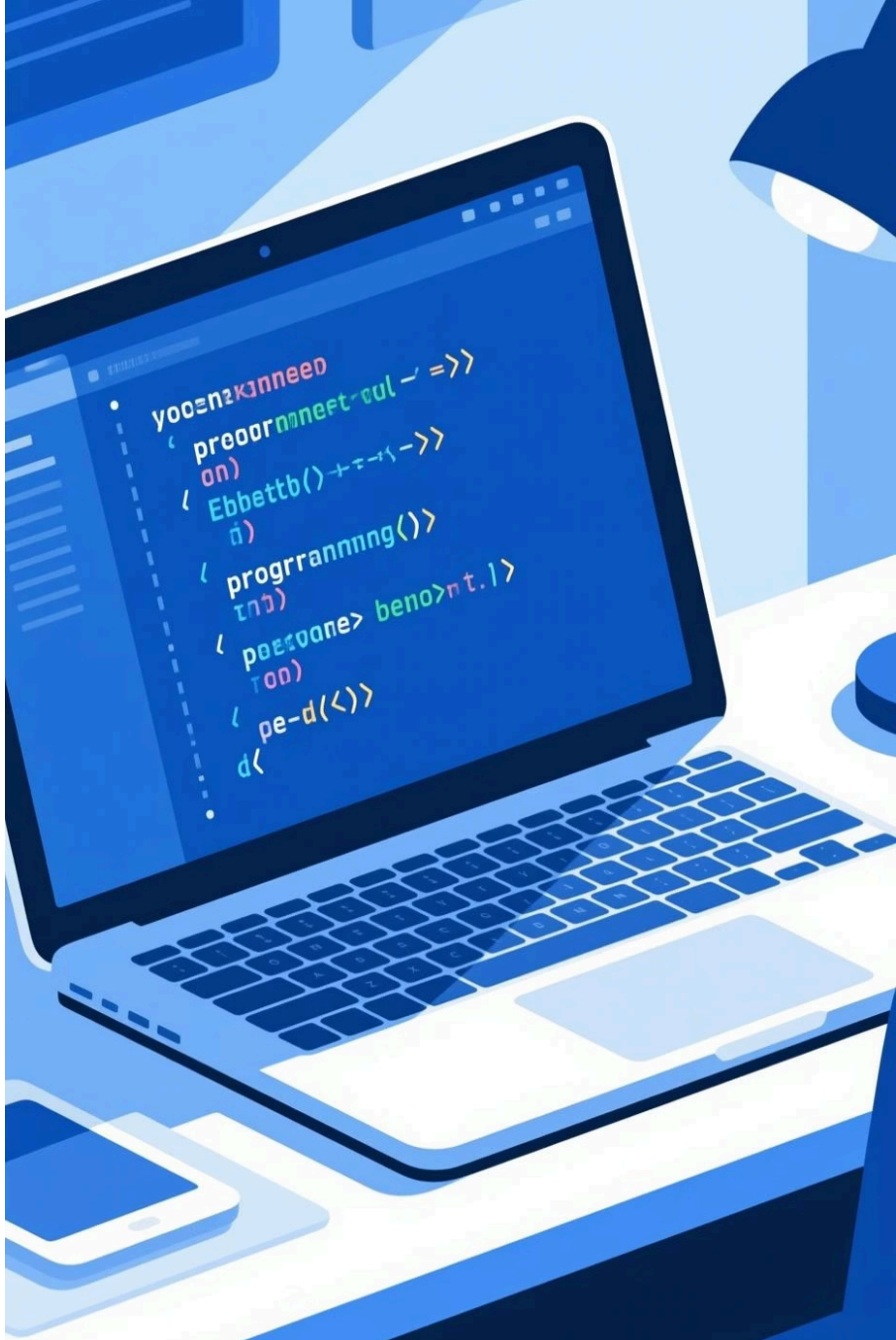
Python: From Basics to Web Requests

A beginner's guide to programming and the requests library



What is Python?

The "Swiss Army Knife" of Programming

Python is a **high-level, human-readable programming language** designed for clarity and simplicity. It's one of the most popular languages for beginners and professionals alike.

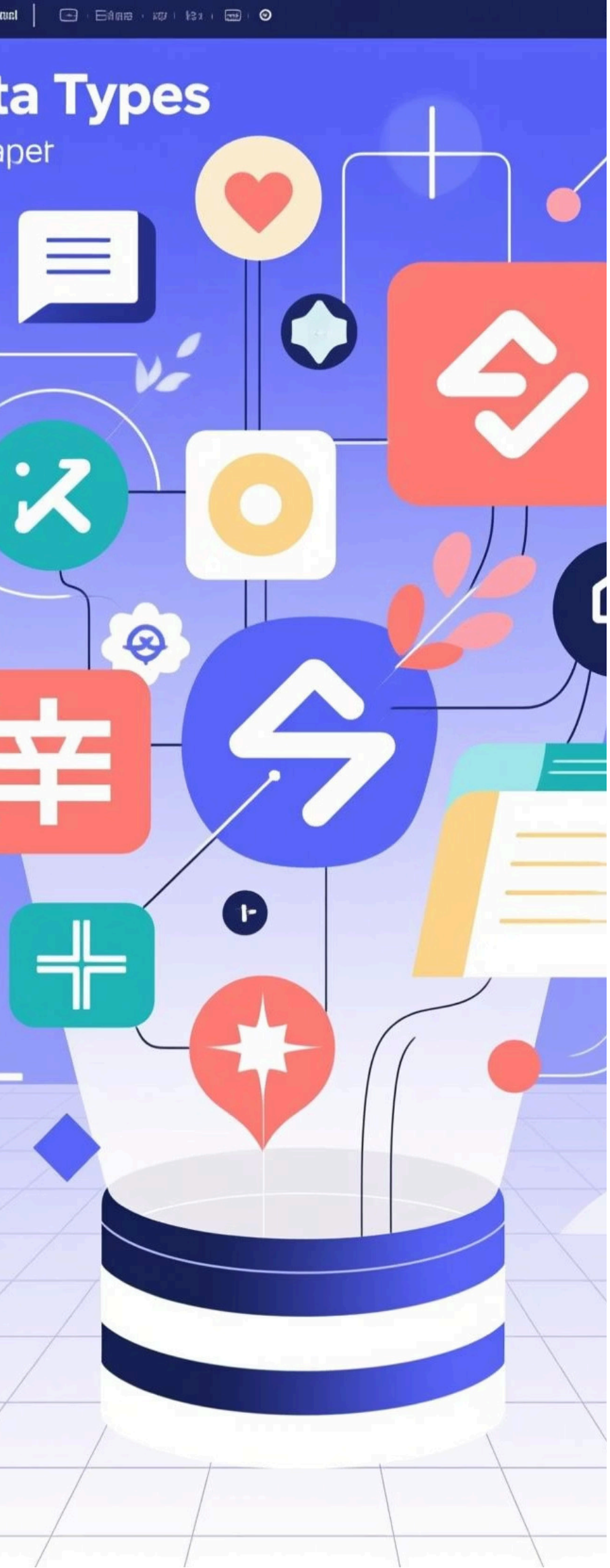
Key strengths:

- Powers web development, data science, automation, and AI
- "Batteries included" philosophy with rich standard library
- Clean syntax that reads almost like English

Your first Python program:

```
print("Hello, World!")
```

That's it! One simple line outputs text to your screen. Python removes unnecessary complexity so you can focus on solving problems.



The Basics: Variables and Data Types

Storing Your Data

Variables are **containers for storing information**. Python is dynamically typed—you don't need to declare what type of data you're storing. Python figures it out automatically.

Strings

Text data enclosed in quotes

```
name = "Alice"
```

Integers

Whole numbers without decimals

```
age = 30
```

Lists

Ordered collections of items

```
friends = ["Bob", "Charlie"]
```

Dictionaries

Key-value pairs for structured data

```
person = {"city": "New York"}
```

Making Decisions with If Statements

Control Flow: Teaching Your Code to Think

If statements let your programs make decisions based on conditions. Your code can respond differently depending on the situation—just like you do in real life.

The structure uses if, elif (else if), and else to handle different scenarios.

```
temperature = 25

if temperature > 30:
    print("It's a hot day!")
elif temperature > 20:
    print("It's a nice day.")
else:
    print("It's cold.")
```

In this example, Python checks each condition in order and runs only the first matching block. This is the foundation of program logic.

Repeating Actions with Loops

Automation Through Iteration

Loops are Python's way of doing repetitive tasks efficiently. Instead of writing the same code multiple times, loops handle it automatically.

For Loops

Iterate over sequences like lists or ranges

```
# Prints each name
for name in ["Bob", "Charlie"]:
    print(name)
```

Perfect when you know how many times to repeat

While Loops

Continue as long as a condition stays true

```
# Prints numbers 1 through 5
count = 1
while count <= 5:
    print(count)
    count = count + 1
```

Ideal when the number of iterations is unknown

Libraries & Modules

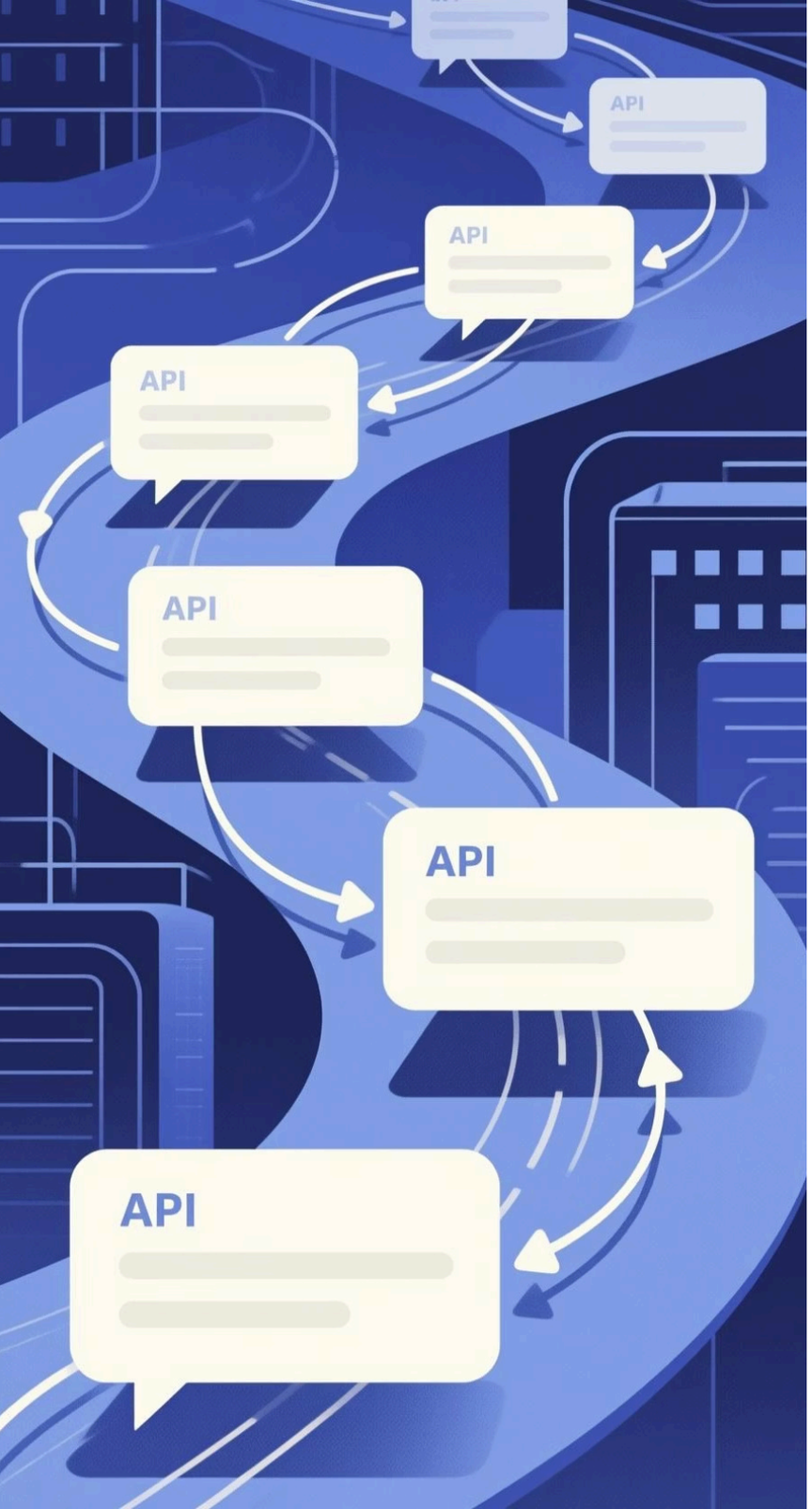
Python's Superpower: Standing on the Shoulders of Giants

Libraries (also called modules or packages) are collections of pre-written, tested code that extend Python's capabilities. They're Python's secret weapon.

Why libraries matter:

- Save time—don't reinvent the wheel
- Access specialized functionality instantly
- Leverage code written by experts

Need to work with dates? `import datetime`. Complex math? `import math`. And for web interaction, there's `requests`.



Spotlight: The requests Library

HTTP for Humans

What It Does

The **requests library** is the standard tool for making HTTP requests in Python. It simplifies sending and receiving data from web servers.

Getting Started

Install it once with pip:

```
pip install requests
```

Then import and use it in any project

Real-World Uses

Build web scrapers, interact with APIs, automate web tasks, gather data, and integrate with online services.

Fetching Data with GET Requests

Retrieving Information from the Web

A **GET request** retrieves data from a URL—it's what your browser does when you visit a website. With requests, you can automate this process and work with the data programmatically.

```
import requests

# The URL we want to get data from
url = "https://api.github.com/users/python"

# Send the GET request
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    # Get the response data as a JSON dictionary
    data = response.json()
    print(f"User: {data['login']}")
    print(f"Followers: {data['followers']}")
else:
    print("Failed to get data")
```

01

Import the library and specify your target URL

02

Send the request using `requests.get()`

03

Check the status code (200 = success)

04

Parse and use the returned data

Sending Data with POST Requests

Submitting Information to Web Services

A **POST request** sends data to a server. Common uses include:

- Submitting forms
- Posting comments or messages
- Logging into websites
- Creating new records in databases

While GET *retrieves*, POST *creates or modifies* information on the server.

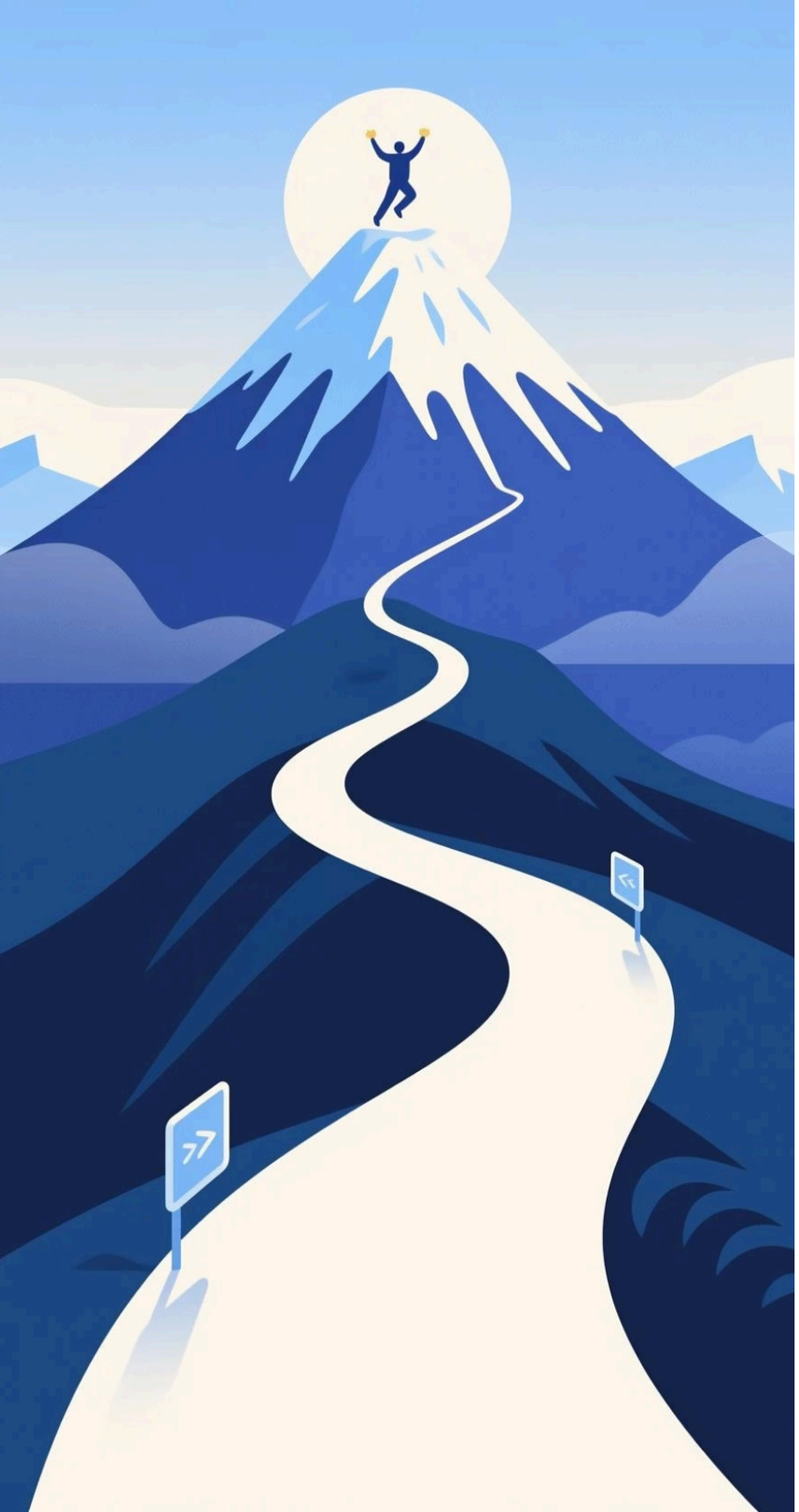
```
import requests

# URL to send data to
url = "https://httpbin.org/post"

# Data as a dictionary
payload = {
 "username": "my_user",
 "message": "Hello from Python!"
}

# Send the POST request
response = requests.post(url, data=payload)

# Print server's reply
print(response.json())
```



Your Python Journey Begins

Simple Yet Powerful

Python combines beginner-friendly syntax with professional-grade capabilities. Its clarity lets you focus on solving problems, not fighting complexity.

Master the Fundamentals

Variables, conditionals, and loops are the core building blocks. Every complex program is built from these simple concepts.

Leverage Libraries

Thousands of libraries like `requests` extend Python infinitely. The community has already solved many problems you'll encounter.

Connect to the Web

With requests, you can build scrapers, consume APIs, automate workflows, and integrate with any web service. The internet becomes your playground.

Next steps: Practice the basics, experiment with requests, and start building real projects. The best way to learn is by doing.