Zero-to-Web-Scraper: Python, NumPy, BeautifulSoup & Scrapy (Step-by-Step)

Friendly plan that starts at the **very basics**, with **simple syntax**, **examples**, and **hand-made diagrams**. We'll go **one module at a time**.

📚Roadmap (What we'll learn)

Module 1 — Python Crash Course (today) - Variables, types, lists & dicts - Loops, slicing, functions, imports - Files & errors - A tiny taste of HTTP + HTML so scraping later makes sense

Module 2 — NumPy Essentials - Arrays, shapes, slicing, vectorized math - Real-world mini-tasks (e.g., temperature data, simple analytics)

Module 3 — BeautifulSoup (HTML parsing) - Fetching a page, parsing HTML, selecting elements - Extracting text, links, tables; saving to CSV

Module 4 — Scrapy (full crawler) - Spiders, selectors, pipelines; pagination; respecting robots & delays - Exporting structured data (CSV/JSON)

Capstone - Crawl a small site (e.g., quotes or books), parse with **BeautifulSoup** patterns, scale it with **Scrapy**, then clean a numeric column with **NumPy** and save results.

> Tip: You can copy-paste and run these snippets directly.

Module 1 — Python Crash Course (absolute basics)

1) Setup (quick)

```
# check versions
python --version
pip --version

# create & activate a virtual environment (optional but recommended)
python -m venv .venv
# Windows PowerShell
.venv\Scripts\Activate.ps1
# macOS/Linux
source .venv/bin/activate

# install packages we'll use later
pip install requests beautifulsoup4 lxml numpy
```

---

2) Python syntax cheat-sheet

Variables & types

```
x = 10                # int
pi = 3.14             # float
name = "Harry"         # str
is_new = True         # bool
```

Printing & f-strings

```
print("Hello")
print(f"Hi {name}, x = {x}")
```

Lists (ordered, changeable)

```
nums = [10, 20, 30]
nums.append(40)       # [10, 20, 30, 40]
nums[0]               # 10
nums[1:3]             # [20, 30]  (slice)
```

Dictionaries (key → value)

```
person = {"name": "Asha", "age": 21}
person["name"]        # "Asha"
person.get("age", 0)  # 21 (or default 0)
```

Tuples (fixed lists)

```
pt = (3, 4)
```

---

3) Loops & decisions

```
age = 18
if age >= 18:
    print("adult")
else:
    print("minor")

# for-loop
for n in [2, 4, 6]:
```

```
    print(n)

# while-loop
count = 0
while count < 3:
    print("go", count)
    count += 1
```

Hand-made diagram — for-loop flow

```
[Start] → take next item → run block → more items? ──yes──► repeat
                                          └── no ──► [End]
```

---

4) Functions & modules

```
def add(a, b):
    return a + b

print(add(2, 5))  # 7

# using a module (library)
import math
print(math.sqrt(16))  # 4.0
```

Error handling (don't crash!)

```
try:
    n = int("42")
    bad = int("oops")
except ValueError as e:
    print("Could not convert:", e)
finally:
    print("Done")
```

---

5) Files & JSON (very common in scraping)

```
# write a text file
with open("hello.txt", "w", encoding="utf-8") as f:
    f.write("hello world\n")

# read it back
with open("hello.txt", "r", encoding="utf-8") as f:
    print(f.read())
```

```python
# JSON (structured text)
import json
user = {"name": "Asha", "skills": ["python", "scraping"]}
with open("user.json", "w", encoding="utf-8") as f:
    json.dump(user, f, ensure_ascii=False, indent=2)
```

---

6) Tiny taste of HTTP & HTML (so scraping will click later)

What happens when we fetch a page?

```
Your code ──HTTP GET──► Website Server ──HTML back──► Your code
```

Fetch a page (just the HTML text)

```python
import requests
url = "https://example.com"
resp = requests.get(url)
print(resp.status_code)  # 200 means OK
print(resp.text[:300])   # first 300 characters of the HTML
```

HTML looks like this (simplified)

```html
<html>
  <head><title>Example</title></head>
  <body>
    <h1>Welcome</h1>
    <p class="msg">Hello there!</p>
    <a href="/about">About</a>
  </body>
</html>
```

Hand-made diagram — HTML tree (DOM)

```
html
├─ head
│  └─ title: "Example"
└─ body
   ├─ h1: "Welcome"
   ├─ p.msg: "Hello there!"
   └─ a[href="/about"]: "About"
```

You'll use **BeautifulSoup** to walk this tree and pick the pieces you want.

---

7) Mini practice (today)

Q1. Make a list `[3, 6, 9, 12]` and print the last two numbers using slicing.
Q2. Create a dict for a book: title, author, year. Print the author.
Q3. Write a function `square(n)` that returns `n*n`. Test it for 7.
Q4. Write a file `note.txt` with the text `I am learning Python` and then read it back.
Q5. (Stretch) Use `requests` to fetch `https://example.com` and print the first 120 characters.

🔗Answers (peek when done)

```
# Q1
nums = [3, 6, 9, 12]
print(nums[-2:])  # [9, 12]

# Q2
book = {"title": "The Guide", "author": "R. K. Narayan", "year": 1958}
print(book["author"])  # R. K. Narayan

# Q3
def square(n):
    return n * n
print(square(7))  # 49

# Q4
with open("note.txt", "w", encoding="utf-8") as f:
    f.write("I am learning Python\n")

with open("note.txt", "r", encoding="utf-8") as f:
    print(f.read())

# Q5 (requires internet access from your machine)
import requests
r = requests.get("https://example.com")
print(r.text[:120])
```

What's next (Module 2 preview: NumPy Essentials)

We'll cover: - `np.array`, `np.arange`, `np.linspace`, shapes (`.shape`, `.ndim`), slicing - Vectorized math (add/mul, mean/std), aggregations - Real-world mini-task: daily temperatures → compute averages, min/max, simple alerts

Hand-made diagram — array shapes

```
1D: [1 2 3 4]

2D:
[
  [1 2 3]
  [4 5 6]
]
shape=(2,3), rows=2, cols=3
```

When you’re comfortable with Module 1, we’ll jump into NumPy (Module 2), then BeautifulSoup, then Scrapy. Keep this doc open; we’ll keep adding to it as you progress. 💪