



# STAGE-1 SCRAPING PLAYBOOK

*(Static Websites · Sync · Production-Safe)*

**Goal:**

Extract clean, usable data from a static website

**without crashing, without silent bugs, without fragile selectors**



## SYSTEM THINKING (ALWAYS START HERE)

Before writing code, answer this **in words**:

### ? What is the system?

URL → HTML → Records → Clean Data → JSON/File

### ? What can fail?

- Network (timeout, 403, 404)
- HTML structure change
- Missing fields
- Pagination break
- Silent bad data

👉 **Your code exists to survive these failures.**

## 1 PROJECT SKELETON (ALWAYS SAME)

```
scraper/  
|  
├─ scraper.py  
├─ data.json  
└─ README.md (optional but recommended)
```

## 2 IMPORTS — WHY EACH EXISTS

```
import requests                # HTTP requests  
from bs4 import BeautifulSoup # HTML parsing  
from urllib.parse import urljoin # pagination URLs  
from pathlib import Path       # safe file paths  
import json                   # saving data  
import logging                 # observability  
import time                   # rate limiting  
from typing import Optional    # explicit None handling
```

### 👉 Rule:

If you can't explain *why* an import exists — remove it.

## 3 LOGGING SETUP (NON-NEGOTIABLE)

```
logging.basicConfig(  
    level=logging.INFO,  
    format="%(asctime)s | %(levelname)s | %(message)s"  
)  
  
logger = logging.getLogger(__name__)
```

### Why?

- INFO → normal flow

- WARNING → recoverable problems
- ERROR → serious issues (but program alive)

✗ `print()` is banned for real systems.

## 4 CLASS = SCRAPER SYSTEM

```
class MyScraper:
```

### Why class?

- Holds **state**
- Groups **related behavior**
- Makes system extensible later (async, DB, APIs)

## 5 `__init__` — STATE DEFINITION {# 5 -init--state-definition }

```
def __init__(self, base_url: str):  
    self.base_url = base_url  
    self.headers = {  
        "User-Agent": "Mozilla/5.0",  
        "Accept-Language": "en-US,en;q=0.9"  
    }  
    self.data = []
```

### Job:

- Store configuration
- Create output container

👉 No logic here. Only setup.

## 6 FETCH LAYER (NETWORK BOUNDARY)

### Syntax

```
def fetch_html(self, url: str) -> Optional[str]:
```

### Job

- Talk to internet
- Return HTML **or None**
- Never crash program

### Canonical implementation

```
def fetch_html(self, url: str) -> Optional[str]:
    try:
        response = requests.get(url, headers=self.headers, timeout=10)

        if response.status_code == 200:
            response.encoding = "utf-8"
            return response.text

        elif response.status_code == 404:
            logger.warning(f"404 Page not found: {url}")
        elif response.status_code == 403:
            logger.error("403 Blocked by server")
        elif response.status_code == 429:
            logger.warning("429 Too many requests")
        else:
            logger.error(f"{response.status_code} unexpected status")

        return None

    except requests.exceptions.RequestException as e:
        logger.error(f"Network error: {e}")
        return None
```

#### 👉 Rule:

Network errors return None , not exceptions.

## 7 PARSE LAYER (HTML → DOM)

```
def parse_html(self, html: str) -> BeautifulSoup:
    return BeautifulSoup(html, "lxml")
```

### Job:

- Convert string → navigable structure
- No validation, no logic

## 8 SELECTOR STRATEGY (MOST IMPORTANT SKILL)

### 🔑 Selector Rules (MEMORIZE)

#### 1. Semantic > Styling

- h1 , h3 , a , text > CSS classes

#### 2. Local > Global

- record.find(...) > soup.find(...)

#### 3. Meaning-based

- currency symbol, keywords ( stock ), known values

## Example patterns

### Title

```
title = record.find("h3").find("a")["title"]
```

## Price

```
price = None
for p in record.find_all("p"):
    if "£" in p.text:
        price = p.text.strip()
        break
```

## Availability

```
availability = None
for p in record.find_all("p"):
    if "stock" in p.text.lower():
        availability = p.text.strip()
        break
```

## Rating

```
rating = None
for cls in record.find("p").get("class", []):
    if cls in {"One", "Two", "Three", "Four", "Five"}:
        rating = cls
```

## 9 VALIDATION (DATA GATE)

### Rule:

Bad data never enters the system

# Validation function

```
def is_valid_record(self, item: dict[str, Optional[str]]) -> bool:
    if not item.get("Title"):
        return False
    if not item.get("Price"):
        return False
    if not item.get("Availability"):
        return False
    return True
```

## 10 RECORD EXTRACTION PIPELINE

```
def extract_records(self, soup: BeautifulSoup) -> None:
    records = soup.find_all("article")

    for record in records:
        item = {
            "Title": title,
            "Price": price,
            "Availability": availability,
            "Rating": rating
        }

        if not self.is_valid_record(item):
            continue

        self.data.append(item)
```

## Flow:

extract → validate → append

Never:

extract → append → clean later ❌

## 1 1 PAGINATION LOOP (CONTROL FLOW)

```
def extract_all_pages(self) -> None:
    current_url = self.base_url

    while True:
        html = self.fetch_html(current_url)
        if html is None:
            logger.info(f"Skipping page: {current_url}")
            break

        soup = self.parse_html(html)
        self.extract_records(soup)
        logger.info(f"Scraped page: {current_url}")

        time.sleep(1)

        next_button = soup.find("li", class_="next")
        if not next_button:
            break

        next_link = next_button.find("a")["href"]
        current_url = urljoin(current_url, next_link)
```

### Key rules:

- break if HTML is missing
- Pagination URL comes from soup → no soup = stop
- Sleep = respect server

## 1 2 SAVE OUTPUT

```
def save_to_json(self, filename: str = "data.json") -> None:
    path = Path(__file__).parent / filename
    with open(path, "w", encoding="utf-8") as f:
        json.dump(self.data, f, indent=4, ensure_ascii=False)
```






## 1 3 MAIN ENTRY POINT

```
if __name__ == "__main__":
    scraper = MyScraper(BASE_URL)
    scraper.extract_all_pages()
    scraper.save_to_json()

    logger.info(f"Total records: {len(scraper.data)}")
```

## DEBUGGING CHECKLIST (WHEN STUCK)

Ask in this order:

1.  Is HTML coming? ( fetch\_html )
2.  Is soup created?
3.  Are selectors returning None ?
4.  Is validation skipping everything?
5.  Do logs show silent failures?

 **Never guess. Always trace.**

## STAGE-1 MENTAL MODEL (FINAL)

**Scraping is not about extraction.  
It is about survival.**

You now know:

- How to design a scraper
- How to protect it
- How to debug it
- How to reuse this structure everywhere