Session 2.1: Web Data Extraction: Scraping Financial and Competitor Intelligence with Beautiful Soup

THE BUSINESS SITUATION

What Are Competitors Charging Today?

- Our pricing team watches 3 competitor websites, where prices sit on web pages, not in CSV files
- Copying hundreds of prices by hand is slow and error-prone
- The pricing manager asks: what does each competitor charge, and who discounts most?

Today, Python reads a web page and turns it into a table.

Today's flow: 60 min lecture, then 120 min hands-on lab

EXPECTED OUTCOMES

By the end of today, you will be able to:

1. ***"How does a web page reach my computer?"***
   Explain an HTTP GET request and read HTML
   HTTP: HyperText Transfer Protocol; HTML: HyperText Markup Language, the text with tags that describes a page
2. ***"What is BetaMart charging?"***
   Extract prices with Beautiful Soup and CSS selectors
   CSS (Cascading Style Sheets) selector: an address for tags, such as `.price`
3. ***"What if a product has no stock tag?"***
   Handle a missing tag and fix an `AttributeError`
   Missing tag: the search returns `None` (nothing found)

Session Map: Follow the Numbers

Today's question: *What does each competitor charge, and who discounts most?*

1 Why it matters
Lecture, part 1

1. **Session Purpose**
   answer today's question
2. **Learning Outcomes**
   3 skills you will show today
3. **Business Context**
   Why Scrape?
4. **Data Basics**
   From Web Request to HTML

→

2 Learn the tools
Lecture, part 2

5. **Lab Dataset**
   2 web pages, 1 CSV
6. **Keywords**
   6 terms
7. **Core Foundations**
   6 ideas, one at a time
8. **IDE Setup**
   `pip install beautifulsoup4`
9. **Code Examples**
   3 demos + Quick Checks
10. **References**
    3 official docs

→

3 Practice
Lab

11. **Exercise 1: Guided**
    Competitor Price Benchmark
12. **Exercise 2: Applied**
    Scrape Stock Levels Safely
13. **Exercise 3: Advanced**
    Detect a Price Change on a New Site

Why Scrape? Competitor Prices Live on Web Pages

By hand

- Open each competitor page
- Copy each price into Excel
- Do it all again tomorrow

3 sites × 100 products = 300 copy-pastes a day, with typos

VS

With a Python script

- Reads every product on the page in seconds
- Gives the same result every run
- Runs again every morning

Output: a clean table, ready for pandas

RULES OF THE ROAD: SCRAPE RESPONSIBLY

Public pages only, never behind a login

Check `robots.txt`, the site's list of pages scripts may read, and its terms of use

Pause between requests: `time.sleep(2)` waits 2 seconds

Send a User-Agent: a label that tells the site who is asking

From Web Request to HTML

Browser or Python script

1. HTTP GET request: "send me this page" →

← 2. Response: the page as HTML text

Web server `betamart.example`

WHAT THE SERVER SENDS: HTML TEXT

```
<div class="product" data-sku="SKU-801">
  <h3 class="name">Office Desk Chair</h3>
  <span class="price">$49.50</span>
  <span class="stock">120 units</span>
</div>
```

Python reads this text. `data-sku` holds the SKU (Stock Keeping Unit), the product code.

→

WHAT THE BROWSER DRAWS

Office Desk Chair

$49.50

120 units

People see this page

Three Files for Today

SAVED WEB PAGE 1

`session_02_01_catalog.html`

- Fictional BetaMart page, 4 product cards
- Tags: `div.product`, `.name`, `.price`, `.stock`
- SKU-807 has no stock tag (on purpose)
- Used in: Demos 1-2, Exercise 2

SAVED WEB PAGE 2

`session_02_01_gamma.html`

- Fictional GammaGlobal page, 4 products
- Different tags: `li.item`, `.title`, `.list-price`, `.sale-price`
- Products not on sale have no `.sale-price`
- Used in: Exercise 3

SCRAPED DATA (CSV)

`session_02_01_dataset.csv`

- 12 rows: 3 competitors × 4 categories
- `List_Price_USD`, `Promotion_Discount_Pct`, `Stock_Availability_Units`
- Used in: Demo 3, Exercises 1 and 3

All three files are in the `data/` folder. Every name and price is fictional teaching data.

Six Terms You Need Today

REQUEST

HTTP GET

What: a HyperText Transfer Protocol (HTTP) request that asks a server for a page

Why: every page your browser or script reads starts with one

PAGE TEXT

HTML Tag

What: an HTML (HyperText Markup Language) label around content, such as `<span class="price">`

Why: tags tell you where each price or name sits

STRUCTURE

DOM Tree

What: Document Object Model: tags nested inside tags, like folders

Why: one product card holds its name, price, and stock

TOOL

Beautiful Soup (`bs4`)

What: a Python library that reads HTML into a searchable tree

Why: find tags by name or class in one line

ADDRESS

CSS Selector

What: a Cascading Style Sheets (CSS) address for tags: `.price` = class "price"

Why: `.select()` / `.select_one()` use it to pick tags

RESPONSIBILITY

Ethical Scraping

What: follow `robots.txt`, pause between requests, stay on public pages

Why: protects the website and your company

How Scraping Works

7.1
Text with Tags

`class="price"` marks where the price is

Remember: find the tag, take the text

7.2
Tags Inside Tags

Loop over cards, then search inside each

Remember: values stay with their product

7.3
Four Ways to Find Tags

First: `.find()`, `.select_one()`
All: `.find_all()`, `.select()`

Remember: then `.text` for the words

7.4
Text Is Not a Number Yet

`"$49.50"` → `49.5`

Remember: clean first, then convert

7.5
Class, Not Position

`.price` follows the label

Remember: recheck after a redesign

7.6
Missing Tag = `None`

`None.text` raises `AttributeError`

Remember: check `is not None` first

Scraping, Seen Up Close

1-2. FOUR CARDS BECOME FOUR ROWS

`div.product SKU-801`
`div.product SKU-804`
`div.product SKU-807`
`div.product SKU-810`

→ `for`

sku	name	price_usd
SKU-801	Office Desk Chair	49.5
SKU-804	Wireless Router	320.0
SKU-807	Compact Refrigerator	429.0
SKU-810	Cordless Drill	195.0

3. POSITION BREAKS, CLASS HOLDS

Tag order	Before	After a SALE badge is added
1st	h3.name	h3.name
2nd	**span.price**	**span.badge "SALE"**
3rd	span.stock	**span.price**

"2nd tag" now returns "SALE"; `.price` still finds the price

4. A MISSING TAG RETURNS `None`

sku	.select_one(".stock")
SKU-801	120 units
SKU-807	*None*

```
None.text  # AttributeError
if tag is not None:  # check first
```

The SKU-807 card on our page has no stock tag

Read a Web Page with Beautiful Soup

YourName_S21_demo_1.py — Input → Process → Output

```
# 1. Input: read the saved web page (HTML text)
from bs4 import BeautifulSoup
html_text = open("data/session_02_01_catalog.html").read()

# 2. Process: build the tree, then find the first product name
soup = BeautifulSoup(html_text, "html.parser")
first_name = soup.find("h3").text

# 3. Output: show what we found
print("Page title:", soup.title.text)
print("First product:", first_name)
```

1. INPUT

`open().read()`

- Reads the saved page as one long text
- First time only, in the Terminal: `pip install beautifulsoup4`
- On a live site, `requests.get(url).text` gives the same text (not run in class)

2. PROCESS

`BeautifulSoup()` and `.find()`

- `"html.parser"`: the built-in HTML reader
- `.find("h3")`: the first `h3` tag
- `.text`: the words inside it

3. OUTPUT

`soup.title.text`

The page title, then the first product

Find Every Match with `.find_all()`

NEW TOOL

`.find_all()`

What: returns **every** matching tag as a list; `.find()` returns only the first

Why: count how many products the page shows

Method	Result for "h3"
`.find()`	`1 tag: Office Desk Chair`
`.find_all()`	**`every h3 tag`**

YOUR TASK

Add One Line

In `YourName_S21_demo_1.py`, add one line to **3. Output**: print how many product names the page has.

Now

```
# (new line goes here)
```

Your change: fill in the yellow blank

```
print("Products on page:", len(soup._____("h3")))
```

Check: the Terminal shows `Products on page: 4`.

Turn Product Cards into Rows

YourName_S21_demo_2.py — Input → Process → Output

```
# 1. Input: read and parse the saved web page
from bs4 import BeautifulSoup
html_text = open("data/session_02_01_catalog.html").read()
soup = BeautifulSoup(html_text, "html.parser")

# 2. Process: one dictionary per product card
products = []
for card in soup.select(".product"):
    price_text = card.select_one(".price").text
    products.append({
        "sku": card["data-sku"],
        "name": card.select_one(".name").text.strip(),
        "price_usd": float(price_text.replace("$", "")),
    })

# 3. Output: one line per product
for item in products:
    print(item)
```

1. INPUT

Same read and parse

The page text becomes `soup`

2. PROCESS

Loop over the cards

- `.select()`: every card; `.select_one()`: one tag inside it
- `card["data-sku"]`: a value stored in the tag
- `float()`: `"$49.50"` to `49.5`, after removing `$`

3. OUTPUT

One dictionary per product

4 lines, one per card

Clean Text with `.get_text(strip=True)`

NEW TOOL

`.get_text(strip=True)`

What: returns the words inside a tag, with spaces and line breaks removed from both ends

Why: one call does the work of `.text.strip()`

Code	Result
`.text`	`"↵Office Desk Chair↵"`
`.get_text(strip=True)`	**`"Office Desk Chair"`**

↵ = a line break inside the tag (illustration)

YOUR TASK

Change One Line

In `YourName_S21_demo_2.py`, part **2. Process**: read the name with `.get_text()`.

Now

```
        "name": card.select_one(".name").text.strip(),
```

Your change: fill in the yellow blank

```
        "name": card.select_one(".name").__________(strip=True),
```

Check: the output does not change: the first line is still `{'sku': 'SKU-801', 'name': 'Office Desk Chair', 'price_usd': 49.5}`.

From Scraped Prices to a Comparison Table

YourName_S21_demo_3.py — Input → Process → Output

```
# 1. Input: prices already scraped from 3 competitor sites
import pandas as pd
df = pd.read_csv("data/session_02_01_dataset.csv")

# 2. Process: price after the advertised discount
df["Effective_Price_USD"] = (
    df["List_Price_USD"] * (1 - df["Promotion_Discount_Pct"] / 100)
).round(2)
electronics = df.loc[df["Category_Name"] == "Electronics"]

# 3. Output: compare competitors, then save the table
print(electronics[["Competitor_Name", "List_Price_USD",
                   "Effective_Price_USD"]])
electronics.to_csv("data/electronics_prices.csv", index=False)
```

1. INPUT

12 scraped prices

3 competitors, 4 categories, saved as CSV

2. PROCESS

Effective price

- List price minus the discount: `299.99 × (1 - 10 / 100) = 269.99`
- `.loc[]` keeps the Electronics rows

3. OUTPUT

`.to_csv()`

- Saves the table as a new CSV file
- `index=False`: leave out the row labels

Find the Lowest Price with `.min()`

NEW TOOL

`.min()`

What: returns the smallest value in a column; `.max()` returns the largest

Why: answers "what is the lowest competitor price?" in one step

Effective_Price_USD
269.99
281.6
263.5

Which one does `.min()` return?

YOUR TASK

Add Two Lines

In `YourName_S21_demo_3.py`, add two lines to **3. Output**: print the lowest Electronics price after discount.

Now

```
# (new line goes here)
```

Your change: fill in the yellow blank

```
lowest = electronics["Effective_Price_USD"].____()
print("Lowest price:", lowest)
```

Check: the Terminal shows `Lowest price: 263.5` (GammaGlobal).

Where to Look Things Up

BEAUTIFUL SOUP 4

beautiful-soup-4.readthedocs.io

Every method used today: `.find()`, `.find_all()`, `.select()`, `.get_text()`

REQUESTS

requests.readthedocs.io

How `requests.get(url)` fetches a live page, and how to send a User-Agent

W3C HTML STANDARD

w3.org/TR/html52

The World Wide Web Consortium's official rules for HTML tags

Three Exercises

EXERCISE 1: GUIDED

Competitor Price Benchmark

Goal: who discounts most, and who is cheapest, in each category?

File:
`YourName_S21_benchmark.py`

Check: Hardware: GammaGlobal discounts most (15%), AlphaRetail is cheapest (`157.5`)

EXERCISE 2: APPLIED

Scrape Stock Levels Safely

Goal: add stock to the Demo 2 table without crashing on SKU-807

File:
`YourName_S21_practice.py`

Check: 4 rows; SKU-807 shows `not shown`

EXERCISE 3: ADVANCED

Detect a Price Change on a New Site

Goal: which GammaGlobal discount changed since last week's CSV?

File:
`YourName_S21_challenge.py`

New: `pd.merge()` with `left_on`, `right_on`

Check: only `SKU-811` changed: `15.0`% vs `10.0`%

Full steps for all three exercises: sections 11-13 of the course page.