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

PEDAGOGICAL AGENDA

1:2 Delivery Rhythm

Session Learning Architecture

Structured 3-hour applied computing session designed for business students with zero prior coding experience.

- **Phase 1 (60 Min Lecture):** Computational foundations, business vocabulary, and syntactic mechanics.
- **Phase 2 (120 Min Lab):** Guided hands-on implementation, error diagnosis, and AI co-pilot resolution.
- **Zero Fluff Mandate:** Focus directly on executable Python code, tabular data wrangling, and commercial intelligence.

STRATEGIC OUTCOME

Operational Autonomy

Business Analytical Competency

Transforming passive spreadsheet users into autonomous programmatic decision makers.

- **Speed & Scale:** Processing datasets exceeding spreadsheet limits with instantaneous vectorized execution.
- **Repeatable Pipelines:** Replacing manual copy-paste workflows with fully automated Python scripts.
- **AI Debugging Literacy:** Leveraging modern IDEs to diagnose and resolve errors independently.

Core Computational Keywords & Business Meaning

CONCEPT 01 — Core Keyword

HyperText Transfer Protocol (HTTP GET)

The standard communication protocol used by web browsers and Python scripts to request public web page markup from external web servers.

CONCEPT 02 — Core Keyword

Document Object Model (DOM Tree)

The hierarchical tree structure representing an HTML web page, where parent tags contain nested child elements, classes, and text attributes.

CONCEPT 03 — Core Keyword

CSS Selector Targeting (.select() / .find())

Directing Python to extract specific information from a web page by matching tag names (`<table>`), CSS classes (`.price-tag`), or element identifiers (`#product-grid`).

CONCEPT 04 — Core Keyword

Beautiful Soup Parser

A Python library (`bs4`) that navigates, searches, and modifies the Document Object Model tree to extract clean textual data.

Computational Foundations & Business Operations

PILLAR 01 Architecture

**Core Programming & Computational Foundations

**

PILLAR 02 Architecture

Translating Web Markup into Tabular Intelligence

Most external competitor intelligence (pricing, product assortment, inventory availability) is published on public HTML pages rather than clean CSV files. Web scraping automates the extraction of this unstructured markup into structured tabular datasets.

PILLAR 03 Architecture

Navigating the HTML Document Object Model

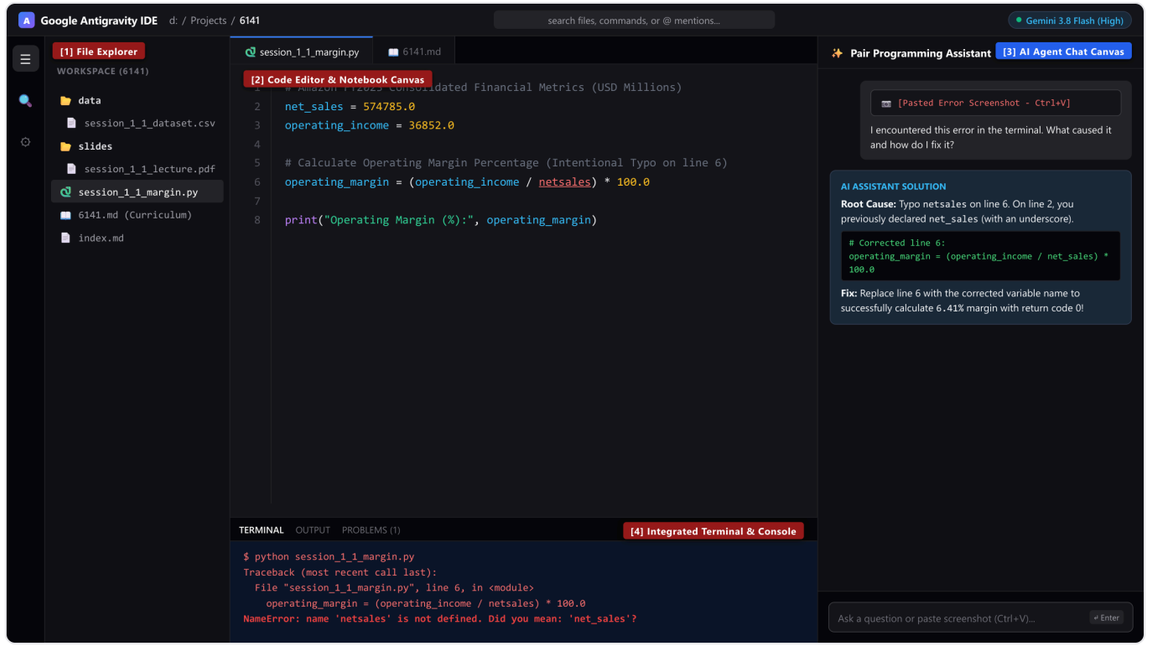
Understanding the tree hierarchy of web documents. An e-commerce product catalog consists of repeated container elements (`<div class="product-card">`) housing title headings, price tags, and inventory spans.

PILLAR 04 Architecture

Robust Selector Strategy

Target semantic CSS classes and attributes rather than fragile absolute XPath coordinates. If a website changes its navigation bar, robust CSS class selectors continue extracting pricing tables without breaking.

Google Antigravity IDE Architecture & AI Debugging Protocol



[1] File Explorer

Project Hierarchy

Manage datasets in `data/`, scripts in root, and exported artifacts.

[2] Code Editor

Syntax Workspace

Modern syntax highlighting, lint diagnostics, and line-by-line script authoring.

[3] AI Agent Chat

AI Pair Programmer

Paste error screenshots directly to inspect root causes and get verified fixes.

[4] Terminal

Execution Engine

Run Python scripts (`python script.py`) and view console logs and tracebacks.

Zero-Friction AI Error Resolution Protocol

1. **Capture:** Press `Win + Shift + S` to clip terminal error traceback.
2. **Paste:** Press `Ctrl + V` into AI Chat [3] with "Explain root cause & fix".
3. **Apply:** Review the explanation and apply verified fix into Editor [2].

Script Demonstration 1: Script Demonstration 1

session_2_1_demo_1.py — Python 3.10

```
"kw">class="kw">import requests
    "kw">class="kw">from bs4 "kw">class="kw">import BeautifulSoup

    # Simulated public e-commerce pricing table
    html_markup = """
    <div "kw">class="catalog">
        <div "kw">class="product" data-sku="SKU-801">
            <h3 "kw">class="name">Enterprise Router</h3>
            <span "kw">class="price">$299.99</span>
            <span "kw">class="stock">In Stock (45 units)</span>
        </div>
        <div "kw">class="product" data-sku="SKU-802">
            <h3 "kw">class="name">Office Desk Chair</h3>
            <span "kw">class="price">$49.50</span>
```

```
$ python session_2_1_demo_1.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Script Demonstration 2: Script Demonstration 2

session_2_1_demo_2.py — Python 3.10

```
products = []
    "kw">class="kw">for item "kw">class="kw">in soup."fn">select(".product"):
        sku = item["data-sku"]
        name = item.select_one(".name").text.strip()
        raw_price = item.select_one(".price").text.strip()
        price_usd = float(raw_price.replace("$", ""))
        products.append({"sku": sku, "name": name, "price_usd": price_usd})

    "fn">print("Parsed Products:", products)
```

```
$ python session_2_1_demo_2.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Script Demonstration 3: Script Demonstration 3

session_2_1_demo_3.py — Python 3.10

```
"kw">class="kw">import pandas "kw">class="kw">as pd

    catalog_df = pd.DataFrame(products)
    "fn">print(catalog_df."fn">describe())
```

```
$ python session_2_1_demo_3.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Interactive Two-Tier Lab Protocol & AI Diagnosis

EXERCISE 1

Guided Lab

Foundational Implementation

Step-by-step guided practice establishing baseline syntax mastery in Antigravity IDE.

- **Step 1:** Ingest dataset from `data/session_2_1_dataset.csv`.
- **Step 2:** Implement core analytical functions and compute primary business metrics.
- **Step 3:** Format console outputs and verify calculations against baseline ledger totals.
- **Step 4:** Run in Terminal [4] and confirm clean execution with exit code 0.

EXERCISE 2+

Coding Challenge

Advanced Scripting & AI Debugging

Applied programming challenge expanding pipeline logic and resolving intentional exceptions.

- **Task 1 (Pipeline Expansion):** Add secondary business unit logic and multi-tier calculations.
- **Task 2 (Deliberate Error & AI Capture):** Introduce intentional syntax or type error, capture traceback with `Win + Shift + S`, and diagnose via AI Chat [3].
- **Task 3 (Verified Production Run):** Apply verified fix, export audit output, and confirm zero errors with return code 0.

Interactive Lab Protocol & Delivery Rhythm

Hands-On Code Execution

The Pedagogical Contract

Computational mastery develops through active problem-solving. By creating intentional errors and using AI co-pilots to resolve them, students build lifelong debugging autonomy.