**Dell Software Engineering Challenge**

**Desired Output:**A collection of files, with all the code necessary to solve the 3 challenges proposed here and answer the questions.  
The code should be properly commented inline.

**Topics Covered:**

Languages: HTML, CSS, JavaScript, Python

Technologies: Web Scraping, HTTP Requests, Async/Await, JSON, REST APIs, Frontend, Backend.

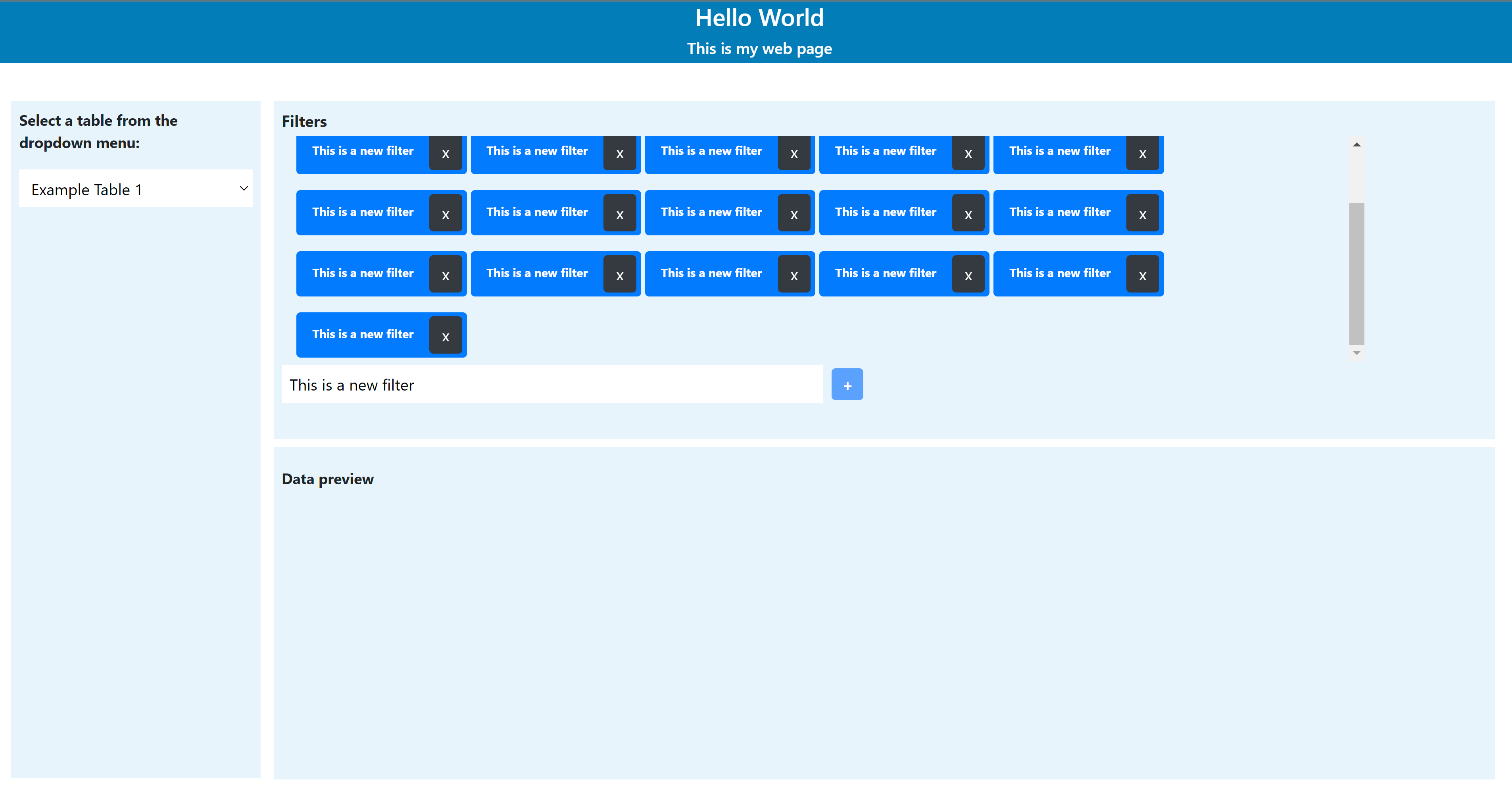
**Exercise 1:**

**Languages**: HTML, CSS, JavaScript

Create a Web Page that looks as similar as possible to the screenshot below.  
It should be possible to add a new Filter element in the “Filters” section, using the text input field and the  button.

When the filter elements are too many, a scrollbar will appear on the right side.  
You can also delete each Filter element by clicking the  button next to them.

You can use HTML, CSS, vanilla JavaScript, jQuery, Bootstrap or any other styling library and framework.



**Result:** Kovac\_dell\_cs > ex\_1 > index.html

**Questions:**

1. **How would you deploy this web page so that users can access it online? Talk about the infrastructure, the resources, and the technologies needed.**

*Infrastructure options* (low traffic: shared hosting or VPS, high traffic: Cloud platforms):

1. Shared hostings (e.g. bluehost), Pros: Easy to set up, cost-effective, Cons: limited control.
2. Virtual Private Server (e.g. linode), Pros: More control over environment, Cons: higher cost.
3. Cloud Platforms (e.g. AWS), Pros: highly scalable, Cons: high cost for high traffic.

*Technologies needed:*

1. Web Server (e.g. Apache)
2. Database Server (e.g. PostgreSQL)
3. Database Management Tools (phpMyAdmin – MySQL)
4. Server-side Scripting Language (e.g. Python – Flask, PHP)
5. Domain Name + Security (Secure Login, role-based access control)
6. File Transfer Protocol/Version control systems (GiLab CI/CD) – git + deployment

*Resources:*

*low traffic (<1000 users/day)*: CPU: 1-2 vCPUs, RAM: 2-4 GB, Storage: 10-20 GB

*medium traffic (1000 – 10,000):* CPU: 2-4 vCPUs, RAM: 4-8 GB, Storage: 20-50 GB

*high traffic (>10,000)*: CPU: 4+ vCPUs, RAM: 8+ GB, Storage: 50+ GB

*Simple steps of deployment*

1. Planning and preparation: functionality, security
2. Development: front-end (html, css, js), back-end (python), database setup
3. Testing
4. Deployment: method (Cloud platforms), version control
5. Security: https, secure login and access
6. **How would you modify the code in order to populate the dropdown menu with values coming from a database? Talk about possible database solutions, and how the backend can interact with the frontend.**

In Kovac\_dell\_cs > ex\_1 > ex\_1\_q2 I provided a possible solution with Python – Flask (first run python app.py), using publicly available PostgreSQL database. There are of course other options:

1. Node.js with Express
2. PHP, ..

**Exercise 2:**

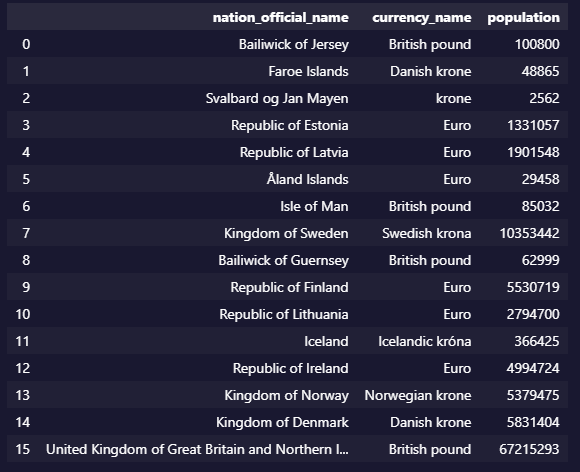
**Language**: Python

Use the following public REST API.  
<https://restcountries.com/>

Get data only for Northern European countries, and filter only for the following fields:   
name  
currencies  
population

The API call should be asynchronous and encapsulated in a function.

Load the JSON response into a dictionary and then turn it into a single index Pandas dataframe.  
The columns should be “nation\_official\_name”, “currency\_name” and “population”.

The dataframe should look like this:  


then connect to a hypothetical Postgres Database and load the dataframe to a new table, in REPLACE mode.

**Result 1:** Kovac\_dell\_cs > ex\_2 > restcountries\_data.py (getting data, tested)

**Result 2:** Kovac\_dell\_cs > ex\_2 > rest\_api\_ex.py (getting data + saving into Postgres, not tested)

**Questions:**

1. **If you didn’t know the structure of the JSON, and there was an arbitrary level of nesting of arrays and dictionaries, how would you need to change the code to dynamically unnest the data into a single-indexed dataframe?**

If the structure is uknown I would create a flatten json. Imagine Json looks like:

{

"person": {

"name": "Alice",

"address": {

"street": "123 Main St",

"city": "Anytown"

}

}

}

To access the value “Alice”, flatten json would have key: “person\_name”. Code example is available in Kovac\_dell\_cs > ex\_2 > sand.py.

1. **If you had to scale this application to read and load data for hundreds of countries and refresh the database every few minutes, what strategies could be used in terms of coding patterns, technologies, resources and infrastructure?**

Using libraries like aiohttp to make asynchronous requests concurrently for multiple countries. Possible API rate limits.

**Exercise 3:**

**Language**: Python

Scrape the S&P 500 companies table from the following Wikipedia page:  
<https://en.wikipedia.org/wiki/List_of_S%26P_500_companies>

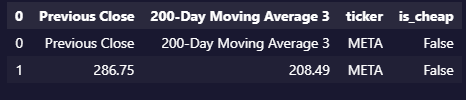
Save the companies ticker symbols into a list. Cut the list to take only the first 50 elements.

For each ticker symbol in the list, call the following API In order to get the Previous Close value for each company:  
<https://finance.yahoo.com/quote/AAPL?p=AAPLtsrc=fin-srch>  
Save this value and the ticker symbol in a Pandas dataframe.

For each ticker symbol also call the following API endpoint in order to get the 200-Day Moving Average value:  
<https://finance.yahoo.com/quote/AAPL/key-statistics?p=AAPL>  
Save this value in a new column of the same dataframe.

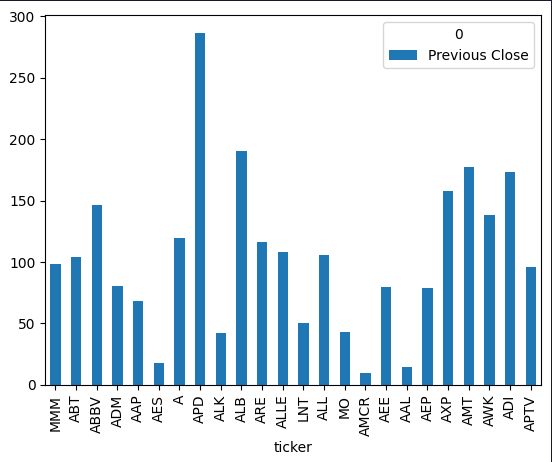
Compute a new column in the dataframe called “is\_cheap” with a Boolean value which is True if the Previous Close is lower than the 200-Day Moving Average and False otherwise.

Example:



Concatenate all dataframes for all ticker symbols in one.

Display the dataframe on a plot only for the companies where is\_cheap = True.  
On the X axis should be the Ticker symbol and on the Y axis the Previous Close value.



**Result:** Kovac\_dell\_cs > ex\_3 > scrapper.py

**Questions:**

1. **If the Wikipedia table was lazy loaded, and only appeared after a few seconds from opening the page, what libraries and strategies could you adopt to get the data?**

Because API from yahoo didn’t work, I used BeautifulSoup and Selenium which are capable to handle lazy load – using for instance WebDriverWait.

1. **If you had to run this script for thousands of companies instead of 500, what kind of patterns, libraries and/or optimization techniques could you use to keep the process efficient?**

* Optimizing data fetching: liberaries like multiprocessing (threading) to run scraping for multiple companies, or batch requests (group of companies and library requests/futures should be able to help asznchronous requests), and of course API if works.
* Code optimization: instead of loading entire webpage, using headless browsers, and caching (cachetools) could help to store previously retrieved data (for companies without change).
* Scalable infrastructure: running script on cloud platform (Gloogle Cloud Funcitons)