Scenario

In the ever-expanding digital landscape, the ability to extract and analyze data from websites has become an indispensable skill for businesses, researchers, and individuals alike. Web scraping, the process of automating data extraction from websites, empowers us to gather valuable information, monitor trends, and gain insights that can drive informed decision-making. Enter Scrapy, a powerful Python framework designed specifically for web scraping and crawling.

Objective

This activity is designed to introduce you to the fundamentals of web scraping in Python using the Scrapy framework. You will learn how to install and set up a Scrapy project, define spiders to extract specific data from websites, and navigate through web pages to collect data. By the end of this activity, you will have the knowledge to create basic web scrapers for extracting data.

Instructions

Download this compressed folder and save it on your machine's desktop and extract it to a folder where it is easily accessible. Extract the contents of the compressed folder, which contains a starter Scrapy project.

After extracting the files in the setup, navigate to that folder in a Terminal or Command Prompt window and navigate to the scrapy_demo_start folder. Verify that Scrapy has been installed on your machine by running the command:

```
pip install scrapy
```

Open the scrapy_demo_start folder in Visual Studio Code. Your project will have the following structure:

```
scrapy_demo_start/

          scrapy.cfg           # Configuration file for Scrapy

     scrapy_demo/         # Main project folder

          __init__.py          # Marks this as a Python package

          spiders/             # Folder to store your spider files

               quotes_spider.py    # Your spider code will go here

          settings.py          # Scrapy settings (optional adjustments)
```

- **scrapy.cfg:** This file contains your Scrapy project configuration.
- **scrapy_demo/spiders/:** This folder contains the spider scripts, where the main logic of your web scraping will reside.
- **scrapy_demo/settings.py:** This file is where Scrapy's settings are stored. You can customize or leave the default settings.
- **scrapy_demo/spiders/quotes_spider.py:** Inside the `scrapy_demo/spiders/` folder, a starting file, quotes_spider.py, has been added.

Now that you have opened the project, you will build out scraping logic. Navigate to the quotes_spider.py in Visual Studio Code.

The site you will be scraping is http://quotes.toscrape.com/ - a popular site to use for scraping. The main page's HTML contains a series of quotes in this format:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

You want to extract the quote and the author (highlighted in bold above).

Step 3.1: Define scraping variables

Scrapy requires two variables: name and start_urls - the name of the spider can be anything. You just need to know the name of the spider when you run the command **`scrapy crawl`** later on. In this case, you will name the spider **`quotes`**, so you will run the command **`scrapy crawl quotes`**. Locate the line of code labeled **`### STEP 3.1 ADD CODE HERE ###`** and add the following lines of code, ensuring the indenting is even with the provided parse function:

```
name = 'quotes'

start_urls = ['http://quotes.toscrape.com']
```

Step 3.2: Scrape quotes

The parse method will start by extracting the quotes from the page. Here, you will start by loading the quote divs. Refer to the HTML snippet above to view an example of the div. Locate the comment labeled **`### STEP 3.2 ADD CODE HERE ###`** and add the following code:

```
quotes = response.css('div.quote')
```

At this point, in your Terminal or Command Prompt, run the scraper in the scrapy_demo_start folder by issuing the command:

```
scrapy crawl quotes
```

If you see the following error, ensure you are in the correct folder:

```
Scrapy 2.12.0 - no active project

Unknown command: crawl

Use "scrapy" to see available commands

(base)
```

Scrapy tends to print out a lot of details, so you may need to scroll up, but you should find a line stating **`Found 10 quotes on the page`**.

You are planning on storing just the quote and author from each div. You will start with a loop to extract the text from the appropriate elements:

```
<span class="text" itemprop="text">“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”</span>
```

That quote text can be extracted by locating **`span.text::text`** - the text value contained in the **`span`** named **`text`**.

```
<small class="author" itemprop="author">Albert Einstein</small>
```

The author's name can be extracted by locating **`small.author::text`** - the text value contained in the **`small`** tag named **`author`**.

Step 4.1: Extract values from each div

To load the values into a dictionary, locate the spot labeled **`### STEP 4.1 ADD CODE HERE ###`** and paste the following code in:

```
1
2
3
4
5
```

Recall, yield is used for asynchronous requests.

You can run the spider again and you will see a list of quotes go by:

```
scrapy crawl quotes
```

The main page (https://quotes.toscrape.com) includes a "Next" link. You can follow that link by searching for the "Next" button.

The HTML code for the next button is shown here:

```
1
```

The href can be extracted by locating **`li.next a::attr(href)`**. If it exists, there is another page.

Step 5.1: Extract text from following pages

To load the values into a dictionary, locate the spot labeled **`### STEP 5.1 ADD CODE HERE ###`** and paste the following code in:

```
3
```

You can run the spider one more time and there will be more quotes.

```
scrapy crawl quotes
```

Step 5.2: Save output to JSON file

The output is not easy to follow, but you can issue the following command to get the dictionary stored in an output file:

```
scrapy crawl quotes -o output.json
```

Open the output.json file and you will see a list of quotes, each with a quote and author key.

To scrape websites with a lot of data, you can use Scrapy Cluster to distribute the work across multiple machines. This is like having a team of spiders working together! Here's how you can set it up. This step is optional as you would need multiple machines set up to do this.

1. Install Scrapy Cluster and Redis: **`pip install scrapy-cluster redis`**

2. Modify your spider:

```
1
2
3
4
5
6
```

```
7
8
9
10
11
12
13
14
```

3. Configure Scrapy Cluster: Create a **`settings.py`** file in your project's root directory and add configurations like:

- **`CLUSTER_MASTER_PORT`:** The port for the master node (e.g., 6800).
- **`CLUSTER_SLAVE_PORTS`:** A list of ports for slave nodes (e.g., [6801, 6802, 6803]).
- **`REDIS_HOST`:** The host of your Redis server (e.g., 'localhost').
- **`REDIS_PORT`:** The port of your Redis server (e.g., 6379).

Here, the settings.py would look like this:

```
1
2
3
4
5
```

4. Start Redis Server: Scrapy Cluster uses Redis for communication. Make sure you have Redis installed and running.: On your server, start the Redis server by issuing the command:

```
redis-server
```

The server will listen on the default port with a default setup. It should be left running while you run your exercise. Only the server requires Redis to be run.

5. Run the spider: In a new Terminal or command prompt window, start the master node with **`python -m myproject.spiders.quotes_spider`**.

6. On other machines, the settings.py will be set up differently. Fill that in first, ensuring each machine has a unique number. You will need the IP address for your server machine

```
1
2
3
4
```

7. Open a Terminal or command prompt on each slave node. Ensure each slave node has the required libraries:

```
pip install scrapy-cluster redis
```

8. Start the scraper on each slave node.

```
python -m myproject.spiders.quotes_spider
```

Notes:

- **scrapy.cfg:** This file is auto-generated when you create a Scrapy project. It configures Scrapy's default behavior. For this challenge, you don't need to modify this file unless you're adding more advanced configurations.
- **settings.py:** You can customize your project's settings (e.g., changing the user-agent or adjusting the download delay) here, but this is not required for the basic challenge.

1. You're starting a new web scraping project to gather product data from an e-commerce site. After setting up your project, what is the next step to get a spider running? Select the best answer. **1 / 1 point**

Adding spider logic to parse the page

Setting up a database

Deploying the spider to a server

Crawling the website

Correct
Correct! The spider logic will determine how the page is parsed.

2. The product names on the website are within `<h2 class="product-name">` tags. Which Scrapy selector will you use to extract them? Select the best answer. **1 / 1 point**

```
response.extract('h2.product-name')
```

```
response.css('h2.product-name::text')
```

```
response.get('h2')
```

```
response.get('h2.product-name')
```

Correct
Correct! This uses a CSS selector to target the text within the h2 tags.

3. You've successfully extracted data from the first page of product listings. How will you navigate to the next page and continue scraping? Select the best answer. **1 / 1 point**

Use response.follow() with the next page's URL

Use response.redirect()

Use response.css to find the next page link and click it

Manually construct the next page's URL and make a new request

Correct
Correct! This is the primary way to follow links in Scrapy.

4. You need to save the output for your spider named baseball. What option would you choose to save it to a file named statistics.json? Select the best answer.

scrapy crawl baseball > statistics.json

scrapy crawl baseball

scrapy crawl statistics.json

scrapy crawl baseball -o statistics.json

Correct
Correct! This will save to the required file.