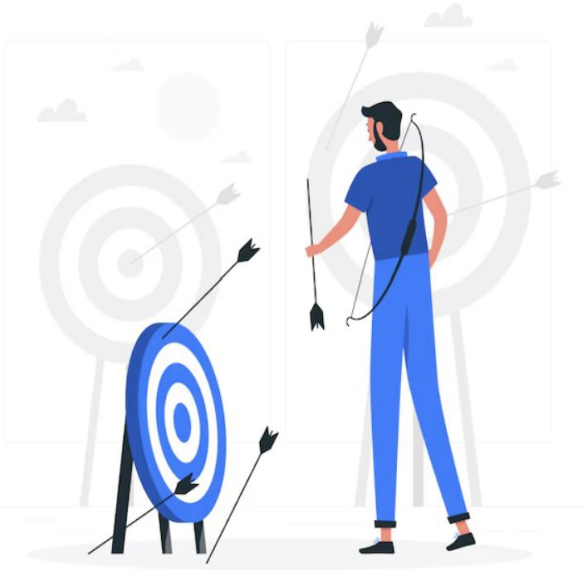
PW SKILLS | DevOps and Cloud Computing

# Python for Web Automation and Data Processing

# Objective

- Understand the fundamentals of web automation and HTTP requests.
- Use Beautiful Soup for web scraping and data extraction.
- Process and manipulate structured data using Pandas and NumPy.
- Apply automation techniques for common tasks like data extraction and transformation.



PW SKILLS | DevOps and Cloud Computing

# Introducing web automation and how Python interacts with web pages

# Web automation

Introduction to Web Automation

- Web automation involves using software tools and scripts to perform tasks on web pages without manual intervention.

- It is commonly used for testing, data scraping, form submissions, and repetitive browser interactions.

# Web automation

How Python Interacts with Web Pages

Python interacts with web pages using libraries like:

- Selenium – Automates browsers for testing and web interactions.
- Beautiful Soup – Parses HTML and extracts data from web pages.
- Requests – Sends HTTP requests to fetch web content.
- Playwright – A modern automation framework for faster and more reliable browser control.

PW SKILLS | DevOps and Cloud Computing

# Explaining HTTP requests and responses (GET, POST).

# HTTP requests & responses

The Hypertext Transfer Protocol (HTTP) enables communication between a client (browser or script) and a server. It works through requests sent by the client and responses returned by the server.

HTTP Requests

An HTTP request consists of:

- Method (GET, POST, etc.)
- URL (the resource being accessed)
- Headers (metadata like content type and authentication)
- Body (data sent in requests like form submissions)

# HTTP requests & responses

Common HTTP Methods

1. GET – Retrieves data from a server.
- Example: Fetching a webpage (GET /home).
- No request body; parameters are in the URL.

2. POST – Sends data to the server to create or update a resource.
- Example: Submitting a login form (POST /login).
- Data is included in the request body.

# HTTP requests & responses

PW SKILLS

HTTP Responses

A response includes:

- Status Code (e.g., 200 OK, 404 Not Found, 500 Internal Server Error)
- Headers (metadata like content type and caching policies)
- Body (HTML, JSON, or other response data)

PW SKILLS | DevOps and Cloud Computing

# Demonstrating using the requests library to fetch web page content

# Let’s do it

Here’s a simple example using the ‘requests’ library to fetch web page content:

```
import requests

# Define the URL
url = "https://example.com"

# Send a GET request to fetch the page content
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    print("Page Content:")
    print(response.text[:500])  # Display first 500 characters of the content
else:
    print(f"Failed to fetch page. Status Code: {response.status_code}")
```

PW SKILLS | DevOps and Cloud Computing

# The concept of web scraping and ethical considerations

# Web scraping & ethical considerations

Web scraping is the process of extracting data from websites using automated scripts. It involves sending HTTP requests, parsing HTML content, and retrieving structured data. Python libraries like Beautiful Soup, Requests, and Scrapy are commonly used for web scraping.

Ethical Considerations

- Respect ‘robots.txt’ – Check a website’s ‘robots.txt’ file to see what is allowed.
- Avoid Overloading Servers – Use rate limiting to prevent excessive requests.
- No Unauthorized Data Use – Do not scrape personal, copyrighted, or sensitive data.
- Follow Website Terms – Ensure compliance with legal and ethical guidelines.

PW SKILLS | DevOps and Cloud Computing

# Introducing BeautifulSoup and demonstrating parsing HTML

# Let’s do it

‘BeautifulSoup’ is a Python library used for parsing HTML and XML documents. It helps extract data from web pages by navigating the HTML structure.

Example: Parsing HTML with BeautifulSoup

```
from bs4 import BeautifulSoup

html = """
<html>
    <head><title>Sample Page</title></head>
    <body>
        <h1>Hello, Web Scraping!</h1>
        <p class="info">This is a sample paragraph.</p>
    </body>
</html>
"""

# Create a BeautifulSoup object
soup = BeautifulSoup(html, "html.parser")

# Extract elements
print("Title:", soup.title.text)
print("Heading:", soup.h1.text)
print("Paragraph:", soup.find("p", class_="info").text)
```

# Pop Quiz

Q. Which library is commonly used with BeautifulSoup to fetch web page content?

A

NumPy

B

Requests

# Pop Quiz

Q. Which library is commonly used with BeautifulSoup to fetch web page content?

A

NumPy

B

Requests

PW SKILLS | DevOps and Cloud Computing

# Demonstrating extracting specific elements from a webpage

# Let’s do it

Example: Extracting Title, Links, and Images

```
import requests
from bs4 import BeautifulSoup

# Define the URL
url = "https://example.com"

# Send a GET request and parse content
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

# Extract and print the page title
print("Page Title:", soup.title.text)

# Extract and print all links
print("\nLinks:")
for link in soup.find_all("a"):
    print(link.get("href"))

# Extract and print all image sources
print("\nImages:")
for img in soup.find_all("img"):
    print(img.get("src"))
```

# Take A 5-Minute Break!


- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS | DevOps and Cloud Computing

# Introducing the importance of structured data processing

# Structured data processing

Structured data processing is crucial for efficiently organizing, analyzing, and utilizing information. It ensures data consistency, accuracy, and accessibility, making it easier for applications, databases, and AI models to interpret and process information effectively.

Key Benefits:

- Data Consistency – Ensures uniform formatting and organization.
- Efficient Analysis – Enables faster querying, filtering, and reporting.
- Automation & Scalability – Facilitates seamless data handling in web scraping, ETL (Extract, Transform, Load), and AI applications.
- Interoperability – Structured data can be easily shared across systems using formats like JSON, CSV, or databases.

PW SKILLS | DevOps and Cloud Computing

# Demonstrating creating and manipulating data with Pandas

# Let’s do it

‘pandas’ is a powerful Python library for working with structured data, enabling easy data reading, filtering, sorting, and aggregation.

1. Read Data from CSV and JSON Files:

```
import pandas as pd

# Read data from a CSV file
csv_df = pd.read_csv("data.csv")
print("CSV Data:\n", csv_df.head())

# Read data from a JSON file
json_df = pd.read_json("data.json")
print("\nJSON Data:\n", json_df.head())
```

# Let’s do it

2. Filtering Data (e.g., Select rows where price > 100):

```
filtered_df = csv_df[csv_df["price"] > 100]
print("\nFiltered Data (price > 100):\n", filtered_df)
```

3. Sorting Data (e.g., Sort by price in descending order):

```
sorted_df = csv_df.sort_values(by="price", ascending=False)
print("\nSorted Data (by price, descending):\n", sorted_df)
```

# Let’s do it

4. Aggregation (e.g., Calculate average price per category):

```
aggregated_df = csv_df.groupby("category")["price"].mean()
print("\nAverage Price per Category:\n", aggregated_df)
```

# Pop Quiz

Q. Which library is primarily used for data manipulation in Python?

A

**Pandas**

B

**NumPy**

# Pop Quiz

Q. Which library is primarily used for data manipulation in Python?

A

Pandas

B

NumPy

PW SKILLS | DevOps and Cloud Computing

# Introducing NumPy for numerical processing

# NumPy

NumPy (Numerical Python) is a powerful library for handling large datasets, performing mathematical operations, and working with multidimensional arrays efficiently. It is widely used in data science, machine learning, and scientific computing.

1. Creating and Manipulating Arrays:

```
import numpy as np

# Create a 1D array
arr1 = np.array([1, 2, 3, 4, 5])
print("1D Array:", arr1)

# Create a 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print("\n2D Array:\n", arr2)

# Reshape array
reshaped = arr2.reshape(3, 2)
print("\nReshaped Array:\n", reshaped)
```

# NumPy

2. Basic Operations (Mean, Sum, etc.):

```
# Compute sum and mean
print("\nSum:", arr1.sum())
print("Mean:", arr1.mean())
print("Max:", arr1.max())
print("Min:", arr1.min())
```

3. Matrix Transformations (Transpose & Dot Product):

```
# Transpose of a matrix
transposed = arr2.T
print("\nTransposed Matrix:\n", transposed)

# Dot product of two matrices
mat1 = np.array([[1, 2], [3, 4]])
mat2 = np.array([[5, 6], [7, 8]])
dot_product = np.dot(mat1, mat2)
print("\nDot Product:\n", dot_product)
```

# Pop Quiz

Q. What is NumPy primarily used for?

A

**Numerical computing and array manipulation**

B

**Machine learning model training**

# Pop Quiz

Q. What is NumPy primarily used for?

A

**Numerical computing and array manipulation**

B

**Machine learning model training**

PW SKILLS | DevOps and Cloud Computing

# Discuss real-world automation scenarios

# Real-world automation scenarios

Automation plays a crucial role in reducing manual effort, increasing efficiency, and improving accuracy in various domains. Here are some key use cases:

1. Web Scraping for Data Collection:
Use Case: Businesses and researchers collect data from websites to analyze trends, monitor competitors, or gather public information.

Example:
- Extract product prices from e-commerce websites.
- Gather news articles from multiple sources for sentiment analysis.

Tools:
- BeautifulSoup and Requests for static pages.
- Selenium or Playwright for dynamic content.

# Real-world automation scenarios

2. Extracting Data from Reports & Transforming It into Structured Formats
Use Case: Companies automate the processing of reports (PDFs, CSVs, JSON) to extract relevant data and store it in databases.

Example:
- Convert financial statements from PDFs into structured Excel sheets.
- Extract log data and transform it into structured formats for analytics.

Tools:
- pandas for handling structured data.
- PyPDF2 or pdfplumber for extracting text from PDFs.
- openpyxl for working with Excel files.

# Real-world automation scenarios

3. Automating Repetitive Web Tasks (e.g., Form Submissions)
Use Case: Automating interactions with web applications, such as filling forms, logging into portals, or data entry.

Example:
- Auto-filling job application forms.
- Automating login processes and downloading reports.

Tools:
- ‘Selenium’ or ‘Playwright’ for browser automation.
- ‘pyautogui’ for GUI-based automation when a web API isn't available.

# Time for case study!

# Important

- **Complete the post-class assessment**
- **Complete assignments (if any)**
- **Practice the concepts and techniques taught in this session**
- **Review your lecture notes**
- **Note down questions and queries regarding this session and consult the teaching assistants**

# Thanks !