**Introduction to web scraping**

Web scraping, also known as web harvesting or web data extraction, is the process of extracting information from websites or web pages. It involves automated retrieval of data from web sources. People use it for various applications such as data analysis, mining, price comparison, content aggregation, and more.

**How web scraping works**

**HTTP request**

The process typically begins with an HTTP request. A web scraper sends an HTTP request to a specific URL, similar to how a web browser would when you visit a website. The request is usually an HTTP GET request, which retrieves the web page's content.

**Web page retrieval**

The web server hosting the website responds to the request by returning the requested web page's HTML content. This content includes the visible text and media elements and the underlying HTML structure that defines the page's layout.

**HTML parsing**

Once the HTML content is received, you need to parse the content. Parsing involves breaking down the HTML structure into components, such as tags, attributes, and text content. You can use BeautifulSoup in Python. It creates a structured representation of the HTML content that can be easily navigated and manipulated.

**Data extraction**

With the HTML content parsed, web scrapers can now identify and extract the specific data they need. This data can include text, links, images, tables, product prices, news articles, and more. Scrapers locate the data by searching for relevant HTML tags, attributes, and patterns in the HTML structure.

**Data transformation**

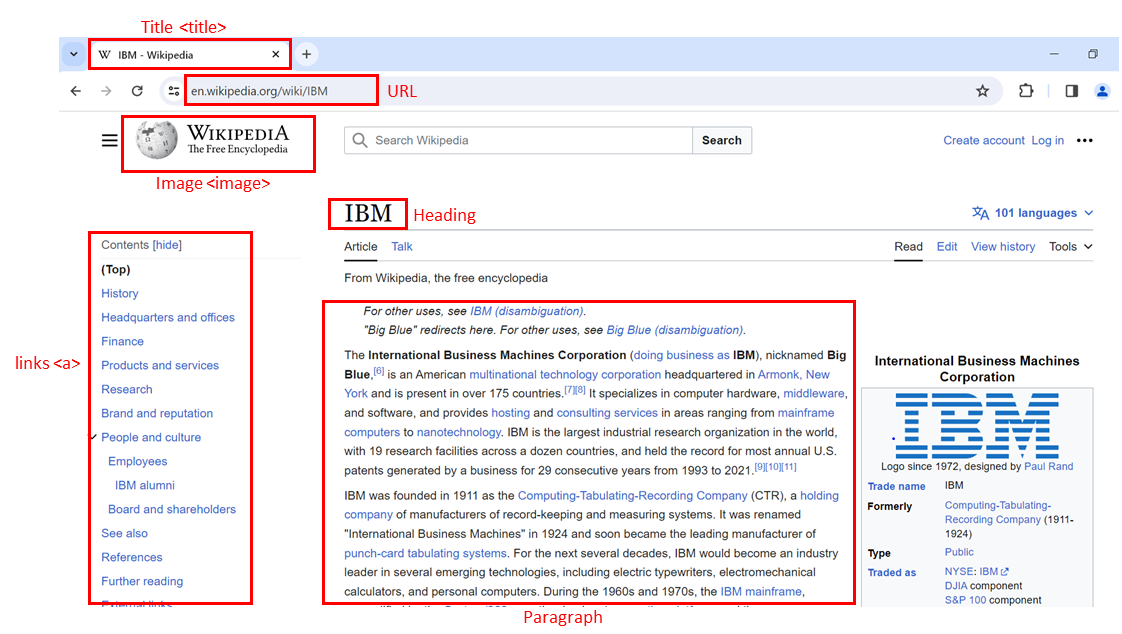
Extracted data may need further processing and transformation. For instance, you can remove HTML tags from text, convert data formats, or clean up messy data. This step ensures the data is ready for analysis or other use cases.

**Storage**

After extraction and transformation, you can store the scraped data in various formats, such as databases, spreadsheets, JSON, or CSV files. The choice of storage format depends on the specific project's requirements.

**Automation**

In many cases, scripts or programs automate web scraping. These automation tools allow recurring data extraction from multiple web pages or websites. Automated scraping is especially useful for collecting data from dynamic websites that regularly update their content.



**HTML structure**

Hypertext markup language (HTML) serves as the foundation of web pages. Understanding its structure is crucial for web scraping.

* <html> is the root element of an HTML page.
* <head> contains meta-information about the HTML page.
* <body> displays the content on the web page, often the data of interest.
* <h3> tags are type 3 headings, making text larger and bold, typically used for player names.
* <p> tags represent paragraphs and contain player salary information.

**Composition of an HTML tag**

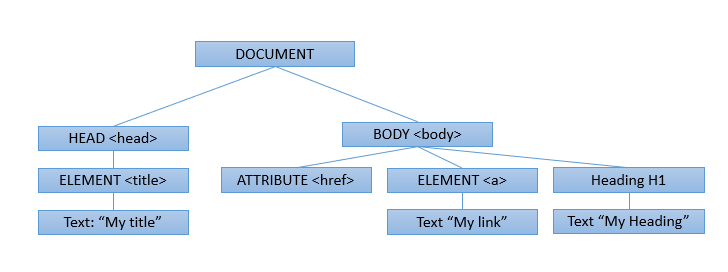
HTML tags define the structure of web content and can contain attributes.

* An HTML tag consists of an opening (start) tag and a closing (end) tag.
* Tags have names (<a> for an anchor tag).
* Tags may contain attributes with an attribute name and value, providing additional information to the tag.

**HTML document tree**

You can visualize HTML documents as trees with tags as nodes.

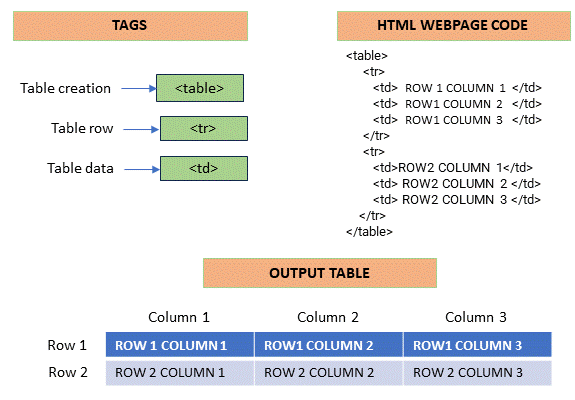
* Tags can contain strings and other tags, making them the tag's children.
* Tags within the same parent tag are considered siblings.
* For example, the <html> tag contains both <head> and <body> tags, making them descendants of <html but children of <html>. <head> and <body> are siblings.



**HTML tables**

HTML tables are essential for presenting structured data.

* Define an HTML table using the <table> tag.
* Each table row is defined with a <tr> tag.
* The first row often uses the table header tag, typically <th>.
* The table cell is represented by <td> tags, defining individual cells in a row.



**Web scraping**

Web scraping involves extracting information from web pages using Python. It can save time and automate data collection.

**Required tools**

Web scraping requires Python code and two essential modules: Requests and Beautiful Soup. Ensure you have both modules installed in your Python environment.

1. # Import Beautiful Soup to parse the web page content
2. from bs4 import BeautifulSoup

**Fetching and parsing HTML**

To start web scraping, you need to fetch the HTML content of a webpage and parse it using Beautiful Soup. Here's a step-by-step example:

1. import requests
2. from bs4 import BeautifulSoup
3. # Specify the URL of the webpage you want to scrape
4. url = 'https://en.wikipedia.org/wiki/IBM'
5. # Send an HTTP GET request to the webpage
6. response = requests.get(url)
7. # Store the HTML content in a variable
8. html\_content = response.text
9. # Create a BeautifulSoup object to parse the HTML
10. soup = BeautifulSoup(html\_content, 'html.parser')
11. # Display a snippet of the HTML content
12. print(html\_content[:500])

**Navigating the HTML structure**

BeautifulSoup represents HTML content as a tree-like structure, allowing for easy navigation. You can use methods like find\_all to filter and extract specific HTML elements. For example, to find all anchor tags () and print their text:

1. # Find all <a> tags (anchor tags) in the HTML
2. links = soup.find\_all('a')
3. # Iterate through the list of links and print their text
4. for the link in links:
5. print(link.text)

**Custom data extraction**

Web scraping allows you to navigate the HTML structure and extract specific information based on your requirements. This process may involve finding specific tags, attributes, or text content within the HTML document.

**Using BeautifulSoup for HTML parsing**

Beautiful Soup is a powerful tool for navigating and extracting specific web page parts. It allows you to find elements based on their tags, attributes, or text, making extracting the information you're interested in easier.

**Using pandas read\_html for table extraction**

Pandas, a Python library, provides a function called read\_html, which can automatically extract data from websites' tables and present it in a format suitable for analysis. It’s similar to taking a table from a webpage and importing it into a spreadsheet for further analysis.

**Introduction**

Web scraping, also known as web harvesting or web data extraction, is a technique used to extract large amounts of data from websites. The data on websites is unstructured, and web scraping enables us to convert it into a structured form.

**Importance of Web Scraping in Data Science**

In the field of data science, web scraping plays an integral role. It is used for various purposes such as:

1. **Data Collection:** Web scraping is a primary method of collecting data from the internet. This data can be used for analysis, research, etc.
2. **Real-time Application:** Web scraping is used for real-time applications like weather updates, price comparison, etc.
3. **Machine Learning:** Web scraping provides the data needed to train machine learning models.

**Web Scraping with Python**

Python provides several libraries for web scraping. Here are some of them:

1. **BeautifulSoup:** BeautifulSoup is a Python library used for web scraping purposes to pull the data out of HTML and XML files. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner.
2. from bs4 import BeautifulSoup
3. import requests
4. URL = "http://www.example.com"
5. page = requests.get(URL)
6. soup = BeautifulSoup(page.content, "html.parser")
7. **Scrapy:** Scrapy is an open-source and collaborative web crawling framework for Python. It is used to extract the data from the website.
8. import scrapy
9. class QuotesSpider(scrapy.Spider):
10. name = "quotes"
11. start\_urls = ['http://quotes.toscrape.com/tag/humor/',]
12. def parse(self, response):
13. for quote in response.css('div.quote'):
14. yield {'quote': quote.css('span.text::text').get()}
15. **Selenium:** Selenium is a tool used for controlling web browsers through programs and automating browser tasks.
16. from selenium import webdriver
17. driver = webdriver.Firefox()
18. driver.get("http://www.example.com")

**Applications of Web Scraping**

Web scraping is used in various fields and has many applications:

1. **Price Comparison:** Services such as ParseHub use web scraping to collect data from online shopping websites and use it to compare the prices of products.
2. **Email address gathering:** Many companies that use email as a medium for marketing, use web scraping to collect email ID and then send bulk emails.
3. **Social Media Scraping:** Web scraping is used to collect data from Social Media websites such as Twitter to find out what's trending.

**Web Scraping Tables using Pandas**

The Pandas library in Python contains a function read\_html() that can be used to extract tabular information from any web page.

Consider the following example:

Let us assume we want to extract the list of the largest banks in the world by market capitalization, from the following link:

1. URL = 'https://en.wikipedia.org/wiki/List\_of\_largest\_banks'

We may use pandas.read\_html() function in python to extract all the tables in the web page directly.

A snapshot of the webpage is shown below.  
A screenshot of a search engine

Description automatically generated

We can see that the required table is the first one in the web page.

*Note: This is a live web page and it may get updated over time. The image shown above has been captured in November 2023. The process of data extraction remains the same.*

We may execute the following lines of code to extract the required table from the web page.

1. import pandas as pd
2. URL = 'https://en.wikipedia.org/wiki/List\_of\_largest\_banks'
3. tables = pd.read\_html(URL)
4. df = tables[0]
5. print(df)

This will extract the required table as a dataframe df. The output of the print statement would look as shown below.

A computer screen shot of a black screen

Description automatically generated

Although convenient, this method comes with its own set of limitations.  
Firstly, web pages may have content saved in them as tables but they may not appear as tables on the web page.

For instance, consider the following URL showing the list of countries by GDP (nominal).

1. URL = 'https://en.wikipedia.org/wiki/List\_of\_countries\_by\_GDP\_(nominal)'

The images on the web page are also saved in tabular format. A snapshot of the web page is shared below.  
A screenshot of a computer screen

Description automatically generated

Secondly, the contents of the tables in the web pages may contain elements such as hyperlink text and other denoters, which are also scraped directly using the pandas method. This may lead to a requirement of further cleaning of data.  
A closer look at table 3 in the image shown above indicates that there are many hyperlink texts which are also going to be treated as information by the pandas function.

A screenshot of a spreadsheet

Description automatically generated

We can extract the table using the code shown below.

1. import pandas as pd
2. URL = 'https://en.wikipedia.org/wiki/List\_of\_countries\_by\_GDP\_(nominal)'
3. tables = pd.read\_html(URL)
4. df = tables(2) # the required table will have index 2
5. print(df)

The output of the print statement is shown below.



Note that the hyperlink texts have also been retained in the code output.

It is further prudent to point out, that this method exclusively operates only on tabular data extraction. BeautifulSoup library still remains the default method of extracting any kind of information from web pages.

* Simple APIs in Python are application programming interfaces that provide straightforward and easy-to-use methods for interacting with services, libraries, or data, often with minimal configuration or complexity.
  + An API lets two pieces of software talk to each other.
  + Using an API library in Python entails importing the library, calling its functions or methods to make HTTP requests, and parsing the responses to access data or services provided by the API.
  + Pandas API processes the data by communicating with the other software components.
  + An Instance forms when you create a dictionary and then use the DataFrames constructor to create a Pandas object.
  + Method “head()” will display the mentioned number of rows from the top (default 5) of DataFrames, while method “mean()” will calculate the mean and return the values
* Rest APIs allow you to communicate through the internet, taking advantage of resources like storage, access more data, AI algorithms, and so on.
  + HTTP methods transmit data over the internet.
  + An HTTP message typically includes a JSON file with instructions for operations.
  + HTTP messages containing JSON files are returned to the client as a response from web services.
  + Dealing with time series data involves using the Pandas time series function.
  + You can get data for daily candlesticks and plot the chart using Plotly with the candlestick plot.
* The HTTP (HyperText Transfer Protocol) transfers data, including web pages and resources, between a client (a web browser) and a server on the World Wide Web.
  + The HTTP protocol is commonly used for implementing various types of REST APIs.
  + An HTTP response includes information like the type of resource, length of resource, and so on
  + Uniform resource locator (URL) is the most popular way to find resources on the web.
  + URL is divided into three parts: scheme, internet address or base URL, and route
  + The GET method is one of the popular methods of requesting information. Some other methods may also include the body.
  + Response method contains the version and body of the response.
  + POST submits data to the server, PUT updates data already on the server, DELETE deletes data from the server
* Requests is a Python library that allows you to send HTTP/1.1 requests easily
  + You can modify the results of your query with the GET method.
  + You can obtain multiple requests from a URL like name, ID, and so on with a Query string.
* Web scraping in Python involves extracting and parsing data from websites to gather information for various applications, using libraries like Beautiful Soup and requests.
  + HTML comprises text surrounded by blue text elements enclosed in angular brackets called tags.
  + You can select an HTML element on a web page to inspect the webpage.
  + Web pages may also contain CSS and JavaScript along with HTML elements.
  + Each HTML document is like an HTML Tree, which may contain strings and other tags.
  + Each HTML table is comprised of table tags and is structured with elements such as rows, headers, body and so on.
* Tabular data can also be extracted from web pages using the `read\_html` method in Pandas.
* Beautiful Soup in Python is a library for parsing and navigating HTML and XML documents, making extracting, and manipulating data from web pages more accessible.
* To parse a document, pass it through the Beautiful Soup constructor to get a beautiful soup object representing the document as a nested data structure.
* Beautiful soup represents HTML as a set of tree-like objects with methods to parse the HTML.
* Navigable string is like a Python string that supports beautiful soup functionality.
* find\_all is a method used to extract content based on the tag’s name, its attributes, the text of a string, or some combination of these.
* The find\_all method looks through a tag’s descendants and retrieves all descendants that match your filters.
* The result is a Python iterable like a list.
* File formats refer to the specific structure and encoding rules used to store and represent data in files, such as .txt for plain text or .csv for comma-separated values.
* Python works with different file formats such as CSV, XML, JSON, xlsx, and so on
* The extension of a file name will let you know what type of file it is and what it needs to open with.
* To access data from CSV files, we can use Python libraries such as Pandas.
* Similarly, different methods help parse JSON, XML, and other files.

**Cheat Sheet: API's and Data Collection**

|  |  |  |
| --- | --- | --- |
| **Package/Method** | **Description** | **Code Example** |
| Accessing element attribute | Access the value of a specific attribute of an HTML element. | Syntax:   1. attribute = element[(attribute)]   Example:   1. href = link\_element[(href)] |
| BeautifulSoup() | Parse the HTML content of a web page using BeautifulSoup. The parser type can vary based on the project. | Syntax:   1. soup = BeautifulSoup(html, (html.parser))   Example:   1. html = (https://api.example.com/data) soup = BeautifulSoup(html, (html.parser)) |
| delete() | Send a DELETE request to remove data or a resource from the server. DELETE requests delete a specified resource on the server. | Syntax:   1. response = requests.delete(url)   Example:   1. response = requests.delete((https://api.example.com/delete)) |
| find() | Find the first HTML element that matches the specified tag and attributes. | Syntax:   1. element = soup.find(tag, attrs)   Example:   1. first\_link = soup.find((a), {(class): (link)}) |
| find\_all() | Find all HTML elements that match the specified tag and attributes. | Syntax:   1. elements = soup.find\_all(tag, attrs)   Example:   1. all\_links = soup.find\_all((a), {(class): (link)})</td> |
| findChildren() | Find all child elements of an HTML element. | Syntax:   1. children = element.findChildren()   Example:   1. child\_elements = parent\_div.findChildren() |
| get() | Perform a GET request to retrieve data from a specified URL. GET requests are typically used for reading data from an API. The response variable will contain the server's response, which you can process further. | Syntax:   1. response = requests.get(url)   Example:   1. response = requests.get((https://api.example.com/data)) |
| Headers | Include custom headers in the request. Headers can provide additional information to the server, such as authentication tokens or content types. | Syntax:   1. headers = {(HeaderName): (Value)}   Example:   1. base\_url = (https://api.example.com/data) headers = {(Authorization): (Bearer YOUR\_TOKEN)} response = requests.get(base\_url, headers=headers) |
| Import Libraries | Import the necessary Python libraries for web scraping. | Syntax:   1. from bs4 import BeautifulSoup |
| json() | Parse JSON data from the response. This extracts and works with the data returned by the API. The response.json() method converts the JSON response into a Python data structure (usually a dictionary or list). | Syntax:   1. data = response.json()   Example:   1. response = requests.get((https://api.example.com/data)) 2. data = response.json() |
| next\_sibling() | Find the next sibling element in the DOM. | Syntax:   1. sibling = element.find\_next\_sibling()   Example:   1. next\_sibling = current\_element.find\_next\_sibling() |
| parent | Access the parent element in the Document Object Model (DOM). | Syntax:   1. parent = element.parent   Example:   1. parent\_div = paragraph.parent |
| post() | Send a POST request to a specified URL with data. Create or update POST requests using resources on the server. The data parameter contains the data to send to the server, often in JSON format. | Syntax:   1. response = requests.post(url, data)   Example:   1. response = requests.post((https://api.example.com/submit), data={(key): (value)}) |
| put() | Send a PUT request to update data on the server. PUT requests are used to update an existing resource on the server with the data provided in the data parameter, typically in JSON format. | Syntax:   1. response = requests.put(url, data)   Example:   1. response = requests.put((https://api.example.com/update), data={(key): (value)}) |
| Query parameters | Pass query parameters in the URL to filter or customize the request. Query parameters specify conditions or limits for the requested data. | Syntax:   1. params = {(param\_name): (value)}   Example:   1. base\_url = "https://api.example.com/data" 2. params = {"page": 1, "per\_page": 10} 3. response = requests.get(base\_url, params=params) |
| select() | Select HTML elements from the parsed HTML using a CSS selector. | Syntax:   1. element = soup.select(selector)   Example:   1. titles = soup.select((h1)) |
| status\_code | Check the HTTP status code of the response. The HTTP status code indicates the result of the request (success, error, redirection). Use the HTTP status codeIt can be used for error handling and decision-making in your code. | Syntax:   1. response.status\_code   Example:   1. url = "https://api.example.com/data" 2. response = requests.get(url) 3. status\_code = response.status\_code |
| tags for find() and find\_all() | Specify any valid HTML tag as the tag parameter to search for elements of that type. Here are some common HTML tags that you can use with the tag parameter. | Tag Example:   1. - (a): Find anchor () tags. 2. - (p): Find paragraph ((p)) tags. 3. - (h1), (h2), (h3), (h4), (h5), (h6): Find heading tags from level 1 to 6 ( (h1),n (h2)). 4. - (table): Find table () tags. 5. - (tr): Find table row () tags. 6. - (td): Find table cell ((td)) tags. 7. - (th): Find table header cell ((td))tags. 8. - (img): Find image ((img)) tags. 9. - (form): Find form ((form)) tags. 10. - (button): Find button ((button)) tags. |
| text | Retrieve the text content of an HTML element. | Syntax:   1. text = element.text   Example:   1. title\_text = title\_element.text |

**Glossary: APIs and Data Collection**

Welcome! This alphabetized glossary contains many of the terms you'll find within this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are important for you to recognize when working in the industry, participating in user groups, and participating in other certificate programs.

| **Term** | **Definition** |
| --- | --- |
| API Key | An API key in Python is a secure access token or code used to authenticate and authorize access to an API or web service, enabling the user to make authenticated requests. |
| APIs | APIs (Application Programming Interfaces) are a set of rules and protocols that enable different software applications to communicate and interact, facilitating the exchange of data and functionality. |
| Audio file | An audio file is a digital recording or representation of sound, often stored in formats like MP3, WAV, or FLAC, allowing playback and storage of audio content. |
| Authorize | In Python, "authorize" often means granting permission or access to a user or system to perform specific actions or access particular resources, often related to authentication and authorization mechanisms. |
| Beautiful Soup Objects | Beautiful Soup objects in Python are representations of parsed HTML or XML documents, allowing easy navigation, searching, and manipulation of the document’s elements and data. |
| Bitcoin currency | Bitcoin is a decentralized digital currency that operates without a central authority, allowing peer-to-peer transactions on a blockchain network. |
| Browser | A browser is a software application that enables users to access and interact with web content, displaying websites and web applications. |
| Candlestick plot | A candlestick plot in Python visually represents stock price movements over time, using rectangles to illustrate the open, close, high, and low prices for a given period. |
| Client/Wrapper | A client or wrapper in Python is a software component that simplifies interaction with external services or APIs, encapsulating communication and providing higher-level functionality for developers. |
| CoinGecko API | The CoinGecko API is a web service that provides cryptocurrency market data and information, allowing developers to access real-time and historical data for various cryptocurrencies. |
| DELETE Method | The DELETE method in Python is an HTTP request method used to request the removal or deletion of a resource on a web server. |
| Endpoint | In Python, an "endpoint" refers to a specific URL or URI that a web service or API exposes to perform a particular function or access a resource. |
| File extension | A file extension is a suffix added to a filename to indicate the file's format or type, often used by operating systems and applications to determine how to handle the file. |
| find\_all | In Python, find\_all is a Beautiful Soup method used to search and extract all occurrences of a specified HTML or XML element, returning a list of matching elements. |
| GET method | The GET method in Python is an HTTP request method used to retrieve data from a web server by appending parameters to the URL. |
| HTML | HTML (Hypertext Markup Language) is the standard language for creating and structuring content on web pages, using tags to define the structure and presentation of documents. |
| HTML Anchor tags | HTML anchor tags in Python are used to create hyperlinks within web pages, linking to other web pages or resources using the <a> element with the href attribute. |
| HTML Tables | HTML tables in Python are used to organize and display data in a structured grid format on a web page, constructed with <table>, <tr>, <th>, and <td> elements. |
| HTML Tag | An HTML tag in Python is a specific code enclosed in angle brackets used to define elements within an HTML document, specifying how content should be presented or structured. |
| HTML Trees | HTML trees in Python refer to the hierarchical structure created when parsing an HTML document, representing its elements and their relationships, typically used for manipulation or extraction of data. |
| HTTP | HTTP (HyperText Transfer Protocol) is the foundation of data communication on the World Wide Web, used for transmitting and retrieving web content between clients and servers. |
| httplib | A library that provides a set of functions and classes to send and handle HTTP and HTTPS requests. |
| Identify | In Python, "identify" usually means determining if two variables or objects refer to the same memory location, which can be checked using the is operator. |
| Instance | In Python, an "instance" typically refers to a specific occurrence of an object or class, created from a class blueprint, with its own unique set of data and attributes. |
| JSON file | A JSON (JavaScript Object Notation) file is a lightweight data interchange format that stores structured data in a human-readable text format, commonly used for configuration, data exchange, and web APIs. |
| Mean value | The mean value in Python is the average of a set of numerical values, calculated by adding all values and dividing by the total number of values. |
| Navigable string | In Python, a Navigable String is a Beautiful Soup object representing a string within an HTML or XML document, allowing for navigation and manipulation of the text content. |
| Plotly | Plotly is a Python library for creating interactive and visually appealing web-based data visualizations and dashboards. |
| PNG file | A PNG (Portable Network Graphics) file is a lossless image format in Python that is commonly used for high-quality graphics with support for transparency and compression. |
| POST method | The POST method in Python is an HTTP request method used to send data to a web server, often used for submitting form data and creating or updating resources. |
| Post request | A POST request in Python is an HTTP method used to send data to a web server for the purpose of creating or updating a resource, typically used in web applications and APIs. |
| PUT method | The PUT method in Python is an HTTP request method used to update an existing resource on a web server by replacing or modifying it. |
| Py-Coin-Gecko | Py-Coin-Gecko is a Python library that provides a convenient interface for accessing cryptocurrency data and information from the CoinGecko API. |
| Python iterable | A Python iterable is an object that can be looped over, typically used in for loops, and includes data structures like lists, tuples, and dictionaries. |
| Query string | A query string in Python is a part of a URL that contains data or parameters to be sent to a web server, typically used in HTTP GET requests to retrieve specific information. |
| rb mode | In Python, "rb" mode is used when opening a file to read it in binary mode, allowing you to read and manipulate non-text files like images or binary data. |
| Resource | In Python, a "resource" typically refers to an external entity such as a file, database connection, or network object that can be managed and manipulated within a program. |
| Rest API | A REST API in Python is a web-based interface that follows the principles of Representational State Transfer (REST), allowing communication and data exchange over HTTP using standard HTTP methods and data formats. |
| Service instance | In Python, a "service instance" typically refers to an instantiated object or entity representing a service, enabling interaction with that service in a program or application. |
| Timestamp | A timestamp is a representation of a specific moment in time, often expressed as a combination of date and time, used for record-keeping and data tracking. |
| Transcribe | "Transcribe" typically means converting spoken language or audio into written text, often using automatic speech recognition (ASR) technology. |
| Unix timestamp | A UNIX timestamp is a numerical value representing the number of seconds that have elapsed since January 1, 1970, 00:00:00 UTC, used for time-keeping in Unix-based systems and programming. |
| url (Uniform Resource Locator) | In Python, a URL (Uniform Resource Locator) is a web address that specifies the location of a resource on the internet, typically consisting of a protocol, domain, and path. |
| urllib | The "urllib" library in Python is used for working with URLs and making HTTP requests, including functions for fetching web content, handling cookies, and more. |
| Web service | Web services in Python are software components that allow applications to communicate over the internet by sending and receiving data in a standardized format, typically using protocols like HTTP or XML. |
| Web scraping | Web scraping in Python is the process of extracting data from websites by parsing and analyzing their HTML structure, often done with libraries like BeautifulSoup or Scrapy. |
| xlsx | An XLSX file is a file format used for storing spreadsheet data in Excel, containing worksheets, cells, and formulas in a structured manner. |
| xml | XML (Extensible Markup Language) is a text-based format for storing and structuring data using tags, often used for data interchange and configuration files. |