UNIT -1 (Python libraries for web development)

1. Collections-Container datatypes :-

"Collections is a built-in python module that provides useful container datatypes. Container datatypes allow us to store and access values in a convenient way. Generally, you would have used lists, tuples, and dictionaries. But, while dealing with structured data we need smarter objects. So I will walk you through the different data structures supported by collections module."

(i) Namedtuple:-

Named tuple container datatype is an alternative to the built-in tuple. This extension type enhances standard tuples so that their elements can be accessed by both their attribute name and the positional index.

"A key difference between a tuple and a namedtuple is: while a tuple let's you access data through indices, with a nametuple you can access the elements with their names."

Example:-

```
from collections import namedtuple
movie = namedtuple('movie','genre rating lead_actor')

# Create instances of movie
ironman = movie(genre='action',rating=8.5,lead_actor='robert downey junior')

# Access the fields
print(ironman.rating)
```

```
#> 8.5
```

(ii) Counter:-

A counter object is provided by the collections library. Counter allows you to compute the frequency easily. It works not just for numbers but for any iterable object, like strings and lists. Counter is dict subclass, used to count objects.It returns a dictionary with the elements as keys and the count as values

EXAMPLES (1) :-

```
#importing Counter from collections

from collections import Counter

numbers = [4,5,5,2,22,2,2,1,8,9,7,7]

num_counter = Counter(numbers)

print(num_counter)

Output:-

#>Counter({2: 3, 5: 2, 7: 2, 4: 1, 22: 1, 1: 1, 8: 1, 9: 1})
```

EXAMPLES (2) :-

```
#counter with strings

string = 'lalalalandismagic'

string_count = Counter(string)

print(string_count)

Output:-

#> Counter({'a': 5, 'l': 4, 'i': 2, 'n': 1, 'd': 1, 's': 1, 'm': 1, 'g':
1, 'c': 1})
```

(iii) defaultdict :-

The Python defaultdict() is defined as a dictionary-like object. It is a subclass of the built-in dict class. If you try to access a key that is not present in a dictionary, it throws a KeyError. Whereas, in a defaultdict it does not give a KeyError ,the defaultdict will return a default value.

Example:-

```
# Creating a defaultdict and trying to access a key that is not present.

from collections import defaultdict

def_dict = defaultdict(object)

def_dict['fruit'] = 'orange'

def_dict['drink'] = 'pepsi'

print(def_dict['chocolate'])

Output:-

#> &lt;object object at 0x7f591a2f4510&gt;
```

(iv) OrderedDict :-

The Python OrderedDict() is similar to a dictionary object where keys maintain the order of insertion. If we try to insert key again, the previous value will be overwritten for that key. A dict is an UNORDERED collection of key value pairs. But, an OrderedDict maintains the ORDER in which the keys are inserted.It is subclass of dict.

Example:-

```
# Create an OrderedDict and print items

from collections import OrderedDict

ordered_vehicle=OrderedDict()

ordered_vehicle['bicycle']='hercules'

ordered_vehicle['car']='Maruti'
```

```
ordered_vehicle['bike']='Harley'

print('This is an ordered dict')



for key,value in ordered_vehicle.items():

    print(key,value)

Output:-

#> This is an ordered dict

#> bicycle hercules

#> car Maruti

#> bike Harley
```

(v) ChainMap:-

ChainMap is a container datatype which stores multiple dictionaries. A chainmap class is used to groups multiple dictionary together to create a single list. You can print all the items in a `ChainMap` using `.map` operator. Below code demonstrates the same.

Example:-

```
# Creating a ChainMap from 3 dictionaries.

from collections import ChainMap

dic1={'red':5,'black':1,'white':2}

dic2={'chennai':'tamil','delhi':'hindi'}

dic3={'firstname':'bob','lastname':'mathews'}
```

```
my_chain = ChainMap(dic1,dic2,dic3)

my_chain.maps

Output:-

#> [{'black': 1, 'red': 5, 'white': 2}, {'chennai': 'tamil', 'delhi':
'hindi'},{'firstname': 'bob', 'lastname': 'mathews'}]
```

(vi) UserList:-

The UserList behaves as a wrapper class around the list-objects. It is useful when we want to add new functionality to the lists. It provides the easiness to work with the dictionary. This list is stored in the data attribute and can be accessed through `UserList.data` method.

Example:-

```
# Creating a user list with argument my_list

from collections import UserList

my_list=[11,22,33,44]



# Accessing it through `data` attribute

user_list=UserList(my_list)

print(user_list.data)

Output:-

#> [11, 22, 33, 44]
```

What is the use of UserLists :-

"Suppose you want to double all the elements in some particular lists as a reward. Or maybe you want to ensure that no element can be deleted from a given list.In such cases, we need to add a certain 'behavior' to our lists, which can be done using UserLists."

(vii) UserString:-

The UserString behaves as a wrapper class around the list objects. The dictionary can be accessed as an attribute by using the UserString object. It provides the easiness to work with the dictionary. It allows you to add certain functionality/behavior to the string. You can pass any string convertible argument to this class and can access the string using the data attribute of the class.

Example:-

```
# import Userstring

from collections import UserString

num=765



# passing an string convertible argument to userdict

user_string = UserString(num)



# accessing the string stored

user_string.data



Output:-#> '765
```

(viii) UserDict :-

The UserDict behaves as a wrapper around the dictionary objects. The dictionary can be accessed as an attribute by using the UserDict object. It provides the easiness to work with the dictionary. UserDict allows you to create a dictionary modified to your needs.

Example:-

```
# importing UserDict

from collections import UserDict

my_dict={'red':'5','white':2,'black':1}



# Creating an UserDict

user_dict = UserDict(my_dict)

print(user_dict.data)

Output:-

#> {'red': '5', 'white': 2, 'black': 1}
```

"All above We discuss the container datatypes from the collections module. They increase efficiency by a great amount when used on large datasets."

2. Tkinter-GUI applications, Requests-HTTP, requests, BeautifulSoup4-web scraping, Scrapy :-

(i) Tkinter-GUI applications:-

Python provides the standard library Tkinter for creating the graphical user interface for desktop based applications.

Developing desktop based applications with python Tkinter is not a complex task. An empty Tkinter top-level window can be created by using the following steps.

1. Import the Tkinter module.
2. Create the main application window.
3. Add the widgets like labels, buttons, frames, etc. to the window.
4. Call the main event loop so that the actions can take place on the user's computer screen.

Example

```
1. from tkinter import *
2. #creating the application main window.
3. top = Tk()
4. #Entering the event main loop
5. top.mainloop()
```

Output:

(ii) Tkinter widgets:-

There are various widgets like button, canvas, checkbutton, entry, etc. that are used to build the python GUI applications.

SN	Widget	Description
1	Button	The Button is used to add various kinds of buttons to the python application.
2	Canvas	The canvas widget is used to draw the canvas on the window.
3	Checkbutton	The Checkbutton is used to display the CheckButton on the window.
4	Entry	The entry widget is used to display the single-line text field to the user. It is commonly used to accept user values.
5	Frame	It can be defined as a container to which, another widget can be added and organized.
6	Label	A label is a text used to display some message or information about the other widgets.
7	ListBox	The ListBox widget is used to display a list of options to the user.
8	Menubutton	The Menubutton is used to display the menu items to the user.
9	Menu	It is used to add menu items to the user.
10	Message	The Message widget is used to display the message-box to the user.

(iii) Python Tkinter Geometry :-

The Tkinter geometry specifies the method by using which, the widgets are represented on display. The python Tkinter provides the following geometry methods.

1. The pack() method
2. The grid() method
3. The place() method

Python Tkinter pack() method

The pack() widget is used to organize widget in the block. The positions widgets added to the python application using the pack() method can be controlled by using the various options specified in the method call.

A list of possible options that can be passed in pack() is given below.

- **expand:** If the expand is set to true, the widget expands to fill any space.
- **Fill:** By default, the fill is set to NONE. However, we can set it to X or Y to determine whether the widget contains any extra space.
- **size:** it represents the side of the parent to which the widget is to be placed on the window.

Python Tkinter grid() method:-

The grid() geometry manager organizes the widgets in the tabular form. We can specify the rows and columns as the options in the method call. We can also specify the column span (width) or rowspan(height) of a widget.This is a more organized way to place the widgets to the python application.

A list of possible options that can be passed inside the grid() method is given below.

- **Column**
  The column number in which the widget is to be placed. The leftmost column is represented by 0.
- **Columnspan**
  The width of the widget. It represents the number of columns up to which, the column is expanded.
- **row**
  The row number in which the widget is to be placed. The topmost row is represented by 0.
- **rowspan**
  The height of the widget, i.e. the number of the row up to which the widget is expanded.

Python Tkinter place() method:-

The place() geometry manager organizes the widgets to the specific x and y coordinates.

A list of possible options is given below.

- **Anchor:** It represents the exact position of the widget within the container. The default value (direction) is NW (the upper left corner)
- **Border mode:** The default value of the border type is INSIDE that refers to ignore the parent's inside the border. The other option is OUTSIDE.
- **height, width:** It refers to the height and width in pixels.
- **x, y:** It refers to the horizontal and vertical offset in the pixels.

Calculator Application using Tkinter :-

```
from tkinter import *

win = Tk() # This is to create a basic window
win.geometry("312x324")  # this is for the size of the window
win.resizable(0, 0)  # this is to prevent from resizing the window
win.title("Calculator")
```

```
###################Starting with functions ####################
# 'btn_click' function :
# This Function continuously updates the
# input field whenever you enter a number

def btn_click(item):
    global expression
    expression = expression + str(item)
    input_text.set(expression)
```

```
# 'bt_equal':This method calculates the expression
# present in input field

def bt_equal():
    global expression
    result = str(eval(expression)) # 'eval':This function is used to evaluates
    input_text.set(result)
    expression = ""

expression = ""
```

```
# 'bt_clear' function :This is used to clear
# the input field

def bt_clear():
    global expression
    expression = ""
    input_text.set("")
```

```
# 'StringVar()' :It is used to get the instance of input field

input_text = StringVar()

# Let us creating a frame for the input field

input_frame = Frame(win, width=312, height=50, bd=0, highlightbackground="black", highlightcolor="black", highlightthickness=2)

input_frame.pack(side=TOP)

#Let us create a input field inside the 'Frame'

input_field = Entry(input_frame, font=('arial', 18, 'bold'), textvariable=input_text, width=50, bg="#eee", bd=0, justify=RIGHT)

input_field.grid(row=0, column=0)

input_field.pack(ipady=10) # 'ipady' is internal padding to increase the height of input field

#Let us creating another 'Frame' for the button below the 'input_frame'
```

```
btns_frame = Frame(win, width=312, height=272.5, bg="grey")

btns_frame.pack()

# first row

clear = Button(btns_frame, text = "C", fg = "black", width = 32, height = 3, bd =

divide = Button(btns_frame, text = "/", fg = "black", width = 10, height = 3, bd

# second row

seven = Button(btns_frame, text = "7", fg = "black", width = 10, height = 3, bd =

eight = Button(btns_frame, text = "8", fg = "black", width = 10, height = 3, bd =

nine = Button(btns_frame, text = "9", fg = "black", width = 10, height = 3, bd =

multiply = Button(btns_frame, text = "*", fg = "black", width = 10, height = 3, b

# third row
```

```
= 0, bg = "#eee", cursor = "hand2", command = lambda: bt_clear()).grid(row = 0
 = 0, bg = "#eee", cursor = "hand2", command = lambda: btn_click("/")).grid(ro


= 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(7)).grid(row =
= 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(8)).grid(row =
 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(9)).grid(row =
od = 0, bg = "#eee", cursor = "hand2", command = lambda: btn_click("*")).grid(
```

```
 0, column = 0, columnspan = 3, padx = 1, pady = 1)
row = 0, column = 3, padx = 1, pady = 1)


 = 1, column = 0, padx = 1, pady = 1)
 = 1, column = 1, padx = 1, pady = 1)
= 1, column = 2, padx = 1, pady = 1)
d(row = 1, column = 3, padx = 1, pady = 1)
```

*** FOR BETTER UNDERSTANDING OF CALCULATOR GO THROUGH THIS LINK**

https://www.studytonight.com/tkinter/calculator-application-using-tkinter

(iv) Python Packages :-

Suppose you have developed a very large application that includes many modules. As the number of modules grows, it becomes difficult to keep track of them all if they are dumped into one location. This is particularly so if they have similar names or functionality. You might wish for a means of grouping and organizing them.
Packages allow for a hierarchical structuring of the module namespace using dot notation. In the same way that modules help avoid collisions between global variable names, packages help avoid collisions between module names. Creating a package is quite straightforward, since it makes use of the operating system's inherent hierarchical file structure. Consider the following arrangement:

Here, there is a directory named pkg that contains two modules, mod1.py and mod2.py.

1. Import pkg.mod1, pkg.mod2
2. from pkg import mod1

Package Initialization

If a file named __init__.py is present in a package directory, it is invoked when the package or a module in the package is imported. This can be used for execution of package initialization code, such as initialization of package-level data.

For example, consider the following __init__.py file:

Top 20 Python Libraries/Packages:

- TensorFlow
- NumPy
- SciPy
- Pandas
- Matplotlib
- Keras
- SciKit-Learn
- PyTorch
- Scrapy
- BeautifulSoup

(V) FAST API

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

The key features are:

- Fast: Very high performance, on par with Nodejs and Go . One of the fastest Python frameworks available.
- Fast to code: Increase the speed to develop features by about 200% to 300%.
- Fewer bugs: Reduce about 40% of human (developer) induced errors.
- Intuitive: Great editor support. Completion everywhere. Less time debugging.
- Easy: Designed to be easy to use and learn. Less time reading docs.
- Short: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
- Robust: Get production-ready code. With automatic interactive documentation.
- Standards-based: Based on (and fully compatible with) the open standards for APIs: OpenAPI (previously known as Swagger) and JSON Schema.

Installation

pip install fastapi

You will also need an ASGI server, for production such as Uvicorn or Hypercorn.

pip install uvicorn[standard]

Simple example

- Create a file main.py with:

```
from typing import Optional

from fastapi import FastAPI

app = FastAPI()


@app.get("/")
```

```
def read_root():
    return {"Hello": "World"}


@app.get("/items/{item_id}")

def read_item(item_id: int, q: Optional[str] = None):
    return {"item_id": item_id, "q": q}
```

- Run the server:

```
uvicorn main:app --reload
```

- Open your browser at http://127.0.0.1:8000/items/5?q=somequery. You will see the JSON response as:

```
{
  "item_id": 5,
  "q": "somequery"
}
```

You already created an API that:

- Receives HTTP requests in the paths / and /items/{item_id}.
- Both paths take GET operations (also known as HTTP methods).
- The path /items/{item_id} has a path parameter item_id that should be an int.
- The path /items/{item_id} has an optional str query parameter q.
- Has interactive API docs made for you:
- Swagger: http://127.0.0.1:8000/docs.
- Redoc: http://127.0.0.1:8000/redoc.

You will see the automatic interactive API documentation (provided by Swagger UI):

Sending data to the server

When you need to send data from a client (let's say, a browser) to your API, you have three basic options:

- As path parameters in the URL (/items/2).
- As query parameters in the URL (/items/2? skip=true).
- In the body of a POST request.

To send simple data use the first two, to send complex or sensitive data, use the last.

It also supports sending data through cookies and headers.

Path Parameters

You can declare path "parameters" or "variables" with the same syntax used by Python format strings:

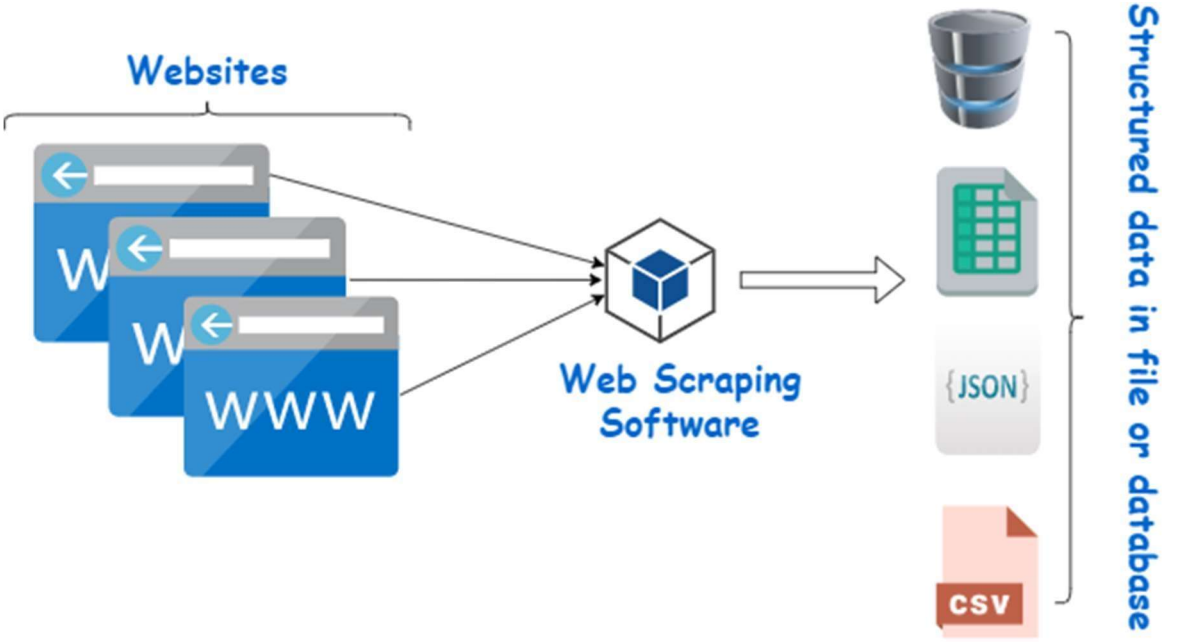
```
@app.get("/items/{item_id}")
def read_item(item_id: int):
    return {"item_id": item_id}
```

If you define the type hints of the function arguments, FastAPI will use pydantic data validation.

(vi) Web Scrapping:-

"Web Scraping (also termed Screen Scraping, Web Data Extraction, Web Harvesting etc.) is a technique used to automatically extract large amounts of data from websites and save it to a file or database. The data scraped will usually be in tabular or spreadsheet format"

Data displayed by websites can only be viewed using a web browser. Most websites do not allow you to save or download this data. If you need the data, the only option is to manually copy and paste the data - a very tedious job which can take many hours or days to complete. Web Scraping is the technique of automating this process, so that instead of manually copying the data from websites, the Web Scraping software will perform the same task within a fraction of the time.



A web scraping software will automatically load, crawl and extract data from multiple pages of websites based on your requirement.

With the click of a button, you can easily save the data displayed by websites to a file in your computer.

What is Web Scraping used for:-

Web Scraping is used for getting data. Access to relevant data, having methods to analyze it and performing intelligent actions based on analysis can make a huge difference in the success and growth of most businesses in the modern world. Data collection and analysis is important even for government, non-profit and educational institutions.

The following are few of the many uses of Web Scraping:

1. In e-commerce, Web Scraping is used for competition price monitoring.

2. In Marketing, Web Scraping is used for lead generation, to build phone and email lists for cold outreach.

3. In Real Estate, Web Scraping is used to get property and agent/owner details.

4. Web Scraping is used to collect training and testing data for Machine Learning projects.

Web Scraping Applications :-

There are numerous applications of web scraping under this segment which can be sub-divided into different categories as follows:-

1. Web Scraping Applications in Retail and Manufacturing
2. Web Scraping Applications in Equity and Financial Research
3. Web Scraping Applications in Data Science
4. Web Scraping Applications in Risk Management
5. Web Scraping Applications in Product, Marketing and Sales
6. News and Reputation Monitoring
7. Data Journalism ,Academic and Employment
8. Content Marketing and Competitive Analysis
9. Machine Learning Training Models
10. Natural Language Processing

Web Scraping Process:-

Steps involved in web scraping are:

1. Identify the target website
2. Collect URLs of the pages where you want to extract data from
3. Make a request to these URLs to get the HTML of the page
4. Use locators to find the data in the HTML
5. Save the data in a JSON or CSV file or some other structured format

(vii) BeautifulSoup4:-

1. Beautiful Soup is a pure Python library for extracting structured data from a website. It allows you to parse data from HTML and XML files.
2. It was first introduced by **Leonard Richardson**, who is still contributing to this project
3. It usually saves programmers hours or days of work since it works with your favorite parsers like lxml and html5lib to provide organic Python ways of navigating, searching, and modifying the parse tree.
4. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner.
5. As a developer, you do not have to take care of that unless the document intrinsic doesn't specify an encoding or Beautiful Soup is unable to detect one.
6. It is also considered to be faster when compared to other general parsing or scraping techniques.

"It is a library that allows you to efficiently and easily pull out information from HTML. In the real world, it is often used for web scraping projects".

Advantage:-

1. Very fast
2. Extremely lenient
3. Parses pages the same way a Browser does
4. Prettify the Source Code

Features of Beautiful Soup:-

1. This Python library provides a few simple methods, as well as Pythonic idioms for navigating, searching, and modifying a parse tree

2. The library automatically converts incoming and outgoing documents to Unicode and UTF-8, respectively

3. This library sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility

Example of BeautifulSoup:-

We can install the package by doing a simple **pip install beautifulsoup4** we will refer to BeautifulSoup4 as "BS4".

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(response.content, 'html.parser')
```

This soup object is very handy and allows us to easily access many useful pieces of information.

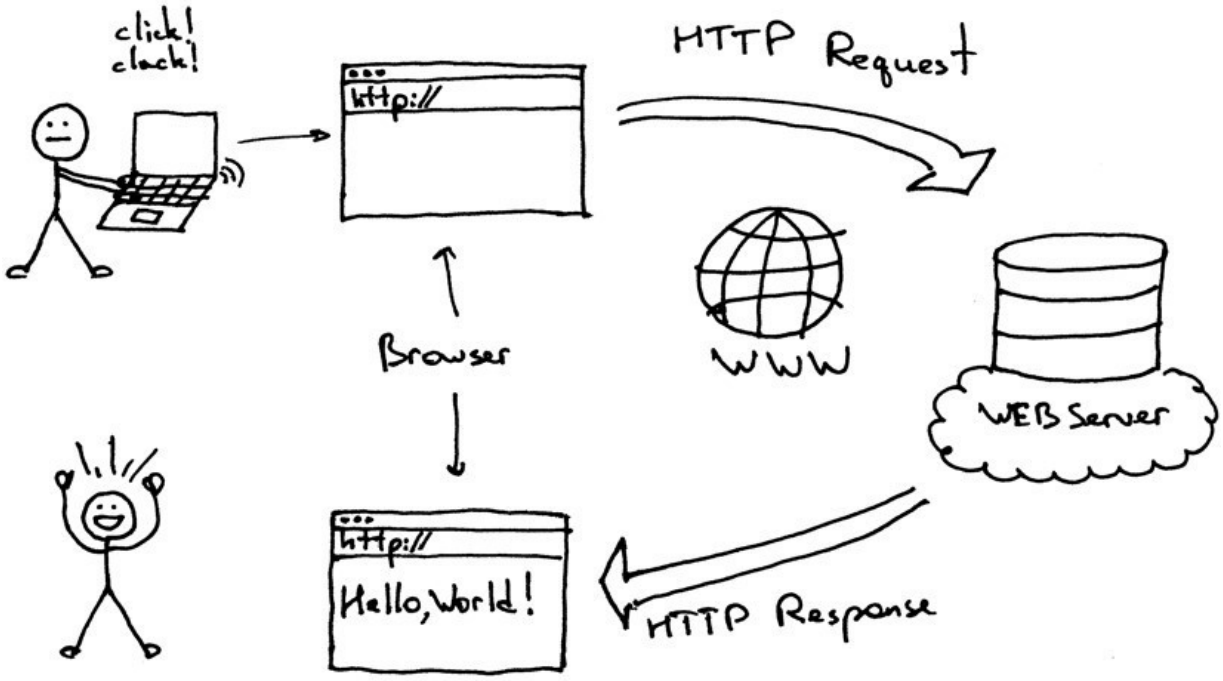
```
# The title tag of the page
print(soup.title)
> <title>Hacker News</title>

# The title of the page as string
print(soup.title.string)
> Hacker News

# All links in the page
nb_links = len(soup.find_all('a'))
print(f"There are {nb_links} links in this page")
> There are 231 links in this page

# Text from the page
print(soup.get_text())
> Hacker News
> Hacker News
> new | past | comments | ask | show | jobs | submit
> login
> ...
```

(v) Requests-HTTP, requests:-



Example:-So, for starters, we need an HTML document. For that purpose, we will be using Python's Requests package and fetch the main page

```
import requests
response = requests.get("https://news.ycombinator.com/")
if response.status_code != 200:
    print("Error fetching page")
    exit()
else:
    content = response.content
print(content)

> b'<html op="news"><head><meta name="referrer" content="origin"><meta name="vie
> content="width=device-width, initial-scale=1.0"><link rel="stylesheet" type="t
```

(viii) Zappa :-

1. Zappa is a very lightweight and powerful tool which allows you to deploy and update python applications to AWS. Using Zappa you can host your WSGI app on AWS Lambda, API Gateway quickly

2. Zappa is a system for running “serverless” Python web applications using AWS Lambda and AWS API Gateway. It handles all of the configuration & deployment automatically.

3. Now it is easy to deploy an infinitely scalable application to the cloud with a just single command at the least possible cost often just a small fraction of the cost of a traditional web server.

4. Zappa is an open source tool that was developed and designed by **Rich Jones**.

5. Zappa makes it super easy to build and deploy all Python WSGI applications on AWS Lambda + API Gateway. Lambda is the Serverless platform by AWS. It supports Python, NodeJS, Java, and .NET Core

Why do we need Zappa:-

1. Generally, we prefer to use a Virtual Machine to host our python application and the VM is costly compared to serverless technology because it will be up and running all the time and the cloud provider is billing us as per run time. Pricing based on the amount of computing power, storage, and time.
2. Infinite scaling
3. Zero downtime
4. Zero maintenance

What is a serverless architecture:-

A serverless architecture is a way to build and run applications and services without having to manage infrastructure. Your application still runs on servers, but all the server management is done by AWS. You no longer have to provision, scale, and maintain servers to run your applications, databases, and storage systems. Learn more about serverless computing.

Why use serverless architectures:-

By using a serverless architecture, your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.

Core functions of Zappa:

Zappa has two core functions:

1. **Code Packaging**: It takes a legacy python web application and creates a Lambda compatible code package. Internally, it creates a lambda function that wraps the original application. Inside the Lambda function it parses the trigger and routes it to the right web API in the application. For example it hides all the REST endpoints defined in a flask App and instead exposes only the lambda function it has created. However, when the request is received, the function parses the request to route it to the appropriate flask API as an inline function call.

2. **AWS Infrastructure Management**: It automatically uploads packages to S3, creates IAM roles, policies, API Gateway configuration. However, it needs practically administrator level access to the user-provided AWS account. Further, it is a CLI tool so it has no notion of multi-developer isolation workspaces. DevOps tasks like integration of the Lambda deployment with rest of the application topology (like databases, ELB, data strores like DynamoDB) or other parts of the organization's AWS deployment are out-of-scope.

Benefits of Zappa:

1. No more tedious web server configuration!
2. No more paying for 24/7 server uptime!
3. No more worrying about load balancing / scalability!
4. No more worrying about web server security!
5. Scalability since we can have as many responses processed in parallel as we need. Since AWS Lambda handles all of the requests
6. 100 function executions per second, and if you scale beyond that you only need to ask Amazon to raise your limit.
7. Inexpensive since we pay by the millisecond. Not to mention the cost saving on not having to spend time on deployment, operations and maintenance!
8. Maintainability and Ease of Use
9. Easy to deploy.- deployed in a single command

Challenges of Zappa:-

Integration with the other services of the cloud platform is often said to raise an issue.

(ix) Dash:-

- Dash is a python framework created by plotly for creating interactive web applications. Dash is written on the top of Flask, Plotly.js and React.js

- There are 2 different versions of the Dash framework available on the market today — 'Dash Open Source' and 'Dash Enterprise'

- Dash is a language-agnostic framework — with support for the Python, R, and Julia programming languages.

- There are 2 main libraries which make up the contents of your application, the '**dash_core_components'** and '**dash_html_components'** libraries. Both of these libraries contain Python classes so it is possible for programmers with no web development experience to program using Dash, however, HTML, CSS, and even JavaScript code can also be used in optional parameters within these Python classes

- Dash is an open source framework for building data visualization interfaces. Released in 2017 as a Python library, it's grown to include implementations for R and Julia.

- Dash is easy to use in a basic fashion to create dashboard applications

- Dash helps data scientists build analytical web applications without requiring advanced web development knowledge.

- Dash will help you build dashboards quickly. If you're used to analyzing data or building data visualizations using Python, then Dash will be a useful

- Three technologies constitute the core of Dash:
  Flask supplies the web server functionality.
  React.js renders the user interface of the web page.
  Plotly.js generates the charts used in your application.

Building blocks of Dash:-

Dash applications are made up of 2 building blocks :

1. Layout
2. Callbacks

Layout describes the look and feel of the app, it defines the elements such as graphs, dropdowns etc. and the placement, size, color etc. of these elements.

Dash contains **Dash HTML components** using which we can create and style HTML content such as headings, paragraph, and images etc using python. Elements such as graphs,dropdowns, sliders are created using **Dash Core components.**

Callbacks are used to bring interactivity to the dash applications. These are the functions using which, we can define the activity that would happen on clicking a button or a dropdown.

(X) CherryPy:-

1.It is one of the oldest Python web-framework. It is also a lightweighted Python framework with few dependencies.

2. CherryPy follows Object Orientation approach, it made it easy for developers to write and wrap web logic around HTTP protocols.

3. The simplicity is the main asset of CherryPy, and it can use Object Relational Mapper (ORM), and templating languages extensions for database and templating.

4. CherryPy is designed on the concept of multithreading. This gives it the benefit of handling multiple tasks at the same time.

5. It also takes a modular approach, following the Model View Controller (MVC) pattern to buildweb services; therefore, it's fast and developer-friendly.

6.The concept of an object-oriented engine forms the core part of CherryPy's working mechanism.

Features of CherryPy Framework :-

1. It is one of the simplest Python web frameworks.
2. It follows a modular approach to write web app logic.
3. It comes with common web-app building tools like caching, encoding, session, authorizations,static content, and many more.
4. It also provides a built-in test suite for rapid internal testing of web applications.
5. It can support multiple Object Relational Mappers (ORM) for Data Bases access.
6. It can also work with different templating languages such as Mako, Jinja, Genshi,CherryTemplate, etc.
7. It is also known as an Object-Oriented web application framework and supports all of itsproperties like Data hiding and security.
8. It is an open-source project .
9. It comes with a built-in production-ready HTTP server for development and deployment.
10. As it follows Object Orientation concepts, we can use Inheritance in CherryPy for DataReusability.

Advantages of CherryPy :-

At present there are a ton of Python frameworks you can use, especially for creating web applications. So what's special about CherryPy? Here are a few points that make it distinguishable from its counterparts.

1. **Quick learning curve**. The framework uses Python's built-in conventions, allowing you to get started with basic Python knowledge. Just like regular Python applications, it provides extensive tools and plugins for faster development. Almost every framework has its own set of tools for computations, but CherryPy can also use existing Python libraries if necessary. If you or your team is well versed with Python, using CherryPy for your web service would give you a great time to market.

2. **Flexibility**. The major advantage of using CherryPy over other Python-based frameworks like Django is that it doesn't force the developers to stick to a particular structure, which makes it more flexible. You have more control and freedom over how to design your services, best practices to follow, etc.

3. **Versatility.** CherryPy has the ability to create web services like RESTful Web Service (RWS), WSDL, SOAP, etc. You can use it to build things like e-commerce websites or authentication services by integrating various other Python modules.

4. **Affordable.** CherryPy can easily be deployed in a cost-effective manner, as it has its own HTTP server for hosting an application on multiple types of gateways.

Architecture and Working with CherryPy :-

The architecture of CherryPy consists of three integral components:

Cherrypy.engine. This manages your website's behavior, how the HTTP server functions (start/stop), and handles process reloads, file management, etc. However, the two important functions are:

1. Creation of request and response objects
2. Control and management of the entire process

Cherrypy.server. This component configures and controls your actual HTTP server.

Cherrypy.tools. This consists of various tools to help process HTTP requests.CherryPy provides some great tools and plugins out of the box that speed up the development process.

(XI) TurboGears:-

1. TurboGears is a Python web framework that follows the MVC Architecture.
2. It is a full-stack web framework, but with its minimal interface and single page script, it can act as a micro web framework when we do not want the full stack use of TurboGears.
3. It comes with many dependencies such as WSGI for server web-page interaction, SQLAlchemy for database, and Genshi for Markup templating.
4. There are two main series of TurboGears. TurboGears 1.x and TurboGears 2.x, between which 2.x is the latest series.

The main components that TurboGears offers:-

1. **SQLAlchemy:** SQLAlchemy is an Object Relational Mapper (ORM) toolkit that helps users create and interact with SQL databases using Python objects.
2. **Genshi**: Genshi is a Python template engine, and TurboGears uses it to insert data in HTML or XHTML pages.
3. **ToscaWidgets:** To create complex forms and GUIs, TurboGears depend on its built-in component called ToscaWidgets. Tosca can generate complex and straightforward HTML forms and connect JavaScript widgets and toolkits.
4. **Gearbox**: Although the web app can be connected to Apache, Nginx, or any other WSGI server, it provides a built-in Gearbox toolkit to manage its Project.

Features of TurboGears:-

1. It can act as a micro as well as a full-stack web framework.
2. Comes with a powerful Object Relational Mapper (ORM) and supports multi-database.
3. It follows MVC Architecture.
4. Comes with a built-in templating engine like Genshi.
5. Provides FomeEncode for validation.
6. It also includes a toolkit like Repoze for web security like authorization.
7. It also provides a command-line tool for project management.
8. Comes with built-in ToscaWidgets to display backend data on the front-end design.
9. Support Front-Faching WSGI-based server.
10. Use functions and decorators to write the view logic for the web application.

(XII) Flask:-

1. Flask is a web development framework developed in Python. It is easy to learn and use.
2. Flask originated in 2010 when a developer named **Armin Ronacher** created it as an **April Fool's joke.**
3. Flask is known as a micro-framework because it is lightweight and only provides components that are essential.
4. It only provides the necessary components for web development, such as routing, request handling, sessions, and so on. For the other functionalities such as data handling, the developer can write a custom module or use an extension.
5. Flask is one of the popular frameworks of Python.
6. Flask is built on the Werkzeug WSGI toolkit and Jinja2 template engine.
7. Flask Is an Open Source Web Frameworks That Is Specially design To Create Complex Website Mechanism in Fastest and Easiest Way. This Frameworks is Completely Written In Python.

Features of Flask :-

1. Extension Features
2. Large Variety of Automatic handling With Extension Libraries
3. Automatic Template Processing
4. Automatic URL Handling
5. Cross Platform Supports
6. Very Light Weight
7. Supports Wide Range of Servers Applications like Apache, WSGI etc.
8. Completely Written In Pure Python Scripting Language
9. Actively Improving and Developing Features
10. Large Community Support
11. Open Source Software
12. Comes With Built-in Stand Alone Webserver for Development and Testing.

What is WSGI:-

The Web Server Gateway Interface (WSGI) has been used as a standard for Python web application development. WSGI is the specification of a common interface between web servers and web applications.

What is Werkzeug:-

Werkzeug is a WSGI toolkit that implements requests, response objects, and utility functions. This enables a web frame to be built on it. The Flask framework uses Werkzeug as one of its bases.

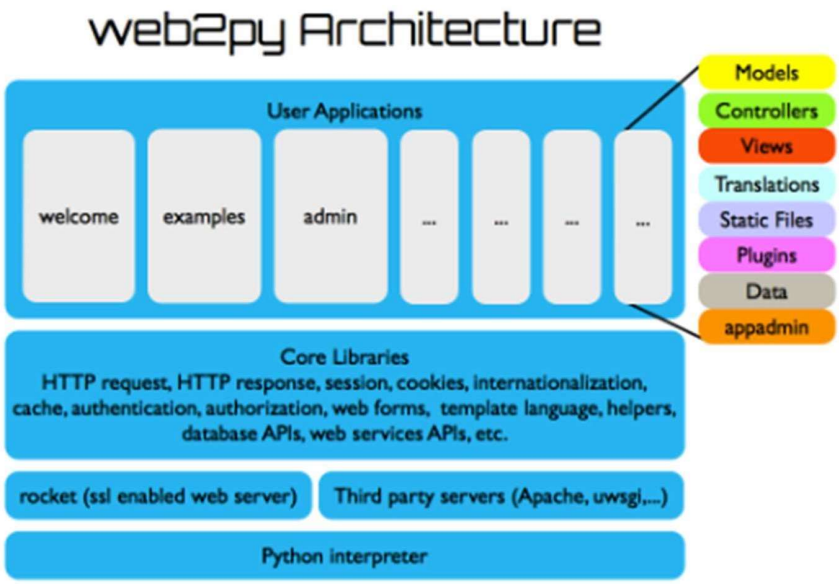
What is jinja2:-

Jinja2 is a popular template engine for Python.A web template system combines a template with a specific data source to render a dynamic web page.

(XIII) Web2Py:-

1. Web2py is a good python web framework. This particular framework comes with a debugger, a deployment tool, and a code editor, which helps create and debug code, test, and maintain applications.
2. It offers an exclusive ticketing system that will issue a ticket once there is an error.
3. Web2py comes with built-in components for handling HTTP responses, requests, sessions, and cookies.
4. Web2py is a free, open-source web framework, it is written in Python.
5. Web2py is a full-stack framework, meaning that it contains all the components you need to build fully functional web applications.
6. Web2py is designed to guide a web developer to follow good software engineering practices, such as using the Model View Controller (MVC) pattern. Web2py separates the data representation (the model) from the data presentation (the view) and also from the application logic and workflow (the controller).
7. Web2py provides libraries to help the developer design, implement, and test each of these three parts separately, and makes them work together.
8. Web2py is built for security. This means that it automatically addresses many of the issues that can lead to security vulnerabilities.

Architecture of Web2py:-



What is the Use of web2py:-

More than web-development purposes web2py is primarily used as a teaching tool. Here are some of the reasons why a Python developer should learn web2py.

1. Because of its graphical interface and built-in web IDE, it makes it easy for the developer to learn server-side web development.
2. It is a stable Python web framework, and all of its APIs are solid as rocks.
3. A web application built on web2py is very secure.
4. It comes batteries included, so developers do not have to worry about building common web components.
5. It uses the Rocket WSGI to run its web app faster.

Pyramid:-

1. Pyramid is a python framework which is general purpose, open source and allows developer to create web applications.
2. Pyramid is similar as Flask framework which takes very little effort to install and run.
3. Python Pyramid has a wide range of scalability that can be applied for different projects.
4. It helps in the smart documentation and development of a complex application.
5. Pyramid is based on Zope, Pylons, and Django. It supports small and large projects so there is no need to rewrite like you would with small frameworks. It's also the fastest Python web framework.

Why choose Pyramid framework:-

These are some of the things that makes Pyramid a suitable choice for beginners:

1. As a beginner, single-file module with little to no complexity.
2. Use the following Convenient scaffoldTo generate a sample project using your subsystems combination
3. You can choose from a range of Security solutions, templating, and database Pyramids add-on systems offer more convenience and quality.
4. You can tap into a wide range of High-quality documentation Pyramid is available for evaluation, testing, and advanced development.

This is why web application programming in Pyramid framework using Python is easy and best.

CORE FEATURES:-

Pyramid comes with a set of features that is unique amongst Python web application frameworks:

1. MVC architecture
2. Platform independence
3. Multiple methods of configuration: ZCML, imperative, decorator-based
4. Extensible templating
5. View predicates and many views per route
6. Built-in HTTP sessioning
7. Configuration extensibility
8. Flexible authentication and authorization
9. Support of relational and non-relational databases.