**CFSS INTERNSHIP**



Whois Lookup

(Python Project)

Name - Jatin Yadav

Phone No. – 7988350346

Mail Id – [Jatin21csu249@ncuindia.edu](mailto:Jatin21csu249@ncuindia.edu)

LinkedIn - [linkedin.com/in/jatin-yadav-37563121b](https://www.linkedin.com/in/jatin-yadav-37563121b)

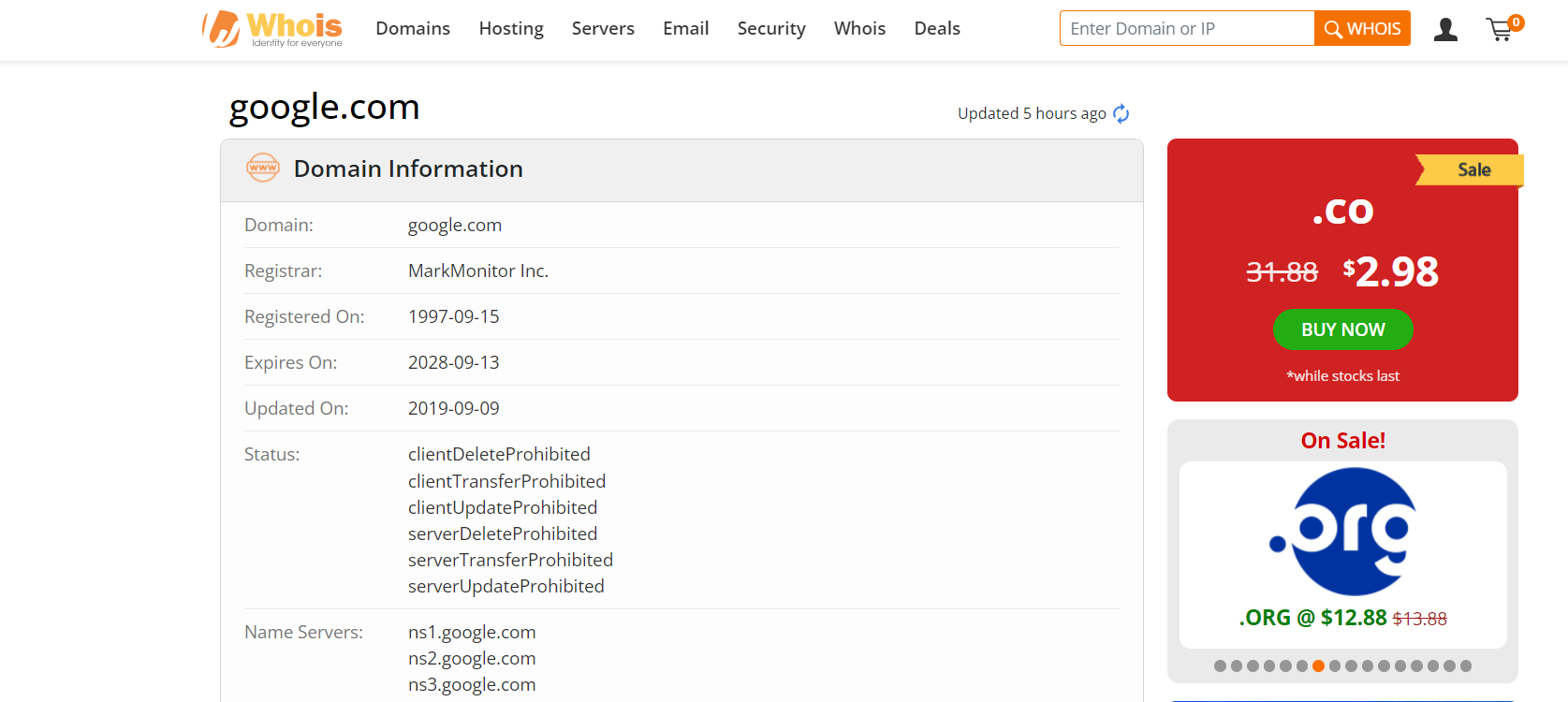
Whois Lookup Tool

(Python Project)

**1.What is Whois Lookup tool ?**

**-**Whois lookup is a vital resource for domain managers, cybersecurity professionals, and anybody curious about the history of a website or domain. WHOIS helps to democratize the Internet. Anyone, from businesses and corporations to law enforcement and individual users, can access and use the WHOIS database to find out who is behind a domain name and any associated website. Beyond that, many people find the WHOIS database a great tool for business opportunities.

**2.Sample :-**

****

**Domain Ownership:** The registrant’s name and contact information

Ex - <Google.com>

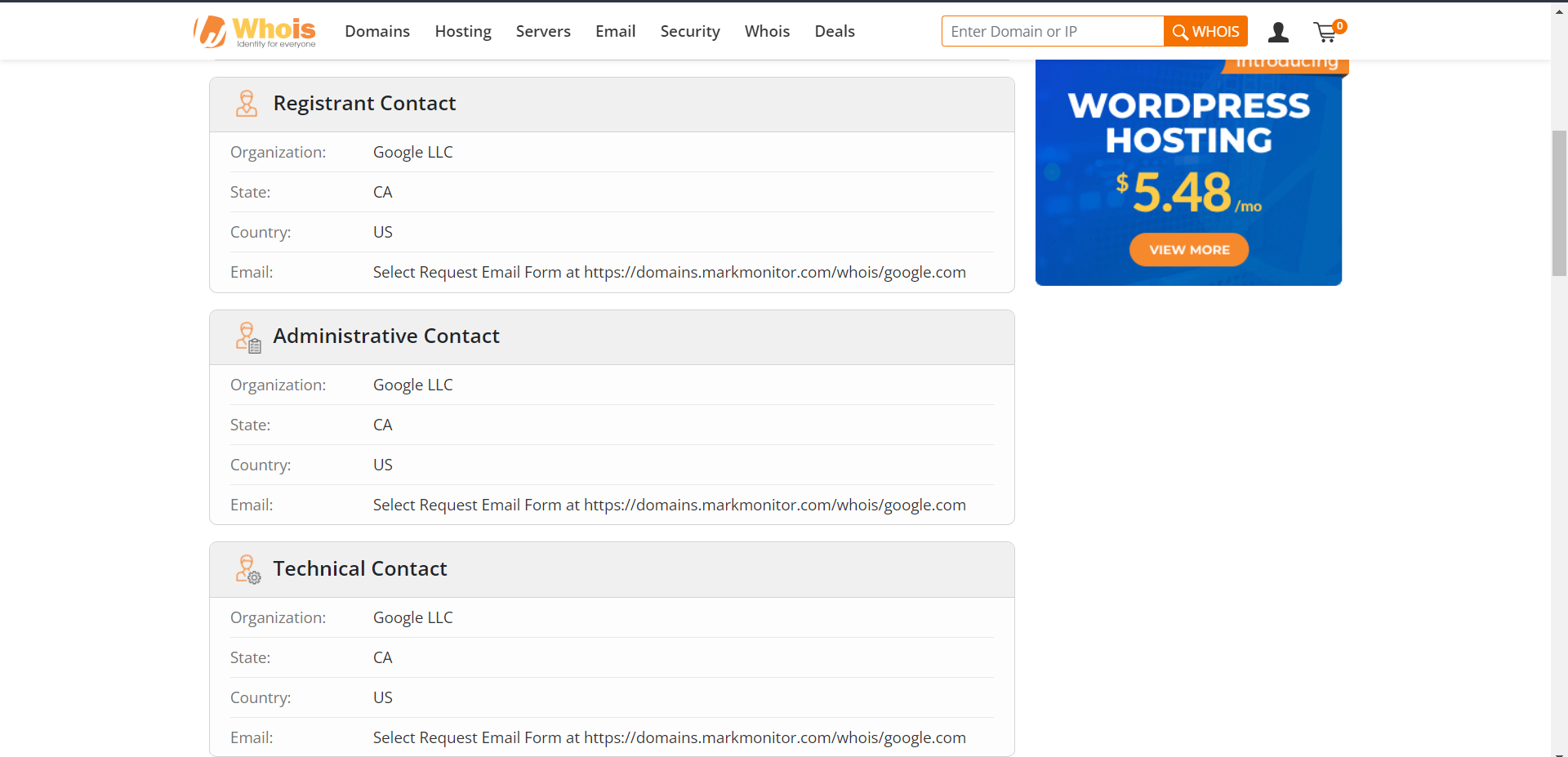
**Dates of Registration:** The domain’s inception and expiration dates.

Ex - <1997-09-15>

**Name Servers:** The servers in charge of responding to domain-specific DNS requests

Ex - < ns1.google.com >

< ns2.google.com >  
 < ns3.google.com >  
 < ns4.google.com >



**Administrative and Technical Contacts:** people or groups in charge of overseeing the domain.

Ex - < Organization: Google LLC

State: CA

Country: US

Email: Select Request Email Form at <https://domains.markmonitor.com/whois/google.com> >

**3.Raw Data**

Domain Name: google.com

Registry Domain ID: 2138514\_DOMAIN\_COM-VRSN

Registrar WHOIS Server: whois.markmonitor.com

Registrar URL: http://www.markmonitor.com

Updated Date: 2019-09-09T15:39:04+0000

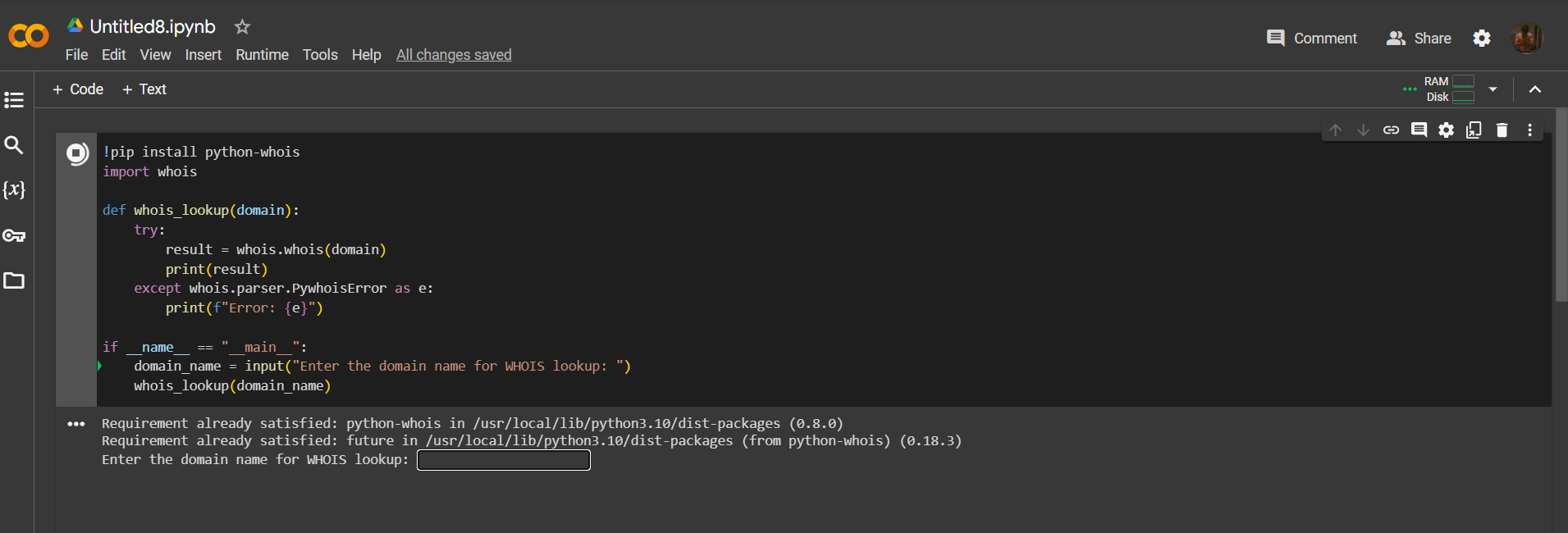
Creation Date: 1997-09-15T07:00:00+0000

Registrar Registration Expiration Date: 2028-09-13T07:00:00+0000

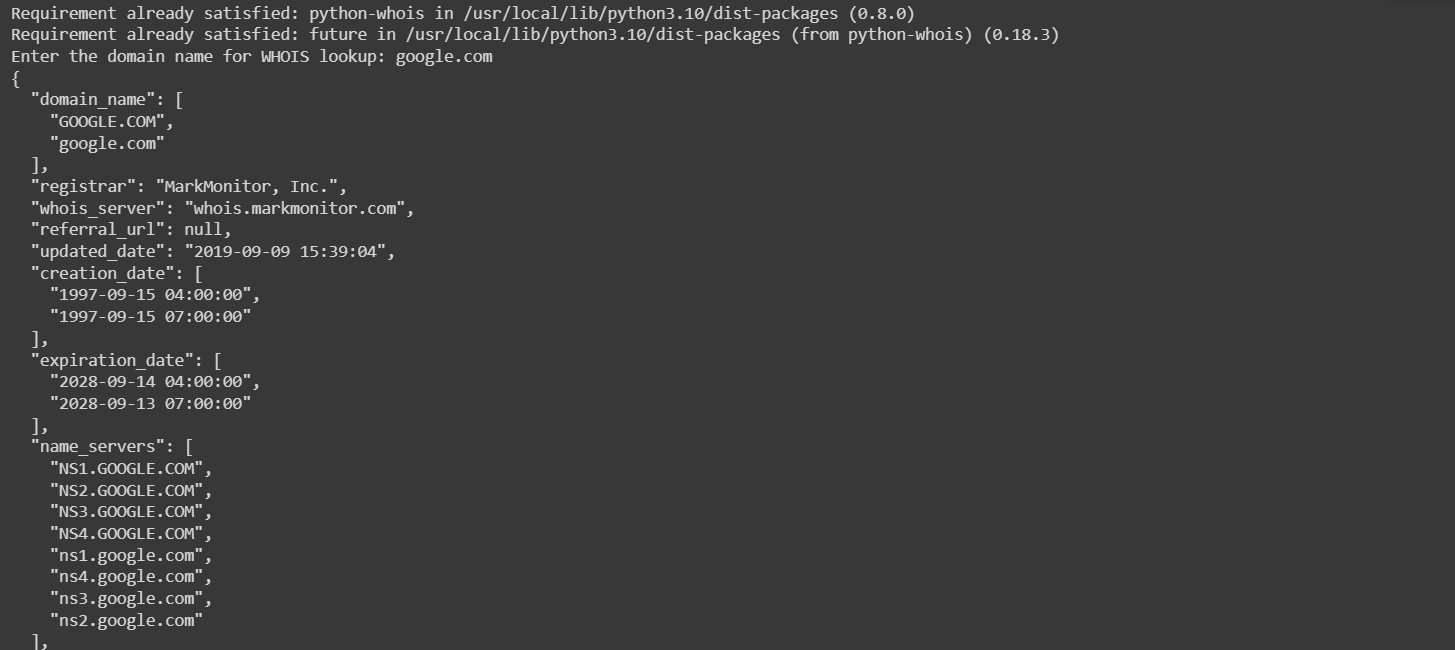
Registrar: MarkMonitor, Inc.

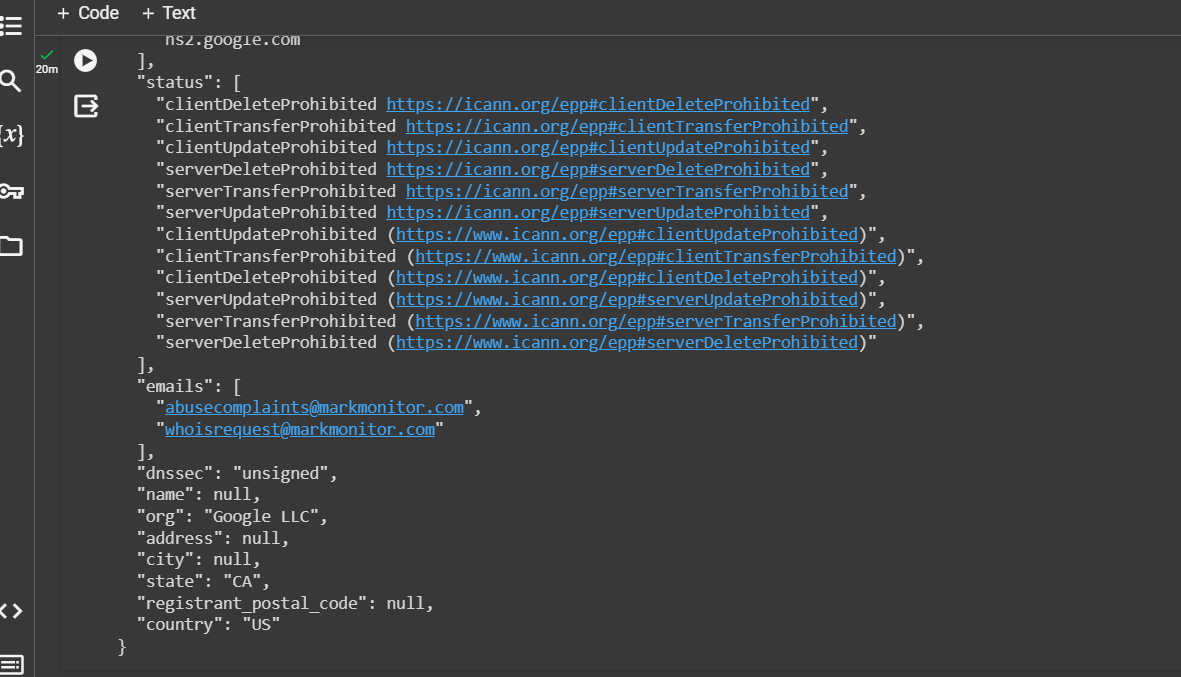
**#Coding Part**

1. **Using Python (User Input):-**



**Output:-**





By using the whois library in Python, we can easily retrieve this information programmatically.

**Key Concepts**

1. Before diving into the code, let's understand a few key concepts related to WHOIS lookup:
2. Domain Name: A domain name is the unique address that identifies a website on the internet, such as "example.com".
3. WHOIS: WHOIS is a protocol that provides information about a domain name, including the registrant's contact details, registration date, and expiration date.
4. WHOIS Lookup: WHOIS lookup is the process of querying a WHOIS server to retrieve information about a domain name.

**Code Structure:-**

The code provided consists of a single function whois\_lookup that takes a domain name as input and performs a WHOIS lookup using the whois library. The function catches any PywhoisError exceptions that may occur during the lookup.

**Code :-**

Let's take a closer look at the code and understand how it works:

language-python

***import whois***

***def whois\_lookup(domain):***

***try:***

***result = whois.whois(domain)***

***print(result)***

***except whois.parser.PywhoisError as e:***

***print(f"Error: {e}")***

***if \_\_name\_\_ == "\_\_main\_\_":***

***domain\_name = input("Enter the domain name for WHOIS lookup: ")***

***whois\_lookup(domain\_name)***

**Explanation:-**

1. First, we import the whois module, which provides the functionality to perform WHOIS lookups.
2. Next, we define the whois\_lookup function that takes a domain parameter. Inside the function, we use the whois.whois method to perform the lookup and store the result in the result variable.
3. If the lookup is successful, we print the result, which contains information about the domain.
4. If an exception of type PywhoisError occurs during the lookup, we catch it and print an error message.
5. Finally, in the if \_\_name\_\_ == "\_\_main\_\_": block, we prompt the user to enter a domain name for the WHOIS lookup. The entered domain name is then passed as an argument to the whois\_lookup function.

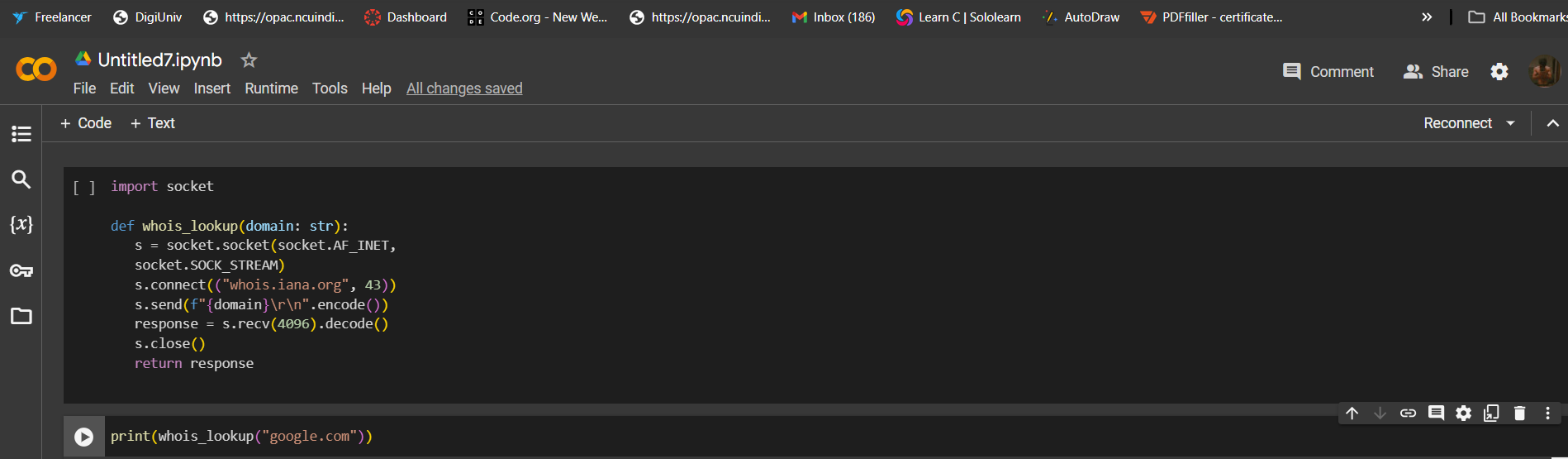
**Conclusion:-**

Performing a WHOIS lookup in Python is straightforward with the help of the whois library. By using the provided code, you can easily retrieve information about a domain name

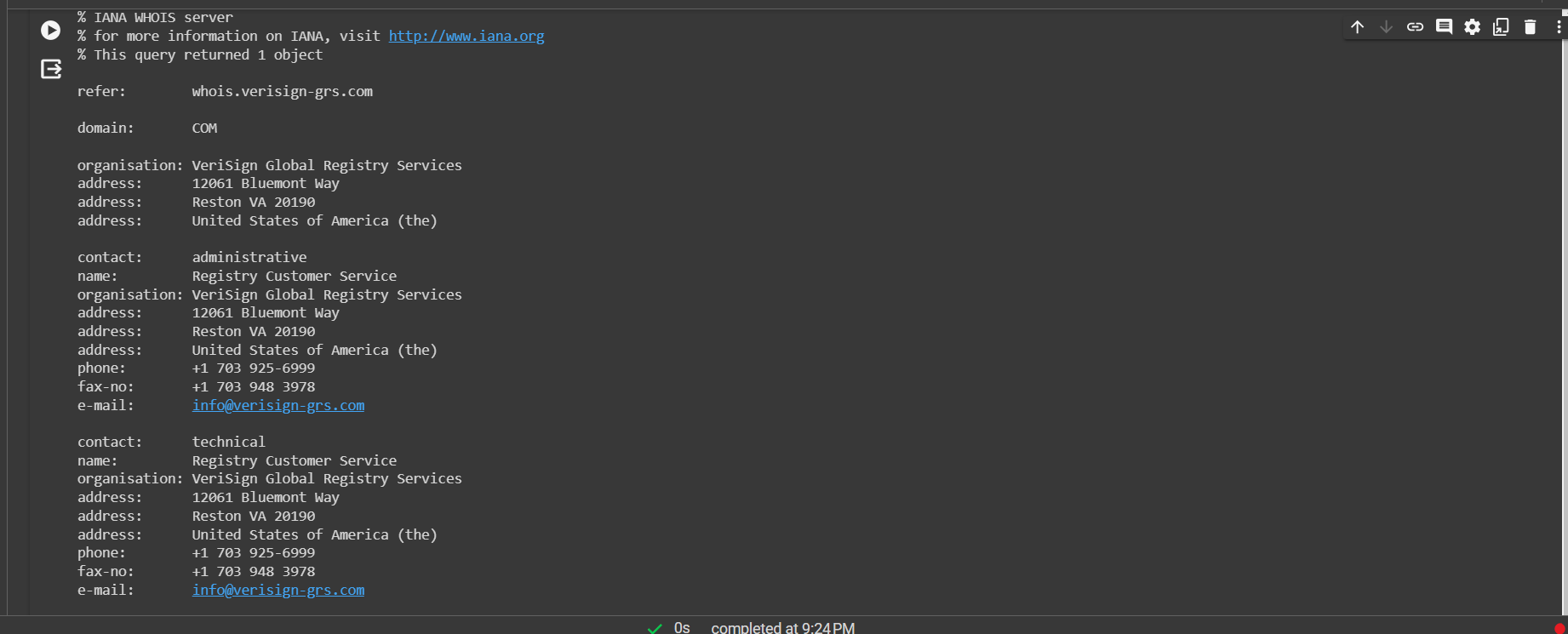
programmatically. Remember to handle any exceptions that may occur during the lookup to ensure a smooth execution of your program.

**#Coding Part**

1. **Using Python (Without User Input) :-**



**Output:-**

****

By using the Whois protocol, we can obtain valuable information about a domain, such as the registrar, registration date, expiration date, and contact details.

**Key Concepts**

Before diving into the code, let's understand some key concepts related to Whois:

1. Domain: A domain is a unique name that identifies a website on the internet. It consists of two parts: the top-level domain (TLD) and the second-level domain (SLD). For example, in the domain "google.com," "google" is the SLD, and ".com" is the TLD.
2. Whois: Whois is a protocol that allows users to query a database and retrieve information about a domain or IP address. It provides details about the domain's registration, ownership, and contact information.
3. Registrar: A registrar is an organization authorized to register domain names. They maintain a database of registered domains and provide services for domain registration, renewal, and management.
4. Socket: In Python, the socket module provides low-level networking interfaces. It allows us to create client and server sockets to establish network connections and communicate over various protocols.

**Code Structure:-**

The code provided demonstrates a simple implementation of a Whois lookup in Python. It consists of a single function, whois\_lookup, which takes a domain name as input and returns the Whois information for that domain.

**Code:-**

Let's take a closer look at the code and understand how it works:

language-python

***import socket***

***def whois\_lookup(domain: str):***

***s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)***

***s.connect(("whois.iana.org", 43))***

***s.send(f"{domain}\r\n".encode())***

***response = s.recv(4096).decode()***

***s.close()***

***return response***

***print(whois\_lookup("google.com"))***

**Explanation:-**

1. We start by importing the socket module, which provides the necessary functions for network communication.
2. The whois\_lookup function takes a domain name as input. It creates a socket using the socket.socket function, specifying the address family (socket.AF\_INET) and the socket type (socket.SOCK\_STREAM).
3. We establish a connection to the Whois server by calling the connect method on the socket object. In this case, we connect to the Whois server at "whois.iana.org" on port 43, which is the standard port for Whois queries.
4. Next, we send the domain name to the server using the send method. We encode the domain name as bytes using the encode method and append "\r\n" to indicate the end of the message.
5. We receive the response from the server using the recv method, specifying the maximum number of bytes to receive (4096 in this case). We decode the received bytes into a string using the decode method.
6. Finally, we close the socket and return the response.
7. In the example, we call the whois\_lookup function with the domain name "google.com" and print the response.

When you run the code, it will establish a connection to the Whois server, send the domain name, and retrieve the Whois information for the specified domain. The response will contain detailed information about the domain, including its registrar, registration date, and expiration date.

**Conclusion:-**

Performing a Whois lookup in Python is a straightforward process using the socket module. By establishing a connection to a Whois server and sending the domain name, we can retrieve valuable information about a domain. This information can be useful for various purposes, such as verifying domain ownership, checking domain expiration dates, and investigating potential domain abuse.

**Theoretical questions**

**Q1. What are virtual environments in Python, and why are they essential in real-life development projects? How do you create and manage virtual environments?**

Ans **Virtual Environments in Python** – It is a self-contained directory that contains a Python interpreter and its own set of libraries and scripts. This allows you to have multiple Python environments with different packages and versions on the same system. Virtual environments are essential in real-life development projects for several reasons :- 1. Version Compatibility

2. Clean Development and Deployment

3. Isolation

4. Dependency Management

**-Creating and Managing Virtual Environments:**

Python comes with a built-in module called venv for creating virtual environments. Here are the basic steps:

1. Create a Virtual Environment: python -m venv myenv (bash code)

This command creates a virtual environment named myenv in the current directory.

1. Activate the Virtual Environment:

* On Windows: myenv\Scripts\activate
* On Unix or MacOS: source myenv/bin/activate

After activation, your terminal prompt should change, indicating that the virtual environment is active.

1. Install Packages:

Now, you can use pip to install packages within the virtual environment:

* pip install package\_name

1. Deactivate the Virtual Environment:

* deactivate

This command deactivates the virtual environment and returns you to the global Python environment.

1. Managing Virtual Environments with virtualenv and pipenv:

Alternatively, you can use third-party tools like virtualenv and pipenv for virtual environment management. Install them using:

* pip install virtualenv

or

* pip install pipenv

Then, you can create and manage virtual environments using commands provided by these tools. For example:

* With virtualenv:
* virtualenv myenv

source myenv/bin/activate

* With pipenv:
* pipenv install

pipenv shell

Choose the tool that best fits your workflow and project requirements. Each has its strengths and features for managing virtual environments and dependencies.

**Q2**. **Discuss the differences between Python 2 and Python 3. Why is it crucial for businesses and developers to migrate to Python 3 in real-world scenarios?**

**Ans** Differences between Python 2 and Python 3:

1. Print Statement:

* Python 2: print is a statement (e.g., print "Hello").
* Python 3: print is a function (e.g., print("Hello")).

1. Integer Division:

* Python 2: / performs integer division if both operands are integers.
* Python 3: // performs integer division, and / always performs true division.

1. Unicode Support:

* Python 2: Strings are ASCII by default, and Unicode support is available using u'unicode\_string'.
* Python 3: Strings are Unicode by default ('unicode\_string').

1. xrange():

* Python 2: xrange() provides an iterator for ranges, saving memory compared to range().
* Python 3: range() behaves like xrange() in Python 2, and there's no xrange() in Python 3.

1. Input Function:

* Python 2: input() evaluates user input as code, and raw\_input() reads input as a string.
* Python 3: input() reads input as a string, and raw\_input() is not available.

1. Exceptions:

* Python 2: except syntax without parentheses (e.g., except IOError, e:).
* Python 3: except syntax with parentheses (e.g., except IOError as e:).

**Reasons to Migrate to Python 3:**

1. End of Python 2 Support:

* Python 2 reached its end of life on January 1, 2020. This means no more official support, updates, or security patches.

1. Improved Syntax and Features:

* Python 3 brings several language enhancements, performance improvements, and new features, making it a more modern and capable language.

1. Library and Ecosystem Support:

* Many third-party libraries have dropped support for Python 2, and new libraries and tools are developed exclusively for Python 3.

1. Security:

* Python 3 is actively maintained, receiving security updates and bug fixes. Sticking with Python 2 exposes systems to potential vulnerabilities.

1. Performance:

* Python 3 has various optimizations and improvements, resulting in better performance compared to Python 2.

1. Unicode Handling:

* Python 3's native Unicode support simplifies string handling and reduces the likelihood of encoding-related bugs.

1. Community and Documentation:

* The Python community is focused on Python 3, and most documentation, tutorials, and resources are geared towards Python 3.

1. Future-Proofing:

* Choosing Python 3 ensures compatibility with ongoing developments in the Python language and ecosystem.

In real-world scenarios, migrating to Python 3 is crucial for long-term sustainability, security, and compatibility with the latest technologies. Businesses and developers should invest in the migration process to benefit from the improvements and community support provided by Python 3. It allows them to stay current, leverage new features, and mitigate the risks associated with using an outdated and unsupported version of the language.

**Q3. Explain the concept of API (Application Programming Interface). How is Python used to interact with external APIs in various applications? Provide real-life examples.**

Ans An API, or Application Programming Interface, is a set of rules and tools that allows one software application to interact with another. It defines the methods and data formats that applications can use to request and exchange information. APIs can be used for various purposes, such as accessing web services, databases, or hardware devices.

In the context of web development, an API is often a set of rules that enable communication between different software applications over the internet. It allows developers to integrate third-party services, retrieve data, or perform specific operations in a standardized way.

Using Python to Interact with External APIs:

Python provides various libraries and modules that make it easy to interact with external APIs. The requests library is commonly used for making HTTP requests, and it simplifies the process of sending and receiving data.

Here's a simple example using the OpenWeatherMap API to get the current weather information for a city:

* Python code

import requests

def get\_weather(api\_key, city):

base\_url = "https://api.openweathermap.org/data/2.5/weather"

params = {

"q": city,

"appid": api\_key,

}

response = requests.get(base\_url, params=params)

if response.status\_code == 200:

weather\_data = response.json()

print(f"Weather in {city}: {weather\_data['weather'][0]['description']}")

else:

print(f"Failed to get weather data. Status code: {response.status\_code}")

# Replace 'your\_api\_key' with your OpenWeatherMap API key

api\_key = 'your\_api\_key'

city\_name = 'New York'

get\_weather(api\_key, city\_name)

In this example, the get\_weather function sends a request to the OpenWeatherMap API, passing the API key and the city name as parameters. The response is then processed to extract and display the weather description.

Real-Life Examples:

* Google Maps API:

Python can be used to interact with the Google Maps API using the googlemaps library. This allows developers to integrate mapping and location-based services into their applications.

* GitHub API:

Python scripts can utilize the GitHub API, accessible through libraries like requests or specialized GitHub libraries. This enables developers to automate tasks like creating repositories, managing issues, or retrieving repository information.

* Weather APIs:

As shown in the example above, weather APIs like OpenWeatherMap can be accessed using Python to retrieve real-time weather data for specific locations.

Python's versatility, along with its rich ecosystem of libraries, makes it a popular choice for developers when working with APIs. Whether it's for data retrieval, integration with third-party services, or automation of tasks, Python's simplicity and readability contribute to its effectiveness in API interaction.

**Q4. What is a hash function, and why is it essential in cybersecurity? Provide**

**an example of a Python library used for hashing.**

**Ans** A hash function is a mathematical algorithm that transforms input data of any size into a fixed-size string of characters, which is typically a hash value or hash code. The output, often called the hash digest, should be unique for different inputs. Hash functions are commonly used in various applications, including data integrity verification, password storage, and digital signatures.

* Key Characteristics of Hash Functions:

1. Deterministic: The same input will always produce the same hash output.
2. Fast Computation: The hash function should be computationally efficient.
3. Fixed Output Length: The hash function produces a fixed-length output, regardless of the input size.
4. Irreversibility: It should be infeasible to reverse the process and obtain the original input from the hash.

* Importance in Cybersecurity:

Hash functions play a crucial role in cybersecurity for several reasons:

1. Password Storage:

Storing passwords in plain text is insecure. Hash functions help secure passwords by storing only their hash values. During authentication, the system hashes the entered password and compares it with the stored hash.

1. Integrity Verification:

Hash functions are used to generate checksums or digital signatures for files. Any change in the file content results in a different hash, allowing users to verify the integrity of downloaded or transmitted files.

1. Digital Signatures:

Hash functions are a fundamental component of digital signatures. They are used to generate a hash of the message, and the hash is then encrypted with the sender's private key, providing a way to verify the sender's identity and the integrity of the message.

1. Data Deduplication:

Hash functions are used to identify and eliminate duplicate data efficiently. By comparing hash values, systems can quickly identify identical files or chunks of data.

1. Cryptographic Applications:

Hash functions are used in various cryptographic protocols, such as HMAC (Hash-based Message Authentication Code), to ensure the integrity and authenticity of messages.

1. Python Library for Hashing:

Python provides the hashlib library, which includes a variety of hash functions, including popular ones like SHA-256 and MD5. Here's a simple example of hashing a string using hashlib:

**python code**

import hashlib

def hash\_string(input\_string, hash\_algorithm='sha256'):

hash\_object.update(input\_string.encode('utf-8'))

return hash\_object.hexdigest()

# Example usage

original\_data = "Hello, World!"

hashed\_data = hash\_string(original\_data)

print(f"Original Data: {original\_data}")

print(f"Hashed Data: {hashed\_data}")

In this example, the hashlib.new() function is used to create a hash object with the specified algorithm (sha256 in this case), and update() is used to update the hash with the input string. The resulting hash digest is obtained using hexdigest(). The choice of hash algorithm depends on the specific security requirements of the application.

**Q5.Explain the role of firewalls in network security. Can Python be used to configure and manage firewalls?**

**Ans Role of Firewalls in Network Security:**

A firewall is a network security device or software that monitors and controls incoming and outgoing network traffic based on predetermined security rules. The primary goal of a firewall is to establish a barrier between a secure internal network and untrusted external networks, such as the internet. Firewalls play a crucial role in network security by:

1. Packet Filtering: Examining network packets and allowing or blocking them based on predefined rules. This can include filtering based on source and destination IP addresses, port numbers, and protocols.
2. Stateful Inspection: Keeping track of the state of active connections and making decisions based on the context of the traffic. Stateful firewalls understand the state of a connection and can make more intelligent decisions about whether to allow or deny traffic.
3. Proxy Services: Acting as an intermediary between client and server, forwarding requests on behalf of clients and returning responses. This can enhance security by hiding internal network details.
4. Network Address Translation (NAT): Modifying network address information in packet headers while in transit, allowing private internal addresses to be used on public networks.
5. Logging and Auditing: Recording information about network traffic and firewall activities, enabling monitoring and analysis for security purposes.

Using Python to Configure and Manage Firewalls:

Python can be utilized to interact with firewall devices or software using appropriate APIs or libraries. The ability to configure and manage firewalls via Python scripts depends on the specific firewall solution and its level of programmability.

Here are a few ways Python can be used with firewalls:

* RESTful APIs:

Many modern firewalls provide RESTful APIs that allow you to manage and configure them programmatically. Python's requests library can be used to send HTTP requests to these APIs.

* Vendor-Specific Libraries:

Some firewall vendors provide Python libraries or SDKs that allow developers to interact with and configure their firewall devices. For example, Cisco provides the ciscoconfparse library for working with Cisco IOS configurations.

* Automation Frameworks:

Automation frameworks like Ansible or NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support) can be used to automate the configuration of network devices, including firewalls.

* SSH/Telnet Scripts:

For firewalls that support command-line interfaces (CLI) accessible via SSH or Telnet, Python scripts using libraries like paramiko can automate interactions by sending commands and parsing responses.

Here's a simplified example using requests to interact with a hypothetical RESTful API for a firewall:

**Python code**

import requests

def configure\_firewall(api\_endpoint, api\_key, rule):

headers = {"Authorization": f"Bearer {api\_key}"}

payload = {"rule": rule}

response = requests.post(api\_endpoint, headers=headers, json=payload)

if response.status\_code == 200:

print("Firewall rule configured successfully.")

else:

print(f"Failed to configure firewall rule. Status code: {response.status\_code}")

# Example usage

api\_endpoint = "https://firewall-api.example.com/configure-rule"

api\_key = "your\_api\_key"

new\_rule = {"source": "192.168.1.0/24", "action": "allow", "destination": "any"}

configure\_firewall(api\_endpoint, api\_key, new\_rule)

Note that the specifics of interacting with firewalls programmatically will depend on the firewall vendor and the capabilities they expose through APIs or other interfaces. Always refer to the documentation provided by the firewall vendor for accurate and secure integration.

**Q6. What is a brute-force attack? How can Python scripts be used to defend against brute-force attacks on passwords?**

**Ans**

**Brute-Force Attack:**

A brute-force attack is a cybersecurity attack method in which an attacker systematically attempts all possible combinations of passwords or encryption keys until the correct one is found. The term "brute force" implies using sheer computational power to try every possible combination, as it does not rely on exploiting vulnerabilities in the system.

For password-related brute-force attacks, an attacker typically uses automated scripts or tools to repeatedly try different passwords until the correct one is discovered. Brute-force attacks are time-consuming, but they can be effective if the password is weak or easily guessable.

**Defending Against Brute-Force Attacks with Python:**

Python can be used to implement various strategies to defend against brute-force attacks on passwords. Here are some common techniques:

1. Password Complexity Policies:

Enforce strong password policies that require users to create complex passwords containing a mix of uppercase and lowercase letters, numbers, and special characters. Python scripts can be used to check and enforce these policies during the password creation process.

1. Account Lockout Policies:

Implement account lockout policies that temporarily or permanently lock an account after a certain number of failed login attempts. Python scripts can monitor login attempts and enforce lockout policies.

1. Rate Limiting:

Implement rate limiting to restrict the number of login attempts within a specific time period. Python scripts can monitor login attempts and delay or block further attempts if they exceed a predefined threshold.

1. CAPTCHA or Two-Factor Authentication (2FA):

Integrate CAPTCHA challenges or two-factor authentication to add an extra layer of security. Python scripts can manage the generation and verification of CAPTCHAs or facilitate 2FA processes.

1. IP Whitelisting/Blacklisting:

Maintain lists of trusted and untrusted IP addresses. Python scripts can monitor login attempts and dynamically update these lists to allow or block access based on IP addresses.

1. Monitoring and Logging:

Implement comprehensive monitoring and logging of login activities. Python scripts can analyze logs in real-time to detect patterns indicative of a brute-force attack, triggering appropriate responses.

1. Honey Pot Accounts:

Create honey pot accounts with intentionally weak passwords to attract attackers. Python scripts can monitor login attempts on these accounts and take action when an attack is detected.

Here's a simplified example of a Python script that implements account lockout policies:

**python code**

import time

MAX\_LOGIN\_ATTEMPTS = 3

LOCKOUT\_DURATION = 60 # seconds

def login(username, password):

# Check username and password

if username == "valid\_user" and password == "strong\_password":

return True

else:

return False

def main():

login\_attempts = 0

while True:

username = input("Enter username: ")

password = input("Enter password: ")

if login(username, password):

print("Login successful!")

break

else:

print("Login failed. Please try again.")

login\_attempts += 1

if login\_attempts >= MAX\_LOGIN\_ATTEMPTS:

print(f"Account locked. Try again after {LOCKOUT\_DURATION} seconds.")

time.sleep(LOCKOUT\_DURATION)

login\_attempts = 0

if \_\_name\_\_ == "\_\_main\_\_":

main()

ss