Using requests.post() in Python

Detailed Explanation of Parameters and Usage

Introduction

- When using requests.post() in Python, an HTTP POST request is sent to a specified URL. This type of request is used to send data to the server, either to store new data or to update existing data. The requests library provides several parameters you can use to customize the request precisely. Below is a detailed explanation of each parameter in a POST request, along with examples.

Basic POST Request

- Example: This basic example does not include any data or additional parameters. In real applications, you will typically need to send data with the request.

```
import requests

response = requests.post("https://example.com/api")
```

1. url - Request URL

- Description: The URL or endpoint you want to send the request to.

- Example:

```
response = requests.post("https://api.example.com/data")
```

2. data - Sending Data

Description: Used to send text data (like form data). The data is passed as a dictionary or other url-encoded objects.

Example:

```
data = {"username": "user123", "password": "pass123"}

response =
requests.post("https://api.example.com/login",
data=data)
```

Note: Data is encoded as application/x-www-form-urlencoded by default.

3. json - Sending JSON Data

Description: To send JSON data, you can use the json parameter directly. The dictionary is automatically converted to JSON, and application/json is added as the content type.

Example:

```
json_data = {"username": "user123", "password": "pass123"}

response = requests.post("https://api.example.com/login", json=json_data)
```

4. headers - Request Headers

Description: Allows you to send additional information with the request, such as content type or authorization.

Format: Pass a dictionary containing header names and values.

Example:

```
headers = {
    "User-Agent": "my-app",
    "Authorization": "Bearer YOUR_ACCESS_TOKEN"
}
response = requests.post("https://api.example.com/data", headers=headers)
```

5. auth - Authentication

Description: If the server requires authentication, you can use auth to send a username and password.

Format: Pass a tuple containing (username, password).

Example:

```
from requests.auth import HTTPBasicAuth


response =
requests.post("https://api.example.com/secure-data",
auth=HTTPBasicAuth("username", "password"))
```

6. cookies - Sending Cookies

- **Description**: Allows you to send cookies with the request as a dictionary.
- **Example**:

```
cookies = {"session_id": "123456"}

response = requests.post("https://api.example.com/data", cookies=cookies)
```

7. files - Uploading Files

Description: Used to upload files via the request. Useful for uploading files to the server.

Format: Pass a dictionary containing the filename and open file object.

Example:

```
files = {"file": open("example.txt", "rb")}

response = requests.post("https://api.example.com/upload", files=files)
```

8. timeout - Timeout

Description: Specifies the maximum time to wait before automatically ending the request if the server does not respond.

Format: A numerical value (in seconds).

Example:

```
response = requests.post("https://api.example.com/data", timeout=5)
```

9. allow_redirects - Redirects

Description: Specifies whether to follow redirects automatically.

Format: True to allow redirects, False to disallow.

Default: True

Example:

```
response = requests.post("https://api.example.com/redirect", allow_redirects=False)
```

10. proxies - Using Proxies

Description: Used to route the request through a proxy server.

Format: A dictionary containing proxy addresses.

Example:

```
proxies = {
    "http": "http://10.10.1.10:3128",
   "https": "https://10.10.1.10:1080"
}
response = requests.post("https://api.example.com/data", proxies=proxies)
```

11. stream - Streaming

Description: When set to True, allows reading the response incrementally, which helps in handling large data.

Example:

```
response = requests.post("https://api.example.com/large-file", stream=True)
```

12. verify - SSL Verification

Description: Specifies whether to verify SSL certificates.

Format: True to verify the certificate, or False to skip verification.

Default: True

Example:

```
response = requests.post("https://self-signed.badssl.com/", verify=False)
```

Complete Example of a POST Request with All Parameters

```
import requests
from requests.auth import HTTPBasicAuth
url = "https://api.example.com/data"
data = {"key1": "value1", "key2": "value2"}
json_data = {"username": "user123",
 "password": "pass123"}
headers = {"User-Agent": "my-app"}
cookies = {"session_id": "123456"}
files = {"file": open("example.txt", "rb")}
timeout = 5
proxies = {
      "http": "http://10.10.1.10:3128",
   "https": "https://10.10.1.10:1080"
}
```

```
response = requests.post(
    url,
    data=data,
    json=json_data,
    headers=headers,

auth=HTTPBasicAuth("username"
, "password"),
    cookies=cookies,
    files=files,
    timeout=timeout,
    allow_redirects=True,
    proxies=proxies,
    stream=False,
    verify=True
)
print(response.status_code)
print(response.json())
```