Flask – HTTP Method

Article by
shubhamgupta

Last Updated : 15 Jun, 2020

Join our Community
Courses
DSA (Online)(Hybrid) Data Structures & Algorithms in Python - Self Paced
Explore
Offline Courses
Live Classes
Delivered by Experts
Geeks Office Suites
Starting at ₹ 2999
Offline Classes
Best Office Suite Software of 2023
Know More
Python Backend Development with Django - Live
This course is designed to learn how to build scalable and robust web applications using Python Django.
3 months
17k+ views
Certification Program
LEARN MORE

In this article, we will learn how to handle HTTP methods, such as GET and POST in Flask using Python. Here, we will understand the concept of HTTP, GET, and HTTP POST, and then we will the example and implement each in Flask. Before starting, let's understand the basic terminology:

- **GET**: to request data from the server.
- **POST**: to send data to be processed to the server.
- **PUT**: PUT request is used to modify the data on the server. It replaces the entire content at a particular location with data that is passed in the body payload. If there are no resources that match the request, it will generate one.
- **PATCH**: PATCH request is similar to PUT request but the difference is, it modifies a part of the data. It will only replace the content that is required.
- **DELETE**: A DELETE request is used to delete the data on the server at a specified location.

Related Article: Flask HTTP methods, Handle GET & POST requests

Flask HTTP Methods

In a Client-Server architecture, there is a set of rules, called a protocol, using which, we can allow the clients to communicate with the server, and vice-versa. Here, the Hyper Text Transfer Protocol is used, through which, communication is possible. For example, Our browser, passes our query to the Google server, and similarly, the Google server, returns relevant suggestions. The commonly used HTTP methods, for this interaction, are – **GET** and **POST**.

GET Method in Flask

The request are type, in the browser address bar, say, 'http://google.com' is called the Uniform Resource Locator. It mentions the address we are looking for, in this case, the Google Landing/signup page. The browser, sends a GET request, to the Google server, which returns the starting webpage, in response. The GET request is simply used, to fetch data from the server. It should not be used, to apply changes, to the server data.

Python Backend Development with Django - Live
This course is designed to learn how to build scalable and robust web applications using Python Django.
3 months
17k+ views
Certification Program
LEARN MORE

Example of HTTP GET in Flask

Let us discuss, the execution of the GET method, using the Flask library. In this example, we will consider, a landing page, that gives us facts, about Math calculations, and, allows us to enter a number, and, return its square. Let us see the example:

Step 1: The '**squaresum.html**' file, has a form tag, that allows the user to enter a number. The form data is sent, to the page mentioned, in the 'action' attribute. Here, the form data, is sent, to the page, as, indicated by '#'. Since we are using the GET method, the data will be appended, to the URL, in the name-value pair format. So, if we enter number, 2, and, click on Submit button, then data will be appended, as 'http://localhost:5000/squaresum?num=2&submum=Submit'

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Square Of Number</title>
</head>
<body>
    <h1>Welcome to the Maths page!</h1><br>
    <p>Simply shows every choice of our daily lives.<br>
    Calculations, Facts about Numbers, Dates, etc.<br>
    make decisions that affects their way of life. </p>
    <form method="GET" action="#">
        <lebel for="num">
        Enter a number:</lebel><br><br>
        <input type="number" id="num"><br><br>
        <input type="submit" value="Submum" id="submum"></input>
    </form>
</body>
</html>
```

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Answer Page</title>
</head>
<body>
    <h1>Keep Learning Maths!</h1>
    <h2>Square of number {{num}} is {{squaresum}}</h2>
</body>
</html>
```

Step 3: Here, we have written a view function, called 'squaresumfunc'. The view function returns an HTTP response. The function is decorated with app.route('/square'). This decorator, matches the incoming request's URL. In the view function, The incoming requests like localhost:5000/ will be mapped to the view squaresumfunc() function.

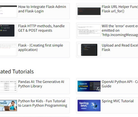
In this case, we have used a GET, hence we mention 'methods=['GET']' in the app.route decorator. The HTML form will, append the number, entered by the user, in the URL. Hence, we check if data is present. To do so, we have to use the request.args bajo. When data is not present, the value of argument "num" will be None. Hence, we will display, the webpage to the user. If the user has not entered, any number, and right away clicked the Submit button, then, we have displayed the error message. We have to use the Flask Jinja2 template, to pass the result, and, the number entered into the HTML file.

Python3

```
# Import the Flask library:
from flask import Flask, render_template, request

# Create the Flask instance and pass the __name
# constructor the path of the current module
app = Flask(__name__)

# The URL 'localhost:5000/square' is mapped to
# the square() function
# The GET request will display the user to enter
# a number (coming from squaresum.html page)

@app.route('/', methods=['GET'])
def square():
    # If method is GET, check if number is entered
    # or user has just requested the page
    # If number is entered, square of number and pass it to
    # answerhtml method
    if request.method == 'GET':
        # Check if the user has requested page the first time
        if(request.args.get('num') == None):
            return render_template('squaresum.html')
        # If number is entered, fetch the number
        # entering number display answer
        elif(request.args.get('num') != ""):
            return "<h1>Welcome to the Maths page!</h1><br>Simply shows
# Start with Flask web app with debug on
# If this is the starting page
if __name__ == '__main__':
    app.run(debug=True)
```

Output:

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 322-000-000
127.0.0.1 - - [12/Dec/2021 19:20:30] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [12/Dec/2021 19:20:30] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [12/Dec/2021 19:20:41] "GET /square?num=2&submum=Submit HTTP/1.1" 200 -
```



The user enters a number, enter the page is entered the URL localhost:5000/square?

Click enter/submit, appended in the URL, and output is displayed.

Example of HTTP POST in Flask

Suppose, we need to register our details, to a website, OR, upload our files, we will send data, from our browser(the client) to the desired server. The HTTP method, preferred here, is POST. The data sent, from HTML, is then saved, on the server side, post validation. The POST method should be used, when we need to changeload data, on the server side.

Example of HTTP POST in Flask

In this example, we will consider, the same landing page, giving us facts, about Math calculations, and, allowing us to enter a number, and, return its square. Let us see the example:

Step 1: The same HTML page, called '**squaresum.html**', is in the templates folder. At the backend side, we will write, appropriate logic, in a view function, to get the number, entered by the user, and, return the same, in the 'answer.html' template. The frontend code file is as shown below:

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Square Of Number</title>
</head>
<body>
    <h1>Welcome to the Maths page!</h1><br>
    <p>Simply shows every choice of our daily lives.<br>
    Calculations, Facts about Numbers, Dates, etc.<br>
    make decisions that affects their way of life. </p>
    <form method="POST" action="#">
        <lebel for="num">
        Enter a number:</lebel><br><br>
        <input type="number" id="num"><br><br>
        <input type="submit" value="Submum" id="submum"></input>
    </form>
</body>
</html>
```

Step 2: The view function squaresumpost(), now also contains value POST in the 'methods' attribute, in the decorator. Thus, when the user requests the page, the first time, by calling 'http://localhost:5000/square', a GET request will be made. Here, the server will render, the corresponding "squaresum.html", webpage. After clicking on Submit, on entering a number, the data is posted back, to the same webpage. Here, the POST method, sends data, in the message body, unlike GET, which appends data in the URL. In the view function, the 'request.form' attribute, retrieves the number, by accessing the 'form' attribute, of the request. The square of the number is calculated, and the value is passed to the same "answer.html" webpage, using the Jinja2 template.

Python3

```
# Import the Flask library:
from flask import Flask, render_template, request

# Create the Flask instance and pass the Flask constructor the path of the
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def square():
    # If method is GET, get the number entered by user
    # Calculate the square of number and pass it to answerhtml
    if request.method == 'POST':
        # Check if the user has requested page the first time
        if(request.form['num'] == None):
            return render_template('squaresum.html')
        # If number is entered, fetch the number
        num = int(request.form['num'])
        sq = int(num)**2 # Calculate square
        return render_template('answer.html',
                               squaresum=sq, num=num)
    # If the method is GET, render the HTML page, for the user
    if request.method == 'GET':
        return render_template('squaresum.html')

# Start with Flask web app with debug on True only
# If this is the starting page
if __name__ == '__main__':
    app.run(debug=True)
```

Output:

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 322-000-000
127.0.0.1 - - [12/Dec/2021 19:22:20] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [12/Dec/2021 19:22:20] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [12/Dec/2021 19:22:41] "POST /square?num=2&submum=Submit HTTP/1.1" 200 -
```

The user enters a number, enter the page is entered the URL localhost:5000/square?

On clicking the Submit button, the data is posted back, to the same URL, and, the result page is rendered. Please note, the value entered, is not visible in the URL now, as the method used, is POST.

Related Articles: Difference between GET and POST

Last Updated : 21 Jun, 2020
Similar Reads
Documenting Flask Endpoint using Flask-Restplus
How to use Flask Session in Python Flask
Minify HTML in Flask using Flask-Minify
Connect Flask to a Database with Flask-SQLAlchemy
How to integrate Flask Admin and Flask-Login
Flask URL Helper Function - url_for()
Flask HTTP methods, handle GET & POST requests
Will the 'error' event ever be emitted on 'http.incomingMessage' in a...
Flask - Creating first simple application
Upload and Read Excel File in Flask
Related Articles
Pandas IN The Generative AI - Python Library
OpenAI Python API - Complete Guide
Python for Kids - Fun Tutorial to Learn Python Programming
Spring MVC Tutorial
Spring Boot Tutorial
Vote for difficulty
Easy
Normal
Medium
Hard
Expert
Improved By : rishikesh, surajkr_gupta
Article Tags : Python3, python, Technical Scripter 2022, Python, Technical Scripter
Practice Tags : python, python
Improve Article
Report Issue
Data Structures & Algorithms in Python - Self Paced
Data Structure and Algorithms are building blocks of programming. Data structures enable us to...
DSA (Online)(Hybrid)
17k+ views
Offered by GeeksforGeeks
LEARN MORE
Previous Article
Flask App Routing
Next Article
Flask - Variable Rule
Article Contributed By :
shubhamgupta
shubhamgupta
Follow
GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org
Company
About Us
Legal
Careers
In Media
Contact Us
Advertise with us
Explore
Job Portal
Jobs
Practice
Contests
Mock Tests
Hiring Challenges
Hack a Thon
GfG Weekly Contest
Offline Tutorials
DSA(Hindi)
DSA in JAVA/C++
Master System Design
Master CP
Languages
Python
Java
C++
PHP
Golang
SQL
R Language
Android Tutorial
DSA Concepts
Data Structures
Arrays
Strings
Linked List
Algorithms
Searching
Sorting
Mathematical
Dynamic Programming
Dsa Roadmaps
Data Structures
Dsa for Beginners
Basic DSA Coding Problems
Complete Roadmap To Learn DSA
DSA for Frontend Developers
DSA with JavaScript
Top 20 DSA Interview Problems
All Cheat Sheets
DSA Roadmap by Sandeep Jain
Web Development
HTML
CSS
JavaScript
Bootstrap
ReactJS
AngularJS
NodeJS
Express.js
Lodash
Competitive Programming
Top DSA for CP
Top 50 Tree Problems
Top 50 Graph Problems
Top 50 DP Problems
Top 50 Array Problems
Top 50 String Problems
Top 50 Greedy Problems
Top 31 Algorithms for CP
System Design
What is System Design
Monolithic and Distributed SD
Scalability in SD
Databases in SD
High Level Design or HLD
Low Level Design or LLD
Top SD Interview Questions
Computer Science
GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Python
Python Programming Examples
Django Tutorial
Python Projects
Python Tkinter
Web Scrapping
Python Numpy Tutorial
Python Interview Question
Data Science & ML
Machine Learning Tutorial
Maths For Machine Learning
Pandas Tutorial
Keras Tutorial
NumPy Tutorial
Pandas Tutorial
Deep Learning Tutorial
DevOps
Git
AWS
Docker
Kubernetes
Azure
GCP
Competitive Programming
Top 50 Array Problems
Top 50 Tree Problems
Top 50 Graph Problems
Top 50 DP Problems
Top 50 Array Problems
Top 50 String Problems
Top 50 Greedy Problems
Top 31 Algorithms for CP
Interview Experience
Interview Preparation
Preparation for SDE
Experienced Interviews
Internship Interviews
Competitive Programming
Aptitude Preparation
GFG School
CBSE Notes for Class 8
CBSE Notes for Class 9
CBSE Notes for Class 10
CBSE Notes for Class 11
CBSE Notes for Class 12
English Grammar
Commerce
Accountancy
Business Studies
Economics
Income Tax
Finance
Statistics for Economics
UPSC
Polity Notes
Geography Notes
History Notes
Science and Technology Notes
Economics Notes
Important Topics in Ethics
UPSC Previous Year Papers
SSC/ BANKING
SSC CGL Syllabus
SBI PO Syllabus
SBI Clerk Syllabus
IBPS PO Syllabus
IBPS Clerk Syllabus
Aptitude Questions
SSC CGL Practice Papers
Write & Earn
Write an Article
Improve an Article
Pick Topics to Write
Write Interview Experience
Internships
geeksforgeeks. Some rights reserved