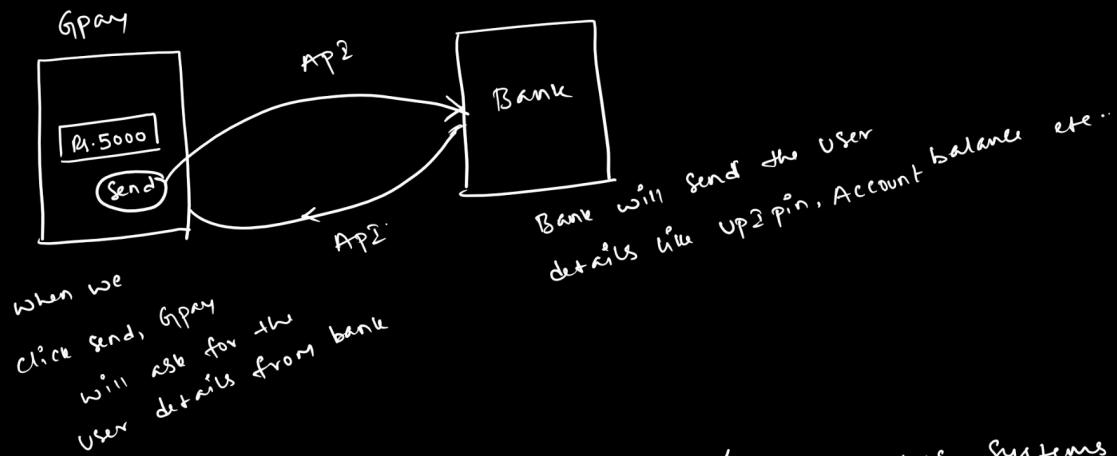


## API - POSTMAN & FLASK

→ API - Application programming interface.



→ API acts as bridge between homogeneous/heterogeneous systems.

↳ It sends & receives the data.

→ API is again used to access the functions & methods in a system.

for eg.. we had used MySQL with python.

In this we had installed my-sql connector which connects python with MySQL.

⇒ Also Jupyter notebook, when we write a code in Jupyter notebook; it doesn't execute our code. It sends our code to python compiler through an API, receives an o/p & shows to us.



Frameworks → Flask, Django

Procedures → SOAP ← REST  
(Representational State Transfer)

(Simple object  
Access protocol)

↓  
IS a protocol

Nowadays for building of  
Apps/API - SOAP is not used.

Apps/API - SOAP is not used.

↓  
IS an Architecture.

↓  
REST is being used  
mostly nowadays.

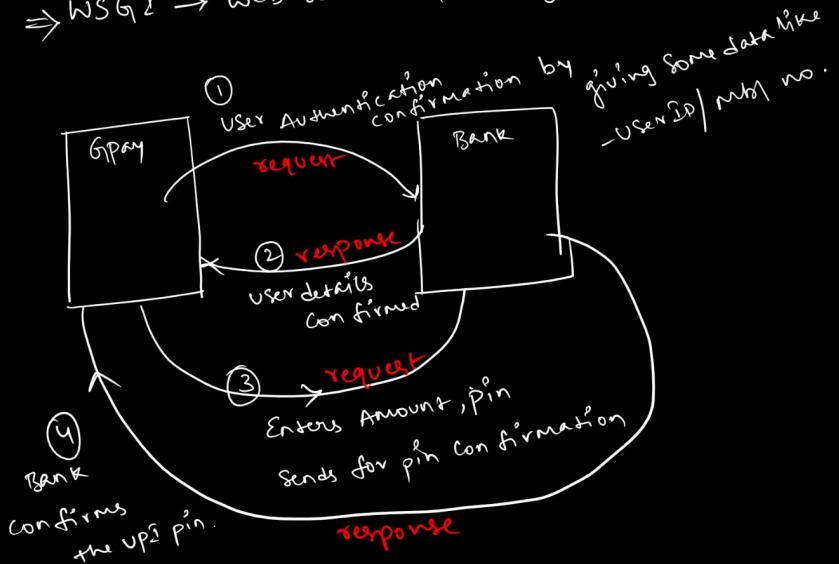
⇒ SOAP/REST is a way of creating APIs.  
(Protocol)

## Flask:

Flask - It is a micro web development framework.

- ↳ It internally uses 2 frameworks
  - \* Werkzeug ← WSGI (to request something from other programs)
  - \* Jinja (for templating)

⇒ WSGI → web server Gateway Interface



→ Methods: GET, POST, PUT, DELETE.

Ideally used to request data by sending some data. Both are used for sending the data. are HTTP methods.

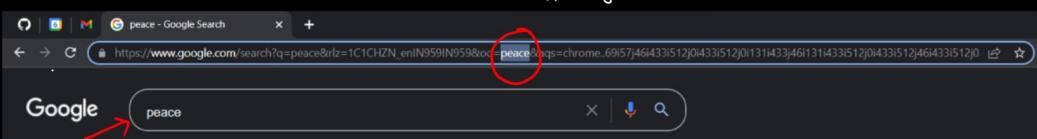
Google search

GET & POST both are used for sending the data. are HTTP methods.

POST
 

- ↳ Gmail
- ↳ Authentication
- ↳ here also we will send the data but it won't be appended in the URL.  
e.g.: g-mail ID, password - won't be displayed in the URL.
- ↳ Here we will send the data in the body of API.

We can see our keyword which we had searched in the URL. (URL + query) like google.com/peace.



GET → when GET method is used, the data which we send/enter will be appended in the URL (URL + query)

→ the data will be visible.

POST → The data will be sent in the body

→ The data won't be displayed in URL & will be secured.

PUT → used to update

DELETE → used to delete.

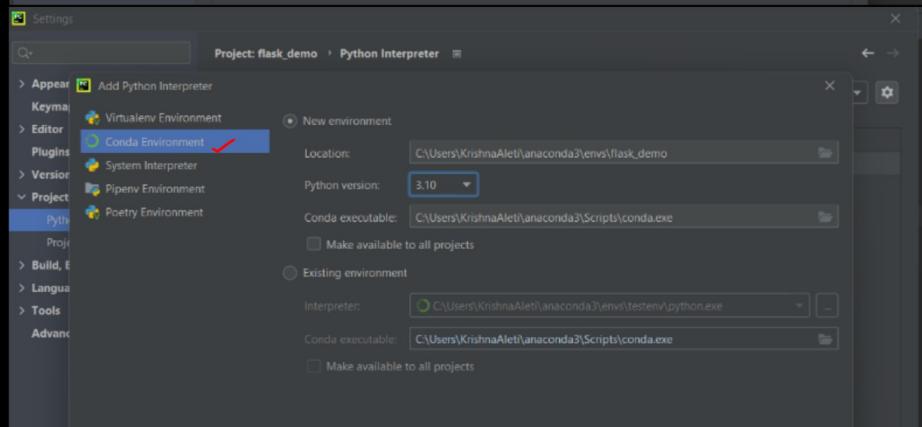
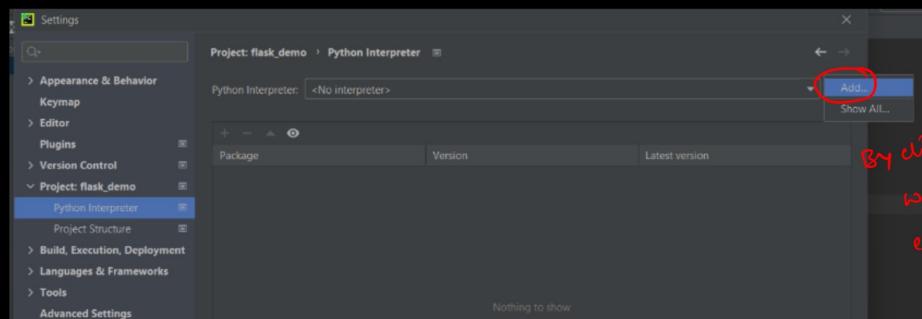
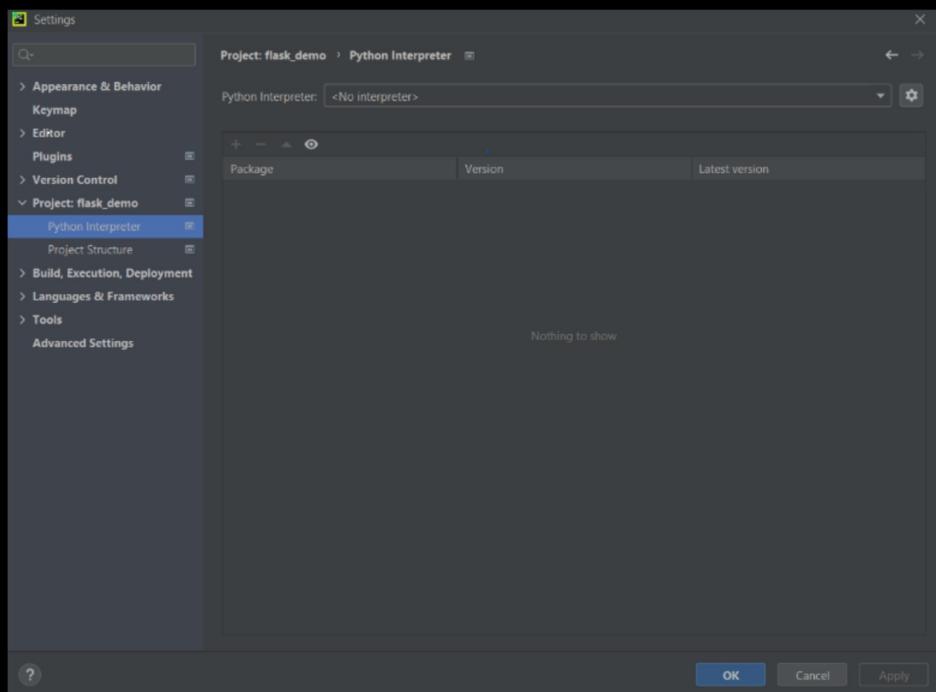
What is difference between PATCH and PUT in REST API?

The main difference between PUT and PATCH in REST API is that PUT handles updates by replacing the entire entity, while PATCH only updates the fields that you give it. PATCH does not change any of the other values. If you use the PUT method, then everything will get updated. 14-Oct-2021

Flask Demo:

PyCharm - Python Interpreter Setup:

File > settings ↴



OK > Apply > OK

In our flask, we follow an architecture (REST) for eg.  $\rightarrow$  we had appended the function we need in the `@app.route`.

```

from flask import Flask, render_template, request, jsonify
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST']) # To render Homepage
def home_page():
    return render_template('index.html')

@app.route('/math', methods=['POST']) # This will be called from UI
def math_operation():
    if (request.method=='POST'):
        operation=request.form['operation']
        num1=int(request.form['num1'])
        num2 = int(request.form['num2'])
        if(operation=='add'):
            r=num1+num2
            result= 'the sum of ' +str(num1)+ ' and ' +str(num2) +' is '+str(r)
        if (operation == 'subtract'):
            r = num1 - num2
            result = 'the difference of ' +str(num1)+ ' and ' +str(num2) +' is '+str(r)
        if (operation == 'multiply'):
            r = num1 * num2
            result = 'the product of ' +str(num1)+ ' and ' +str(num2) +' is '+str(r)
        if (operation == 'divide'):
            r = num1 / num2
            result = 'the quotient when ' +str(num1)+ ' is divided by ' +str(num2) +' is '+str(r)
    return jsonify(result)

if __name__ == '__main__':
    app.run()
    # app.run(debug=True)

```

Terminal: Command Prompt  
Microsoft Windows [Version 10.0.22000.593]  
(c) Microsoft Corporation. All rights reserved.

flask\_demo D:\Edu\Da ta Science \_iNeuronLive\_Class\_Notes\Flask\21.Flask\_iNeuron\flask\_demo>pip install flask

Collecting flask  
 Downloading Flask-2.0.3-py3-none-any.whl (95 kB)  
Collecting Werkzeug>2.0  
 Downloading Werkzeug-2.0.3-py3-none-any.whl (289 kB)

4.1 CRLF UTF-8 4 spaces Python 3.6 (flask\_demo) Event Log

$\rightarrow$  we are transferring one function to other model.  
 $\rightarrow$  we are able to invoke our method/function using our route.  
 $\rightarrow$  By default we use REST procedure in Flask, Django.

```

from flask import Flask, render_template, request, jsonify
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST']) # To render Homepage
def home_page():
    return render_template('index.html')

@app.route('/math', methods=['POST']) # This will be called from UI
def math_operation():
    if (request.method=='POST'):
        operation = request.form['operation']
        num1 = int(request.form['num1'])
        num2 = int(request.form['num2'])
        if (operation == 'add'):
            r = num1 + num2
            math_operation() >> if(request.method == POST) if(operation == 'add')

```

Terminal: Command Prompt  
Downloading dataclasses-0.8-py3-none-any.whl (19 kB)  
Collecting typing\_extensions>3.6.4  
Downloading typing\_extensions-4.1.1-py3-none-any.whl (26 kB)  
Collecting zipp>0.5  
Downloading zipp-3.6.0-py3-none-any.whl (5.3 kB)  
Installing collected packages: zipp, typing\_extensions, MarkupSafe, importlib-metadata, dataclasses, colorama, Werkzeug, Jinja2, itsdangerous, click, flask  
Successfully installed Jinja2-3.0.3 MarkupSafe-2.0.1 Werkzeug-2.0.3 click-8.0.4 colorama-0.4.4 dataclasses-0.8 flask-2.0.3 importlib-metadata-4.8.3 itsdangerous-2.0.1 typing\_extensions-4.1.1  
Structure Alt+7

1733 CPU UTF-8 4 spaces Python 3.6 (flask\_demo) Event Log

$\rightarrow$  we are representing our function math-operation with our route, hence this is REST model.

$\rightarrow$  APP.ROUTE creates API/URL

Inputs to be given in Postman in JSON format

Newest Python function

Key instead of JSON we can use HTML, but also JSON is standard format.

Json format is identical dictionary  
 $\downarrow$   
key: value

```

from flask import Flask, render_template, request, jsonify
app = Flask(__name__)
 $\rightarrow$  app is the object of Flask class.
 $\rightarrow$  APP creates URL/API
@app.route('/via_postman', methods=['POST']) # for calling the API from Postman/SOAPUI
def math_operation_via_postman():
    if (request.method == 'POST'):
        operation = request.json['operation']
        num1 = int(request.json['num1'])
        num2 = int(request.json['num2'])
        if (operation == 'add'):
            r = num1 + num2
            result = 'the sum of ' + str(num1) + ' and ' + str(num2) +' is '+ str(r)
        if (operation == 'subtract'):
            r = num1 - num2
            result = 'the difference of ' +str(num1)+ ' and ' +str(num2) +' is '+str(r)
        if (operation == 'multiply'):
            r = num1 * num2
            result = 'the product of ' +str(num1)+ ' and ' +str(num2) +' is '+str(r)
        if (operation == 'divide'):
            r = num1 / num2
            result = 'the quotient when ' +str(num1)+ ' is divided by ' +str(num2) +' is '+str(r)
    return jsonify(result)

if __name__ == '__main__':
    app.run()
    # app.run(debug=True)

```

Terminal: Command Prompt  
Version Control TODO Problems Terminal Python Packages Python Console  
Shared indexes are downloaded for Python packages in 651 ms (453.21 kB) (55 minutes ago)

The screenshot shows the PyCharm IDE with the following details:

- Project:** flask\_demo
- File:** apppostman.py
- Code Content:**

```
from flask import Flask, render_template, request, jsonify

app = Flask(__name__)

@app.route('/via_postman', methods=['POST']) # for calling the API from Postman/SOAPUI
def math_operation_via_postman():
    if (request.method == 'POST'):
        operation = request.json['operation']
        num1 = int(request.json['num1'])
        num2 = int(request.json['num2'])

        if operation == '+':
            return jsonify({'result': num1 + num2})
        elif operation == '-':
            return jsonify({'result': num1 - num2})
        elif operation == '*':
            return jsonify({'result': num1 * num2})
        elif operation == '/':
            return jsonify({'result': num1 / num2})
        else:
            return jsonify({'error': 'Unknown operation'})
```

- Annotations:** Handwritten notes in pink ink are overlaid on the code area:
  - "Here this code is my backend code exposed as URL/API"
  - "To access this URL/API, this should be up and running."
  - "Just like the SERVER → here this code is my server."
- Run Tab:** Shows the command: C:\Users\KrishnaAleti\anaconda3\envs\flask\_demo\python.exe "D:\Edu\Data Science\_iNeuron\Live\_Class\_Notes\Flask/21.Flask\_iNeuron\flask\_demo\apppostman.py". The output shows the application is running on port 5000.

Postman

File Edit View Help

Home Workspaces API Network Reports Explore

Search Postman

My Workspaces New Import GET New Request

No Environment

Collections +

Postman Echo Python\_Postman

Python\_Postman / New Request

POST http://127.0.0.1:5000/via\_postman

Params Authorization Headers Body Pre-request Script Tests Settings

Query Params KEY VALUE DESCRIPTION

Key Value Description

Complete URL

Send Cookies

Mock Servers

Monitors

Flows

History

POSTMAN IS USED TO TEST APIs.

IP ADDRESS OF LOCAL SYSTEM → 127.0.0.1

FLASK PORT NUMBER → 5000

local ip + port → used to reach the system / server

app.route('path') → used to reach the function

Click Send to get a response

As we had used **POST** method,  
we are sending data in the body  
→ in **JSON** format (**key: Value**)  
(Same as **dictionary**)

The screenshot shows the Postman interface with a POST request to `http://127.0.0.1:5000/via_postman`. The request body is set to `raw` and `JSON`. The JSON payload is `{\"operation\": \"add\", \"num1\": \"3\", \"num2\": \"5\"}`. The response status is `200 OK` with the message `'the sum of 3 and 5 is 8'`.

```

from flask import Flask, render_template, request, jsonify

app = Flask(__name__)

@app.route('/via_postman', methods=['POST']) # for calling the API from Postman/SOAPUI
def math_operation_via_postman():
    if (request.method == 'POST'):
        operation = request.json['operation']
        num1 = int(request.json['num1'])
        num2 = int(request.json['num2'])

Run: apppostman x
C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\python.exe "D:/Edu/Data Science_iNeuron/Live_Class_Notes/Flask/21.Flask_iNeuron/flask_demo/apppostman.py"
* Serving Flask app 'apppostman' (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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [06/Apr/2022 23:36:48] "POST /via_postman HTTP/1.1" 200

```

```

from flask import Flask, render_template, request, jsonify

app = Flask(__name__)

@app.route('/test', methods=['POST']) # for calling the API from Postman/SOAPUI
def math_test():
    if (request.method == 'POST'):
        num0 = int(request.json['num0'])
        num1 = int(request.json['num1'])
        num2 = int(request.json['num2'])
        result = num0+num1+num2
        return jsonify(result)

if __name__ == '__main__':
    app.run()
# app.run(debug=True)

Run: appk x
C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\python.exe "D:/Edu/Data Science_iNeuron/Live_Class_Notes/Flask/21.Flask_iNeuron/flask_demo/appk.py"
* Serving Flask app 'appk' (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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [06/Apr/2022 23:54:54] "POST /testk HTTP/1.1" 404 -
127.0.0.1 - - [06/Apr/2022 23:55:01] "POST /test HTTP/1.1" 200 -

```

Postman

File Edit View Help

Home Workspaces API Network Reports Explore

My Workspace

+

http://127.0.0.1:5000/test

POST http://127.0.0.1:5000/test

Parameters Authorization Headers (B) Body Pre-request Script Tests Settings

Body

none form-data x-www-form-urlencoded raw binary GraphQL JSON

num0: 1 num1: 2 num2: 3

Send

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

1 3

```

Run: appk x
C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\python.exe "D:/Edu/Data Science_iNeuron/Live_Class_Notes/Flask/21.Flask_iNeuron/flask_demo/appk.py"
* Serving Flask app 'appk' (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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [06/Apr/2022 23:54:54] "POST /testk HTTP/1.1" 404 -
127.0.0.1 - - [06/Apr/2022 23:55:01] "POST /test HTTP/1.1" 200 ✓

```

In this we have 2 routes/post

ROUTE 1: `/via_post`

```

from flask import Flask, render_template, request, jsonify
app = Flask(__name__)
@app.route('/via_post', methods=['POST']) # for calling the API from Postman/SOAPUI
def math_postman():
    if (request.method == 'POST'):
        operation = request.json['operation']
        num1 = int(request.json['num1'])
        num2 = int(request.json['num2'])
        if (operation == 'add'):
            r = num1 + num2
            result = 'the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        if (operation == 'subtract'):
            r = num1 - num2
            result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        if (operation == 'multiply'):
            r = num1 * num2
            result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        return jsonify(result)

```

ROUTE 2: `/tester`

```

@app.route('/tester', methods=['POST']) # for calling the API from Postman/SOAPUI
def math_tester():
    if (request.method == 'POST'):
        num0 = int(request.json['num0'])
        num1 = int(request.json['num1'])
        num2 = int(request.json['num2'])
        result = num0+num1+num2
        return jsonify(result)

```

If name == 'main':  
`app.run()`  
`# app.run(debug=True)`  
`math_postman() if (request.method == 'POST')`

Yours via-post → function: math-postman  
Yours tester → function: math-tester  
So, when the server is running, In postman if we give route: via-post then math-postman should be executed.  
route: tester, In postman if we give route: tester, math-tester should be executed.

Flask gives 5000

Flask @ app.route creates `http://127.0.0.1:5000`

→ APP.route ('/tester') = `http://127.0.0.1:5000/tester`

This we give in postman.

POST http://127.0.0.1:5000/via\_post

Body

```

1
2
3     "operation": "multiply",
4     ... "num1": "3",
5     ... "num2": "5"
6
7
8

```

Status: 200 OK Time: 10 ms Size: 177 B Save Response

127.0.0.1 is the local host

This is an API URL

POST http://127.0.0.1:5000/tester

Body

```

1
2     ... "num0": 3,
3     ... "num1": 3,
4     ... "num2": 5
5

```

Status: 200 OK Time: 11 ms Size: 148 B Save Response

We had hit the API 2 times (`/via_post`, `/tester`) & both of them are successful. Hence, we get code: 200 - 2 times.

```

Run: app2.x
C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\python.exe "D:\Edu\Data Science_iNeuron/Live_Class_Notes/Flask/21.Flask_iNeuron\flask_demo\app2.py"
* Serving Flask app 'app2' (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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - [07/Apr/2022 08:05:33] "POST /via_post HTTP/1.1" 200 -
127.0.0.1 - [07/Apr/2022 08:06:44] "POST /tester HTTP/1.1" 200 -

```

As local host = 127.0.0.1 → Instead of `http://127.0.0.1:5000/tester` we can also give `localhost:5000/tester`.

→ `http://127.0.0.1:5000/tester` & `localhost:5000/tester`

URL/API

These helps us reach the particular function & execute.

This helps to reach the particular system using the IP address  
 $\downarrow$   
 $(http://127.0.0.1 \text{ or } \text{localhost})$

then to the server/service which is up and running using the port number (5000),

then reaches the function `math-tester()` using the route `'/tester'`.

→ Here the code in the pycharm acts as Server & postman is client.

→ Here the code in the pycharm acts as Server & postman is client.  
 $\downarrow$   
 Serves the request send from the postman.

→ Currently our server "http://127.0.0.1:5000" is present in the local, when we

expose our API globally we get https in the URL.

→ Pythonic Framework - we can use Flask with Python code only.

→ Flask - Micro Web Framework  $\Rightarrow$  Flask is used to create RESTful APIs.

↓  
Uses REST

↓  
Easy to use/install

↓  
Is a RESTful API web framework.

→ API is created by Flask, so API is a microservice.

↓  
uses REST, hence REST API

↓  
Can integrate in web, hence it is web framework.

→ Flask is a Pythonic Framework - Flask can be used with Python only.

but the API that we had created using Flask can be exposed

to other programs for usage. For eg.. A program written in JAVA can

consume the API created by Flask (Python code).

## FLASK

```
file Edit View Navigate Code Defactor Run Tools VCS Window Help flask_demo - app1.py

flask_demo app1.py
Project flask demo D:\Edu\I Data Science\jNeuron\Live_Class_Notes\Flask
  - flask demo
    - static
      - main.css
      - style.css
    - templates
      - index.html
      - results.html
    - app.py
    - app1.py
    - app2.py
    - appk.py
    - appl2.py
    - appostman.py
  - External Libraries
  - Scratches and Consoles

<h1>
  ↓
  reading tag in hand
```

```
appostman.py appk.py appl2.py app.py app1.py app2.py

50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73

return jsonify(result)

@app.route('/krish_function')
def url_test1():
    test1 = request.args.get('val1')
    test2 = request.args.get('val2')
    test3 = int(test1) + int(test2)

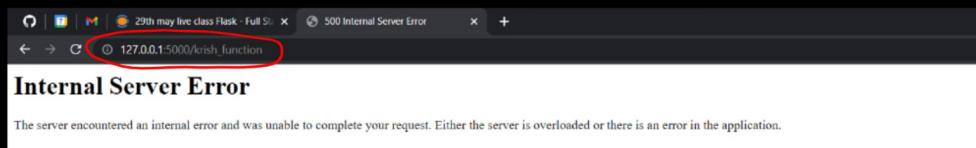
    return '''<h1>my result is : {}</h1>'''.format(test3)

@app.route('/url_function')
def url_data():
    test = request.args.get('val1')
    test1 = request.args.get('val2')
    test3 = int(test) + int(test1)

    return '''result {}'''.format(test3)

if __name__ == '__main__':
    app.run(debug=True)
```

we are using GET method here.  
here we will get the values from browser.  
returning result to the browser itself.  
(html page)  
Browser will send request to this program  
and this program will try to get the  
values ('val1', 'val2') from the url  
http://127.0.0.1:5000/krish-function



by default when we send the data in the url, it will be considered as string & then test1 + test2 means those 2 values will be concatenated as we add the sum of 2 values we used `int(test1) + int(test2)`

File: flask\_demo > app1.py

```
if __name__ == '__main__':
    app.run('/krish_function')

def url_test():
    test1 = request.args.get('val1')
    test2 = request.args.get('val2')
    test3 = int(test1) + int(test2)

    return '''<h1>my result is : {}</h1>'''.format(test3)
```

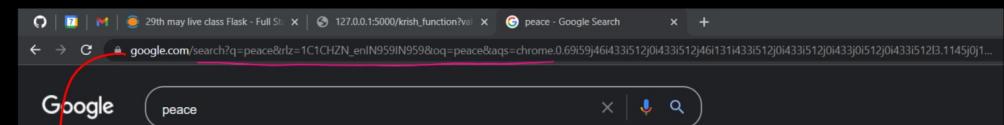
Run: app1(1) ×

```
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
[2022-04-07 19:20:15,134] ERROR in app: Exception on /krish_function [GET]
Traceback (most recent call last):
  File "C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\lib\site-packages\flask\app.py", line 2073, in wsgi_app
    response = self.full_dispatch_request()
  File "C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\lib\site-packages\flask\app.py", line 1518, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\lib\site-packages\flask\app.py", line 1516, in full_dispatch_request
    rv = self.dispatch_request()
  File "C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\lib\site-packages\flask\app.py", line 1502, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
  File "D:\Edu\Data_Science_iNeuron/Live_Class_Notes/Flask/21.Flask_iNeuron/flask_demo/app1.py", line 57, in url_test
    test3 = int(test1) + int(test2)
TypeError: int() argument must be a string, a bytes-like object or a number, not 'NoneType'
127.0.0.1 - - [07/Apr/2022 19:20:15] "GET /krish_function HTTP/1.1" 500 -
```

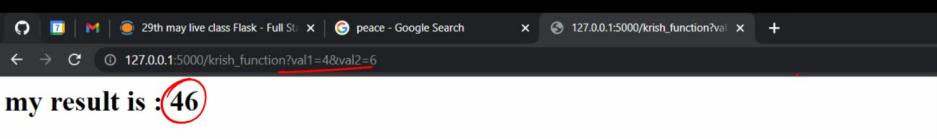
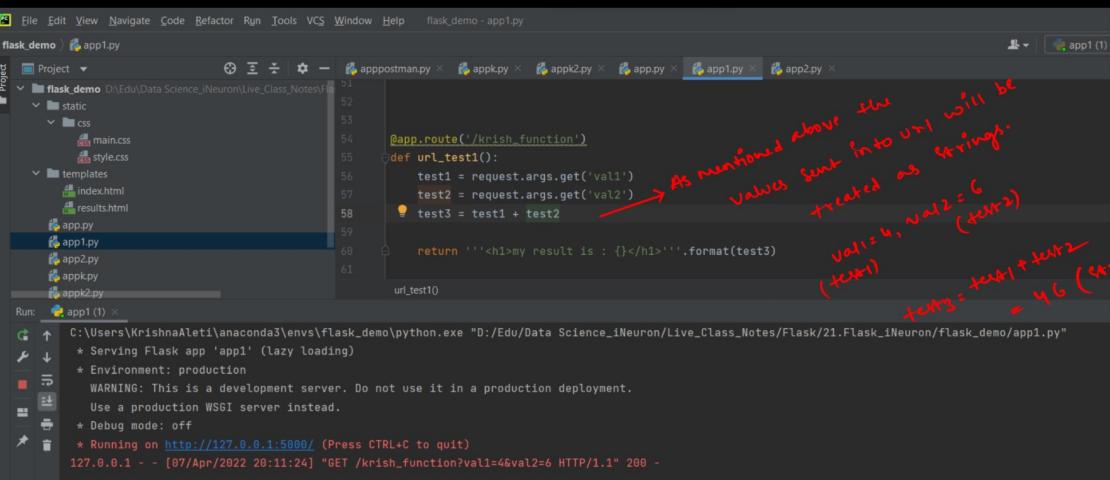
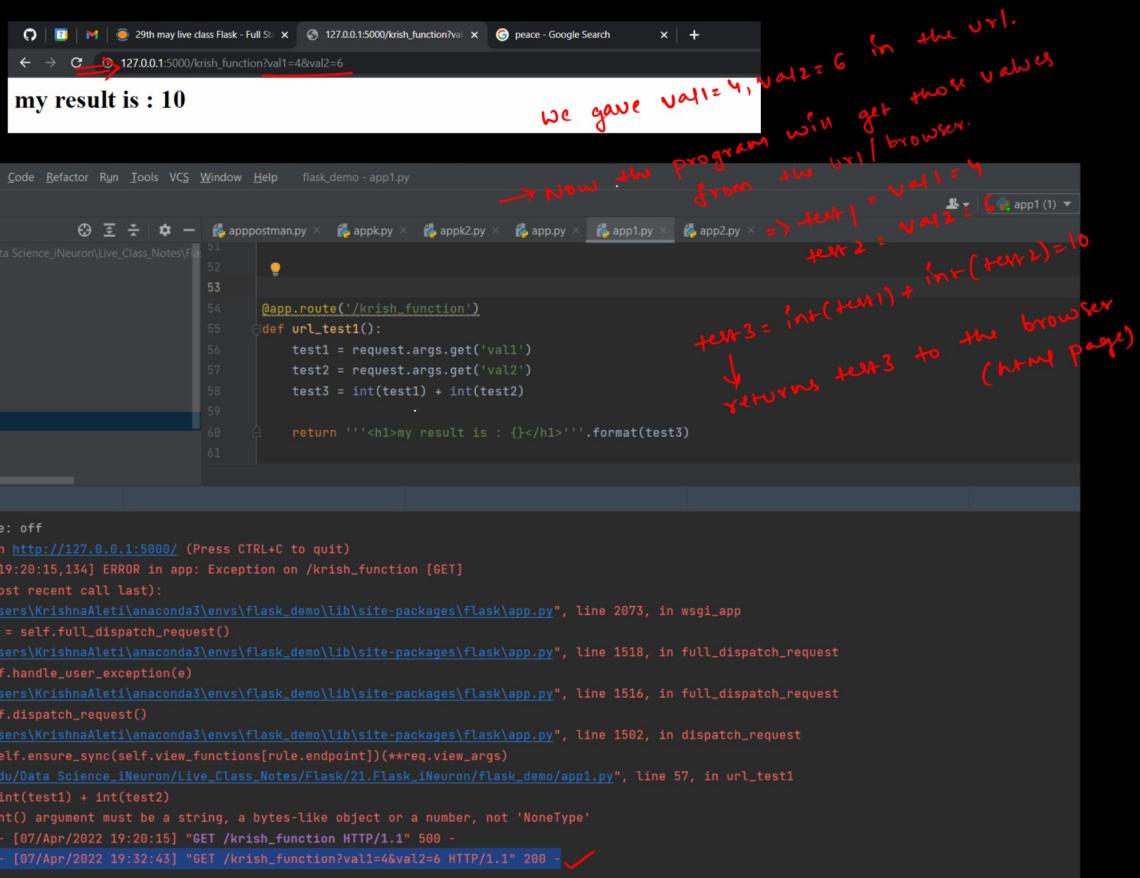
by default when we send the data in the url, it will be considered as string & then test1 + test2 means those 2 values will be concatenated as we need sum of 2 values we used  $\text{int}(\text{test1}) + \text{int}(\text{test2})$

✓ we didn't send any values hence test1, test2 are None.

→ To resolve this issue, we need to send data with the url.



Similarly now we want to pass data in the browser → `http://127.0.0.1:5000/krish-function?val1=4&val2=6`  
 => Notation for search → URL ? q = param & val = .....  
 => only we give the values in URL → these 2 values val1(6), val2(4) will be sent to the program, the program will execute and returns the result back to browser (because in the program/code, we had mentioned to return the o/p to the browser).



```

flask_demo app1.py
Project flask_demo D:\Edu\Python\Flask\21.Flask_iNeuron\flask_demo\app1.py
static css main.css style.css
templates index.html results.html
app.py app1.py app2.py appk.py appk2.py
Run: app1 (1) x
C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\python.exe "D:/Edu/Python/Flask/21.Flask_iNeuron/flask_demo/app1.py"
* Serving Flask app 'app1' (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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
[27.0.0.1 - - [07/Apr/2022 20:11:24] "GET /krish_function?val1=4&val2=6 HTTP/1.1" 200 

```

Let us try the same from the postman:

```

flask_demo app1.py
Project flask_demo D:\Edu\Python\Flask\21.Flask_iNeuron\flask_demo\app1.py
static css main.css style.css
templates index.html results.html
app.py app1.py app2.py appk.py appk2.py
Run: app1 (1) x
C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\python.exe "D:/Edu/Python/Flask/21.Flask_iNeuron/flask_demo/app1.py"
* Serving Flask app 'app1' (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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
[27.0.0.1 - - [07/Apr/2022 20:23:21] "GET /krish_function?val1=4&val2=6 HTTP/1.1" 200 -
[27.0.0.1 - - [07/Apr/2022 20:23:49] "GET /krish_function?val1=4&val2=6 HTTP/1.1" 200 -
[27.0.0.1 - - [07/Apr/2022 20:24:00] "GET /krish_function?val1=4&val2=6 HTTP/1.1" 200 -

```

My Workspace

POST http://127.0.0.1:5000/krish\_function?val1=4&val2=6

GET http://127.0.0.1:5000/krish\_function?val1=14&val2=16

Params Authorization Headers (6) Body Pre-request Script Tests Settings

KEY	VALUE
val1	14
val2	16

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize HTML

1. <h1>my result is : 30</h1>

Url + Values

when we give values in the url, they will be automatically populated under the params.

(here postman doesn't understand the html format hence it displays whatever is returned by the program).

My Workspace

POST http://127.0.0.1:5000/krish\_function?val1=4&val2=6

GET http://127.0.0.1:5000/krish\_function?val1=5&val2=4

Params Authorization Headers (6) Body Pre-request Script Tests Settings

KEY	VALUE
val1	5
val2	4

Initial give http://127.0.0.1:5000/krish\_function (Select GET)

then gives key, values under the params

they gets added to url

=> automatically they are added to url

→ http://127.0.0.1:5000/krish\_function?val1=5&val2=4

```

1 from flask import Flask, render_template, request, jsonify
2
3 app = Flask(__name__)
4
5 @app.route('/krish_function')
6 def url_test1():
7     test1 = request.args.get('val1')
8     test2 = request.args.get('val2')
9     test3 = int(test1) + int(test2)
10
11     return '''<h3>my result is : {}</h3>'''.format(test3)
12
13
14 @app.route('/url_function')
15 def url_data():
16     test = request.args.get('val1')
17     test1 = request.args.get('val2')
18     test3 = int(test) - int(test1)
19
20     return '''result {}'''.format(test3)
21
22
23 if __name__ == '__main__':
24     # app.run(debug=True)
25     app.run()

```

2 routes - 2 functions

*can use any of the methods GET/POST*

*@app.route ('/')*  
views  
the root url  
*http://127.0.0.1:5000/*  
when 2 hit the root url  
homepage function will be executed.  
render-template (*index.html*)  
this says when we hit the root url, execute the *index.html*  
→ renders *index.html*

*def home\_page():*  
return render\_template('index.html')

*@app.route ('/mathops', methods=['POST'])* # This will be called when we hit the homepage

*def math\_operation():*

- if (request.method == 'POST'):*  
operation = request.form['operation']  
num1 = int(request.form['num1'])  
num2 = int(request.form['num2'])  
if (operation == 'add'):  
 r = num1 + num2  
 result = 'the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
- if (operation == 'subtract'):*  
 r = num1 - num2  
 result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
- if (operation == 'multiply'):*  
 r = num1 \* num2  
 result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
- if (operation == 'divide'):*  
 r = num1 / num2  
 result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)

*if \_\_name\_\_ == '\_\_main\_\_':*  
# app.run(debug=True)  
app.run()

*whatever is mentioned in individual will be displayed in the homepage.*  
In the homepage we have submit calculate button when it is hit, the path routes to the function in it & the function in it is executed.

*index.html* → need to give template  
we can also change the name but then we need to give full path instead of just *index.html*

(HTML behaves like the interpreter, it doesn't execute the entire code, it will just parse & whatever we had written it will populate all and will display)

→ In 'templates' folder we keep the HTML files (*index.html*, *results.html*)

→ static > css > main.css, style.css (CSS - Cascading Style Sheet)

⇒ render-template (*index.html*)

↳ It just renders the HTML file - shows whatever is there in HTML file.

```

body {
    background-color: #cccccc;
}
body * {
    box-sizing: border-box;
}
.header {
    background-color: #327fa8;
    color: white;
    font-size: 1.5em;
    padding: 1em;
    text-align: center;
    text-transform: uppercase;
}

```

for the background color of the whole page.

(for the frontend - HTML code we can use getbootstrap.com. copy the code & use in our HTML pages.)

```

flask_demo > templates > index.html
app2.py index.html results.html style.css

1  {% block head %}
2
3  <title>MathOperations</title>
4  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}>
5  {% endblock %}

6  {% block body %}
7  <div class="content">
8    <form action="/mathops" method="POST">
9      <label for="operation">Choose a Mathematical Operation</label>
10     <select id="operation" name="operation">
11       <option value="add">add</option>
12       <option value="subtract">subtract</option>
13       <option value="multiply">multiply</option>
14       <option value="divide">divide</option>
15     </select>
16     <input type="text" name="num1" id="num1">
17     <input type="text" name="num2" id="num2">
18     <input type="submit" value="Calculate">
19   </form>
20 </div>
21 {% endblock %}

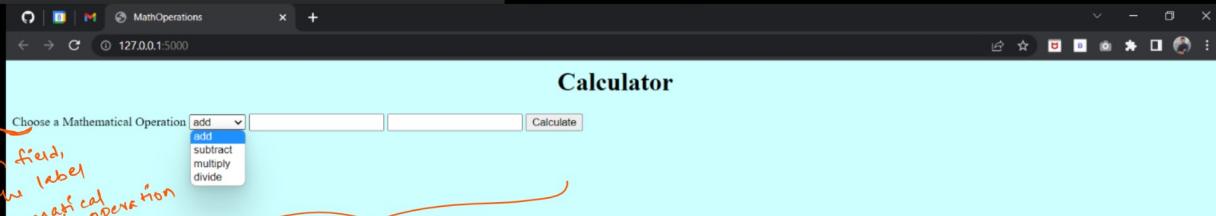
```

```

app2.py index.html results.html style.css

1  from flask import Flask, render_template, request, jsonify
2  app = Flask(__name__)
3  @app.route('/', methods=['GET', 'POST']) # To render Homepage
4  def home_page():
5    return render_template('index.html')
6  @app.route('/mathops', methods=['POST']) # This will be called from UI
7  def math_operation():
8    if (request.method == 'POST'):
9      operation = request.form['operation']
10     num1 = int(request.form['num1'])
11     num2 = int(request.form['num2'])
12     if (operation == 'add'):
13       r = num1 + num2
14       result = 'the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
15     if (operation == 'subtract'):
16       r = num1 - num2
17       result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
18     if (operation == 'multiply'):
19       r = num1 * num2
20       result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
21     if (operation == 'divide'):
22       r = num1 / num2
23       result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)
24    return render_template('results.html', result=result)
25  if __name__ == '__main__':
26    app.run(debug=True)
27  app.run()

```



for operation field we gave the label choose a mathematical operation  
It is having a dropdown & the value that we select from dropdown is the value for 'operation'

```

flask_demo > templates > index.html
app2.py index.html results.html style.css

1  {% block head %}
2
3  <title>MathOperations</title>
4  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}>
5  {% endblock %}

6  {% block body %}
7  <div class="content">
8    <h1 style="text-align: center">Calculator</h1>
9
10   <div class="form">
11     <form action="/mathops" method="POST">
12       <label for="operation">Choose a Mathematical Operation</label>
13
14      <select id="operation" name="operation">
15        <option value="add">add</option>
16        <option value="subtract">sub</option>
17        <option value="multiply">mult</option>
18        <option value="divide">div</option>
19      </select>
20
21      <input type="text" name="num1" id="num1">
22      <input type="text" name="num2" id="num2">
23      <input type="submit" value="Calculate">
24    </form>
25  </div>
26 {% endblock %}

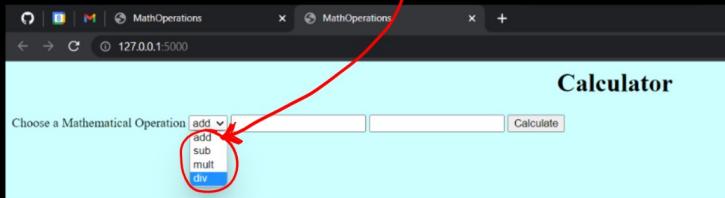
```

```

flask_demo > app2.py
app2.py index.html results.html style.css

1  from flask import Flask, render_template, request, jsonify
2  app = Flask(__name__)
3  @app.route('/', methods=['GET', 'POST']) # To render Homepage
4  def home_page():
5    return render_template('index.html')
6  @app.route('/mathops', methods=['POST']) # This will be called from UI
7  def math_operation():
8    if (request.method == 'POST'):
9      operation = request.form['operation']
10     num1 = int(request.form['num1'])
11     num2 = int(request.form['num2'])
12     if (operation == 'add'):
13       r = num1 + num2
14       result = 'the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
15     if (operation == 'subtract'):
16       r = num1 - num2
17       result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
18     if (operation == 'multiply'):
19       r = num1 * num2
20       result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
21     if (operation == 'divide'):
22       r = num1 / num2
23       result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)
24    return render_template('results.html', result=result)
25  if __name__ == '__main__':
26    app.run(debug=True)
27  app.run()

```



for the field 'operation', we gave the label: choose a math operation and for it we had given a dropdown. So, when we select a value from the dropdown, that value is for the 'Operation' field.  
This value of the Operation field will be given to the 'operation' variable of the program & as per it, the program gets executed & displays results.

```

flask_demo > templates > index.html
app2.py index.html results.html style.css

1  {% block head %}
2
3  <title>MathOperations</title>
4  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}>
5  {% endblock %}>
6  {% block body %}
7  <div class="content">
8    <h1 style="text-align: center">Calculator</h1>
9
10   <div class="form">
11     <form action="/mathops" method="POST">
12       <label for="operation">Choose a Mathematical Operation</label>
13
14      <select id="operation" name="operation">
15        <option value="add">add</option>
16        <option value="subtract">sub</option>
17        <option value="multiply">mult</option>
18        <option value="divide">div</option>
19      </select>
20
21      <input type="text" name="num1" id="num1">
22      <input type="text" name="num2" id="num2">
23      <input type="submit" value="Calculate">
24    </form>
25  </div>
26 {% endblock %}

```

here, we get values from the 'Operation' field of browser.

```

def math_operation():
  if (request.method == 'POST'):
    operation = request.form['operation']
    num1 = int(request.form['num1'])
    num2 = int(request.form['num2'])
    if (operation == 'add'):
      r = num1 + num2
      result = 'the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
    if (operation == 'subtract'):
      r = num1 - num2
      result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
    if (operation == 'multiply'):
      r = num1 * num2
      result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
    if (operation == 'divide'):
      r = num1 / num2
      result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)
  return render_template('results.html', result=result)
if __name__ == '__main__':
  app.run(debug=True)
  app.run()

```

app2.py

```

1  {% block head %}
2  <title>MathOperations</title>
3  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}>
4  {% endblock %}
5  {% block body %}
6  <div class="content">
7    <h1 style="text-align: center">Calculator</h1>
8
9    <div class="form">
10      <form action="/mathops" method="POST">
11        <label for="operation">Choose a Mathematical Operation</label>
12        <select id="operation" name="operation">
13          <option value="add">add</option>
14          <option value="subtract">sub</option>
15          <option value="multiply">mult</option>
16          <option value="divide">div</option>
17        </select>
18
19        <input type="text" name="num1" id="num1">
20        <input type="text" name="num2" id="num2">
21        <input type="submit" value="Calculate">
22      </form>
23    </div>
24  {% endblock %}

```

Here the 'Calculate' button is used to submit the values entered.

When we click calculate, then the operation, num1, num2 will be sent to the server (python program).

we get these values from the browser.

Calculator

Choose a Mathematical Operation

Here 'calculate' is the submit button.  
So, when we submit the form, the action will be performed. The action is '/mathops', so the function defined in the path '/mathops' will be executed by getting the Operation, num1, num2 from Operation, num1, num2 (entered in the browser).

app2.py

```

1  {% block head %}
2  <title>MathOperations</title>
3  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}>
4  {% endblock %}
5  {% block body %}
6  <div class="content">
7    <h1 style="text-align: center">Calculator</h1>
8
9    <div class="form">
10      <form action="/mathops" method="POST">
11        <label for="operation">Choose a Mathematical Operation</label>
12        <select id="operation" name="operation">
13          <option value="add">add</option>
14          <option value="subtract">sub</option>
15          <option value="multiply">mult</option>
16          <option value="divide">div</option>
17        </select>
18
19        <input type="text" name="num1" id="num1">
20        <input type="text" name="num2" id="num2">
21        <input type="submit" value="Calculate">
22      </form>
23    </div>
24  {% endblock %}

```

app2.py

```

1  from flask import Flask, render_template, request, jsonify
2  app = Flask(__name__)
3  @app.route('/', methods=['GET', 'POST']) # To render Homepage
4  def home_page():
5    return render_template('index.html')
6  @app.route('/mathops', methods=['POST']) # This will be called from UI
7  def math_operation():
8    if (request.method == 'POST'):
9      operation = request.form['operation']
10     num1 = int(request.form['num1'])
11     num2 = int(request.form['num2'])
12     if (operation == 'add'):
13       r = num1 + num2
14     if (operation == 'subtract'):
15       r = num1 - num2
16     result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
17     if (operation == 'multiply'):
18       r = num1 * num2
19     result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
20     if (operation == 'divide'):
21       r = num1 / num2
22     result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)
23
24    return render_template('results.html', result=result)
25
26 if __name__ == '__main__':
27   app.run(debug=True)
28
29

```

① As per the operation, we get the result.  
② that result is stored in the 'result' variable.  
The result that we get after executed is stored in the variable named result (variable name).

Flow: we run the server, open the browser (<http://127.0.0.1:5000>)

→ the homepage (whatever written in index.html) will be displayed

→ we select the operation from the dropdown,

The result that we get after executed is stored in the variable named result of the results.html

enter the num1, num2 values & click calculate.

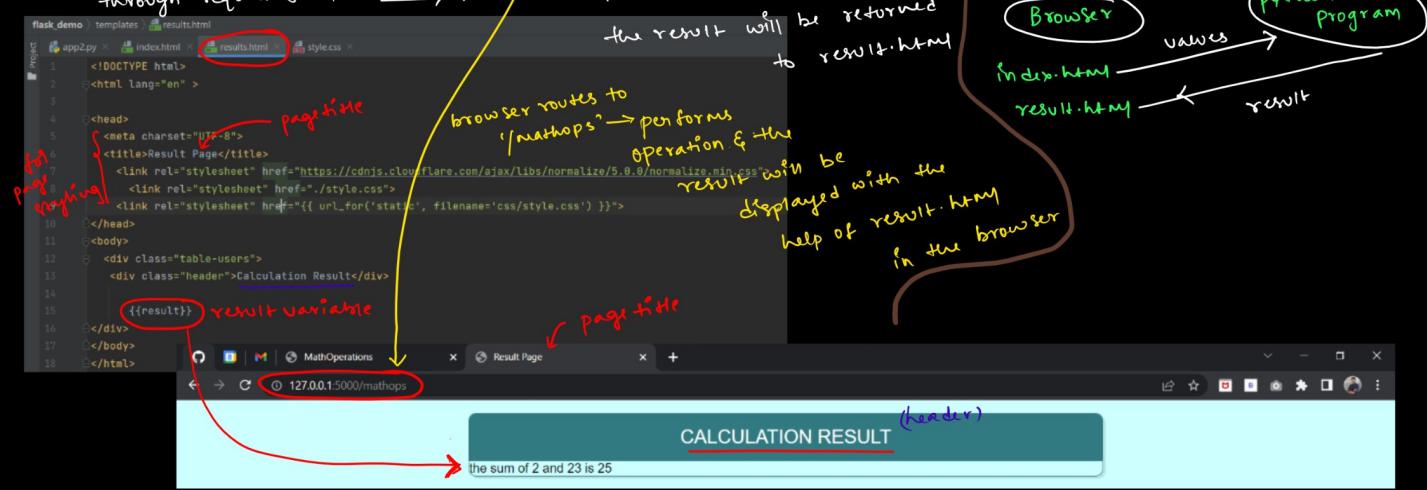
will

then the page route to '/mathops' (action='/mathops' for this form)

⇒ the function defined in the '/mathops' path → math-operation()

will be executed by getting the operation, num1, num2 from the UI through request.form.

Now the operation will be performed & the result will be returned to result.html



## Queries:

```
"""for calling the API from Postman/SOAPUI"""
@app.route('/via_postman', methods=['POST'])
def math_operation_via_postman():
    if (request.method == 'POST'):
        operation = request.json['operation']
        num1 = int(request.json['num1'])
        num2 = int(request.json['num2'])

        if (operation == 'add'):
            r = num1 + num2
            result = 'the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        if (operation == 'subtract'):
            r = num1 - num2
            result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        if (operation == 'multiply'):
            r = num1 * num2
            result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        if (operation == 'divide'):
            r = num1 / num2
            result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)
    return jsonify(result)
```

← we will send data  
from the postman

```
"""from the browser and also from postman"""
@app.route('/krish_function')
def url_test():
    test1 = request.args.get('val1')
    test2 = request.args.get('val2')
    test3 = int(test1) + int(test2)
    return """<h3>my result is : {}</h3>""".format(test3) # http://127.0.0.1:5000/krish_function?val1=4&val2=6
```

method = GET

← The data will be appended to the URL

```
"""from the browser"""
@app.route('/', methods=['GET', 'POST']) # To render Homepage
def home_page():
    return render_template('index.html')

@app.route('/math', methods=['POST']) # This will be called from UI
def math_operation():
    if (request.method=='POST'):
        operation=request.form['operation']
        num1=int(request.form['num1'])
        num2 = int(request.form['num2'])

        if(operation=='add'):
            r=num1+num2
            result='the sum of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        if (operation == 'subtract'):
            r = num1 - num2
            result = 'the difference of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        if (operation == 'multiply'):
            r = num1 * num2
            result = 'the product of ' + str(num1) + ' and ' + str(num2) + ' is ' + str(r)
        if (operation == 'divide'):
            r = num1 / num2
            result = 'the quotient when ' + str(num1) + ' is divided by ' + str(num2) + ' is ' + str(r)
    return render_template('results.html',result=result)
```

← we will have  
index.html, result.html

In the browser (form),  
we will enter the data.