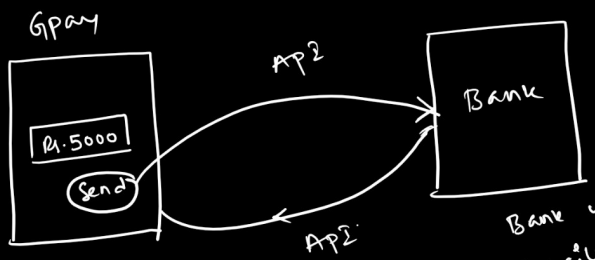
# API - POSTMAN & FLASK

→ API - Application programming Interface.




→ API acts as bridge between homegeneous/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 mysql 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
↓
Now a days for building of Apps/API - SOAP is not used.

↓
Is an Architecture.
↓
REST is being used mostly now-a-days.

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

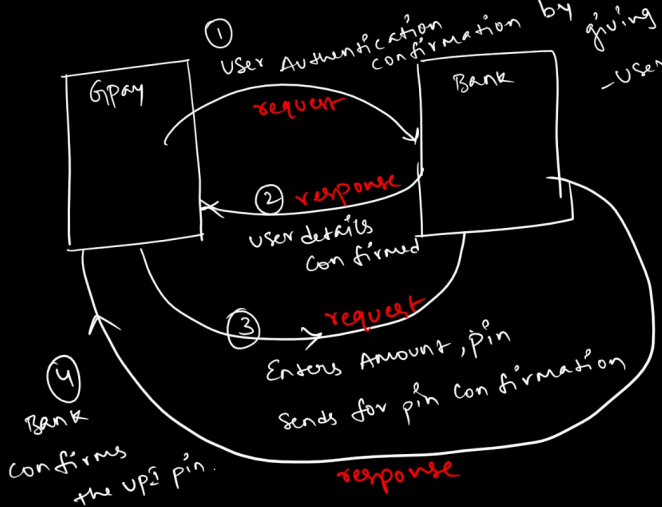
# Flask:

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.

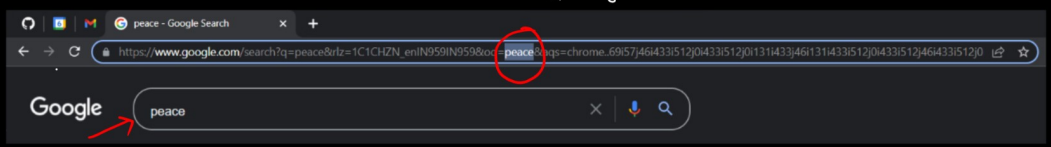
GET } *** both are used for sending the data.
POST } ↳ are HTTP methods.

Ideally used to request data by sending some data.

GET ← Google search

POST → Gmail Authentication → here also we will send the data but it won't be appended in the url.
eg: gmail 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
↳ data won't be displayed in url & will be secured.

PUT → used to update

DELETE → used to delete.

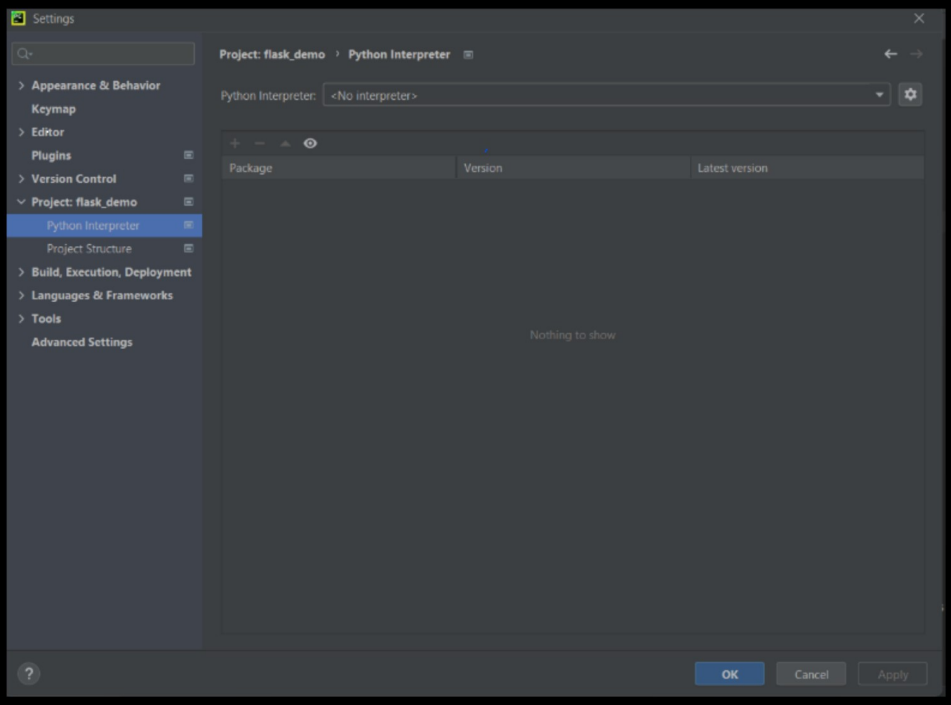
What is difference between PATCH and put in REST API?

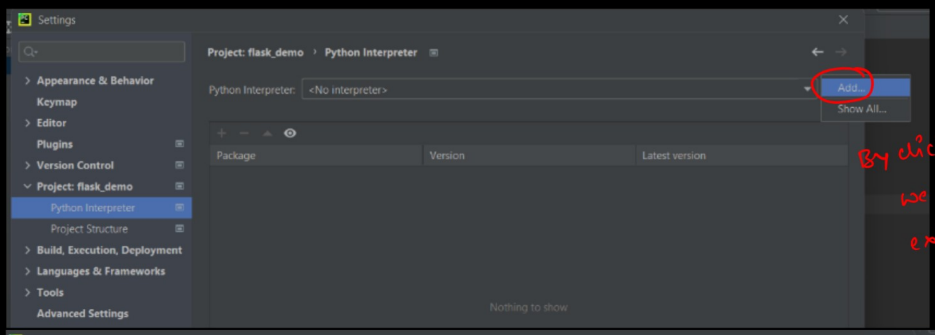
The main difference between PUT and PATCH in REST API is that **PUT handles updates by replacing the entire entity, while PATCH only updates the fields that you give it.** PATCH does not change any of the other values. If you use the PUT method, then everything will get updated. 14-Oct-2021

Flask Demo:

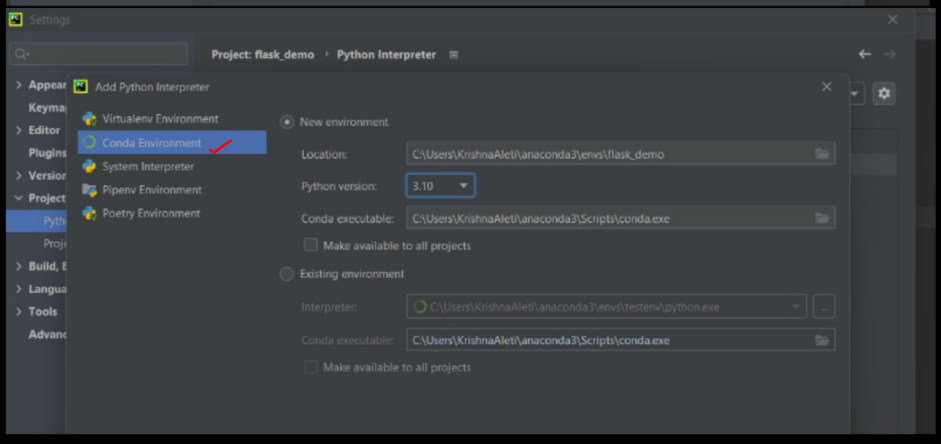
# Pycharm - Python Interpreter Setup:

File > Settings

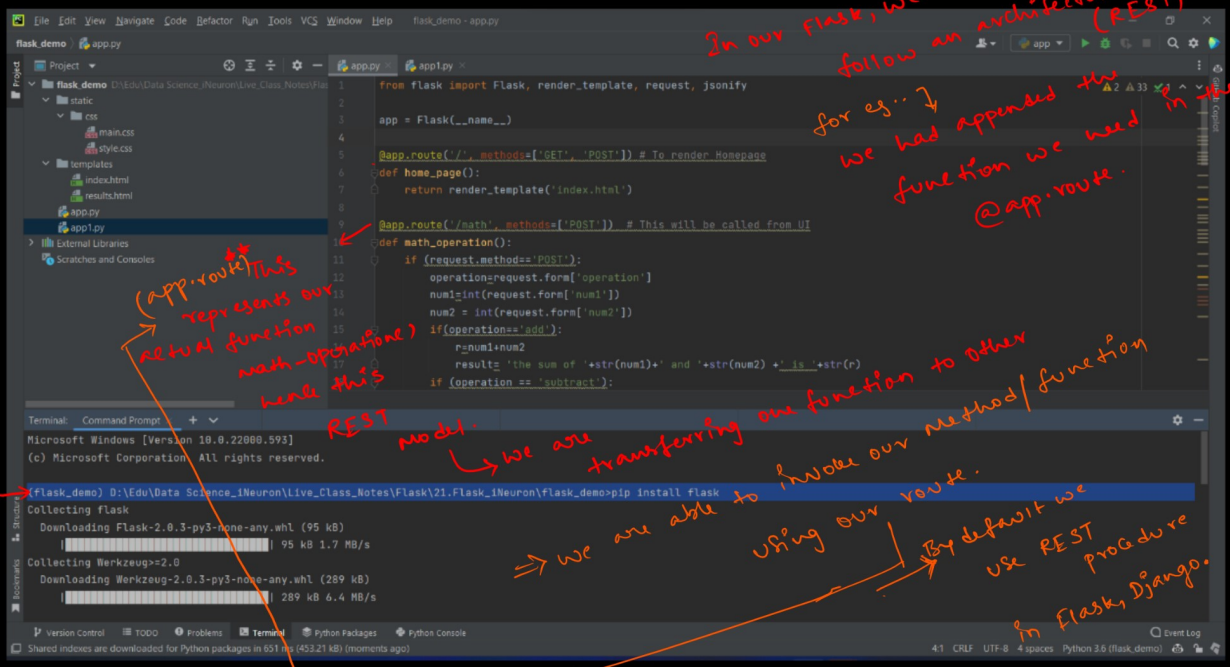


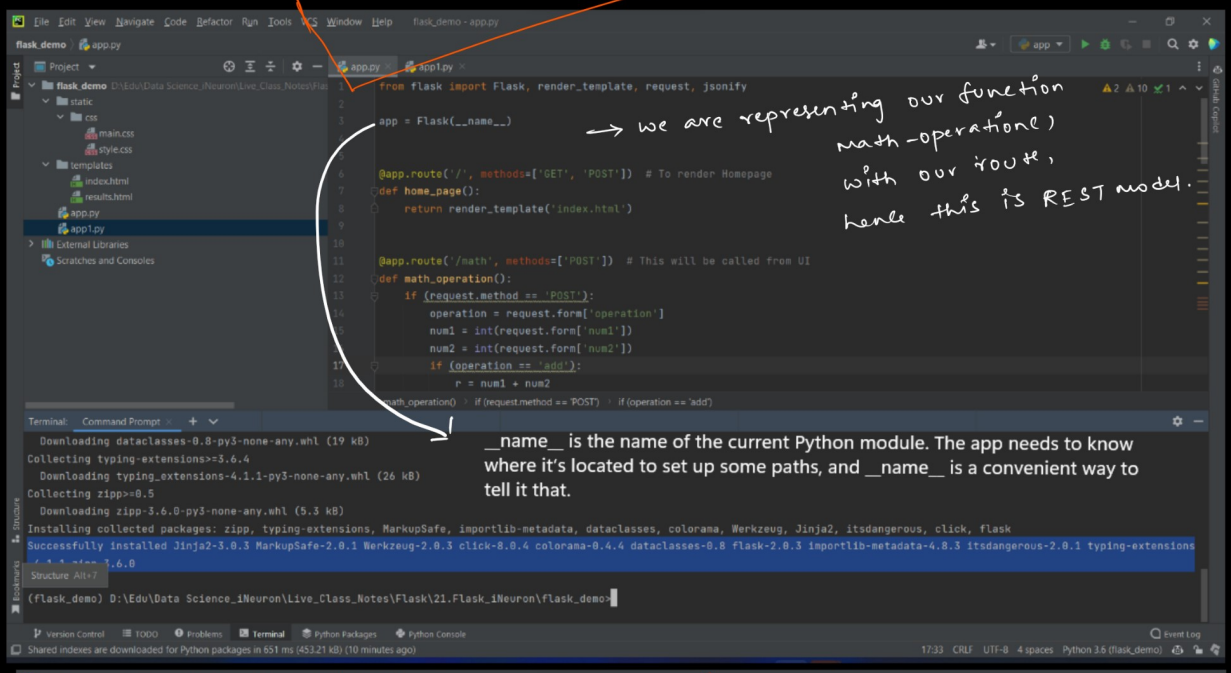



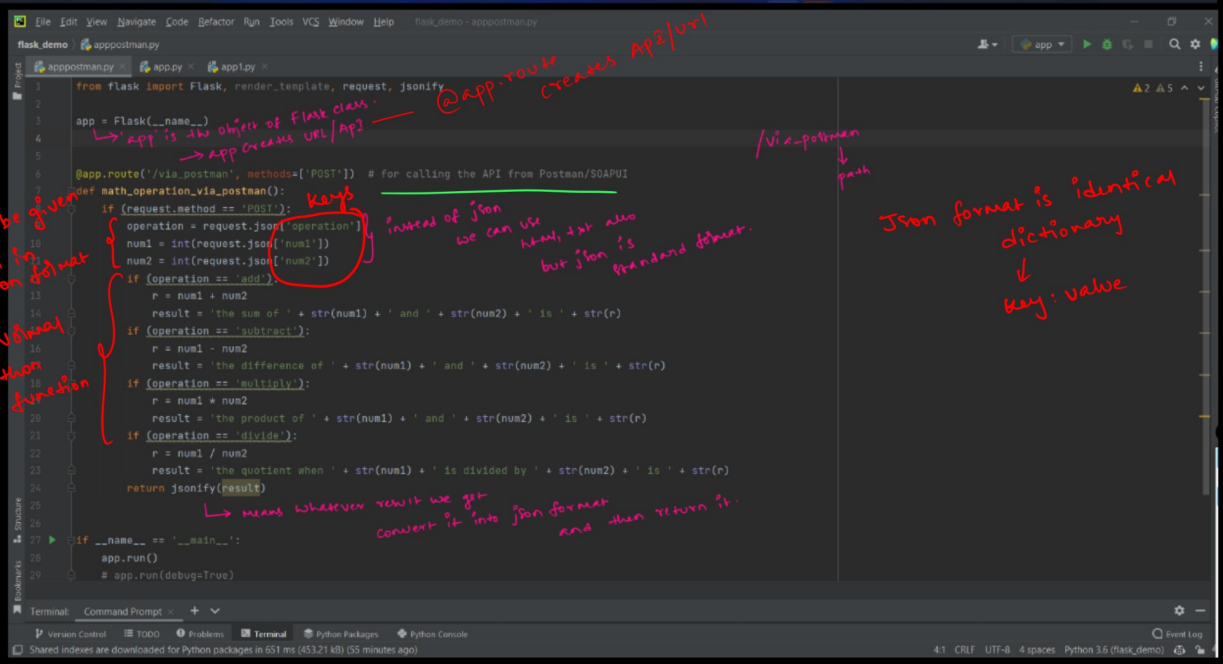

By clicking ShowAll, we can use the existing environments.



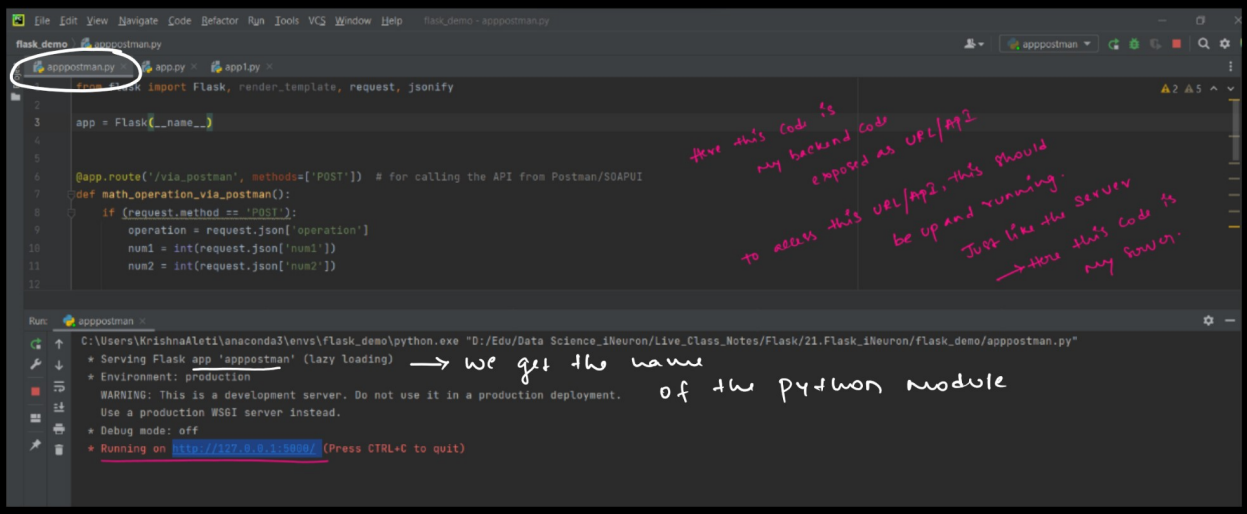

Ok > Apply > ok

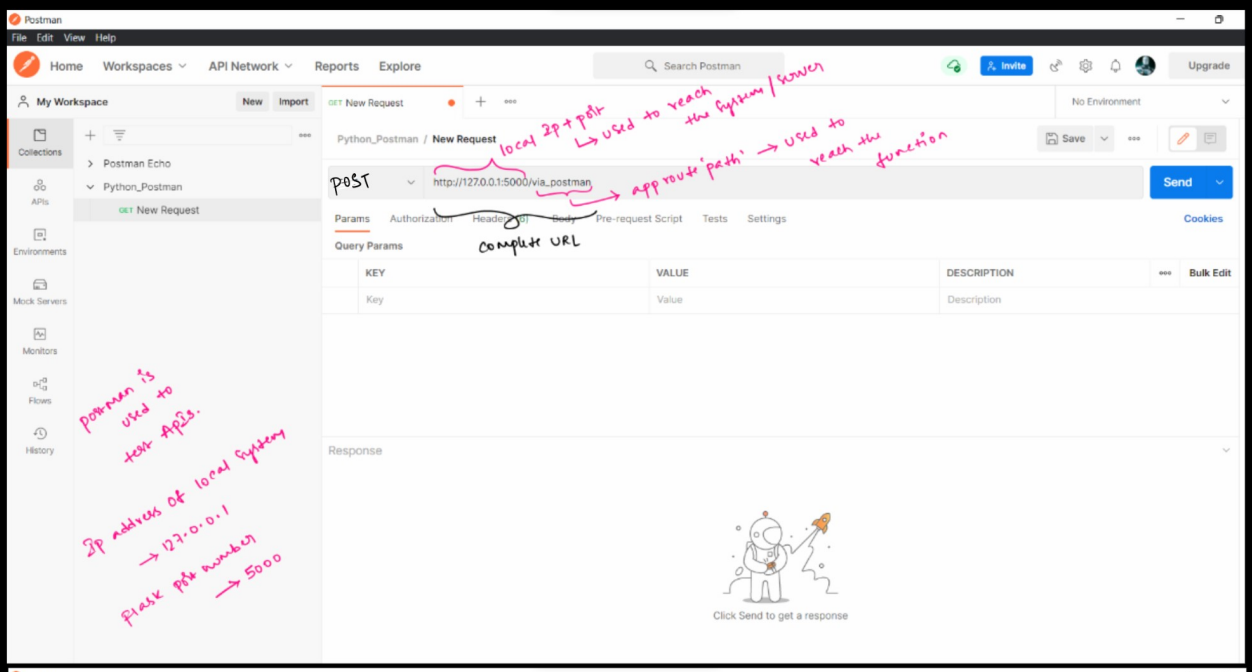

In our Flask, we follow an architecture (REST)
for eg.. }
we had appended the function we need in the @app.route.
(app.route) This represents our actual function (math_operations) here this
REST model.
we are transferring our function to other
we are able to invoke our method/function
using our route.
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
result= 'the sum of '+str(num1)+' and '+str(num2) +' is '+str(r)
if (operation == 'subtract'):
Microsoft Windows [Version 10.0.22000.593]
(c) Microsoft Corporation. All rights reserved.
(flask_demo) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Flask\21.Flask_iNeuron\flask_demo>pip install flask
Collecting flask
Downloading Flask-2.0.3-py3-none-any.whl (95 kB)
95 kB 1.7 MB/s
Collecting Werkzeug>=2.0
Downloading Werkzeug-2.0.3-py3-none-any.whl (289 kB)
289 kB 6.4 MB/s

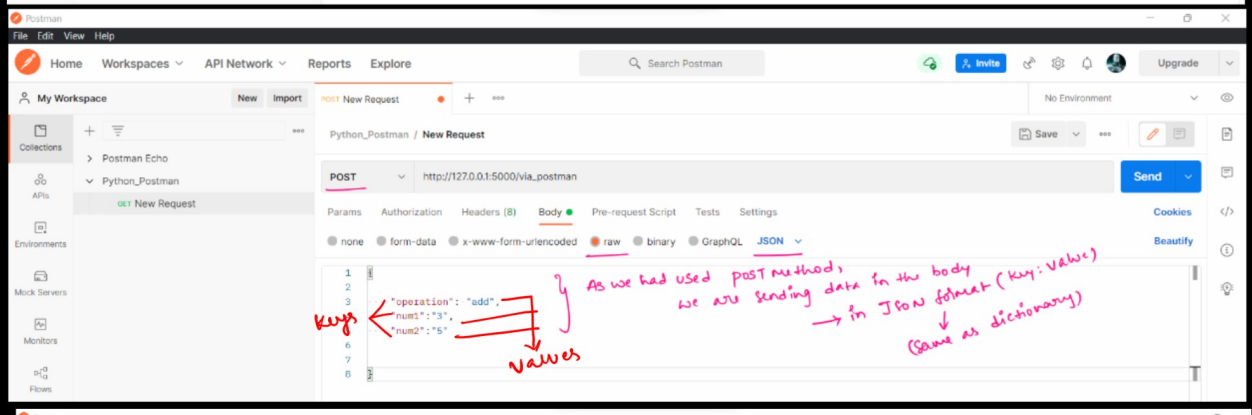

we are representing our function math_operations() with our route, hence this is REST model.
__name__ is the name of the current Python module. The app needs to know where it's located to set up some paths, and __name__ is a convenient way to tell it that.
Downloading dataclasses-0.8-py3-none-any.whl (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
(flask_demo) D:\Edu\Data Science_iNeuron\Live_Class_Notes\Flask\21.Flask_iNeuron\flask_demo>

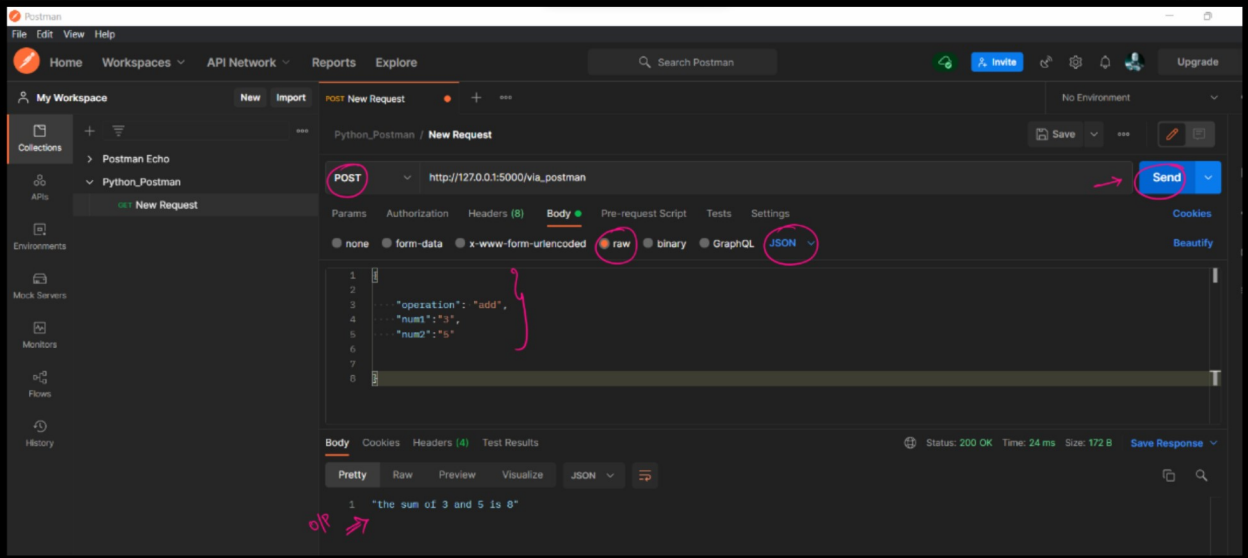

1
@app.route creates API/Url
from flask import Flask, render_template, request, jsonify
app = Flask(__name__)
'app' is the object of Flask class.
App creates URL/API
@app.route('/via_postman', methods=['POST']) # for calling the API from Postman/SOAPUI
/via-postman
path
def math_operation_via_postman():
Inputs to be given in Postman in Json format
keys
instead of json we can use html, it is also but json is standard format.
Json format is identical dictionary
key: value
if (request.method == 'POST'):
operation = request.json['operation']
num1 = int(request.json['num1'])
num2 = int(request.json['num2'])
Nothing Python function
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)
Means whatever result we get convert it into json format and then return it.
if __name__ == '__main__':
app.run()
# app.run(debug=True)

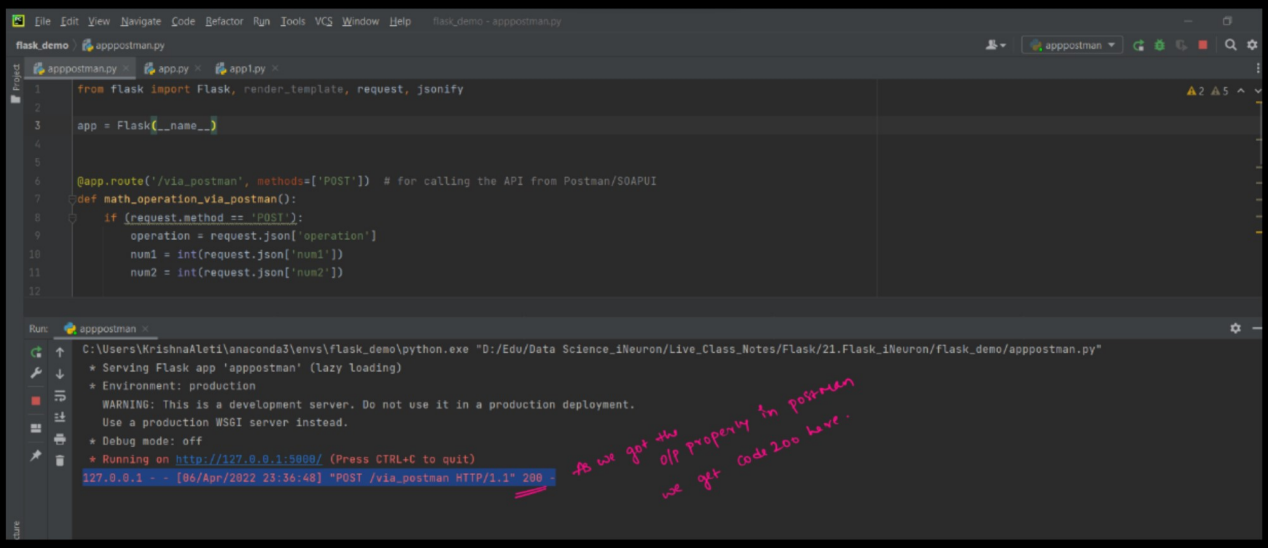
1

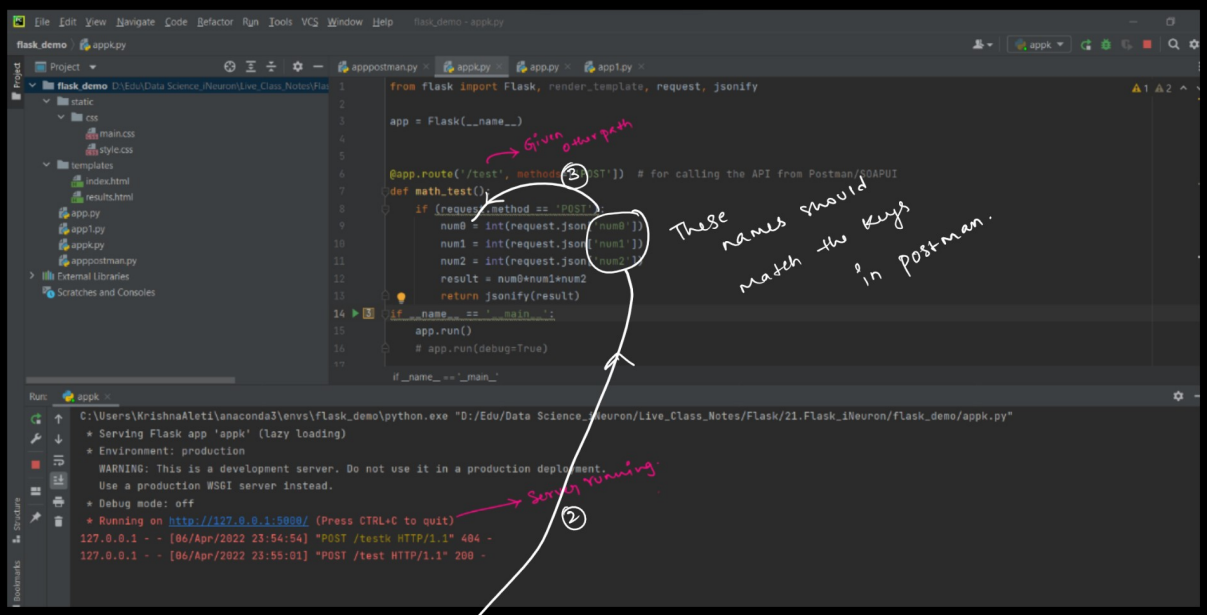

apppostman.py
app.py
app1.py
from flask import Flask, render_template, request, jsonify
app = Flask(__name__)
@app.route('/via_postman', methods=['POST'])  # for calling the API from Postman/SOAPUI
def math_operation_via_postman():
if (request.method == 'POST'):
operation = request.json['operation']
num1 = int(request.json['num1'])
num2 = int(request.json['num2'])
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.
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)
→ we get the name of the python module

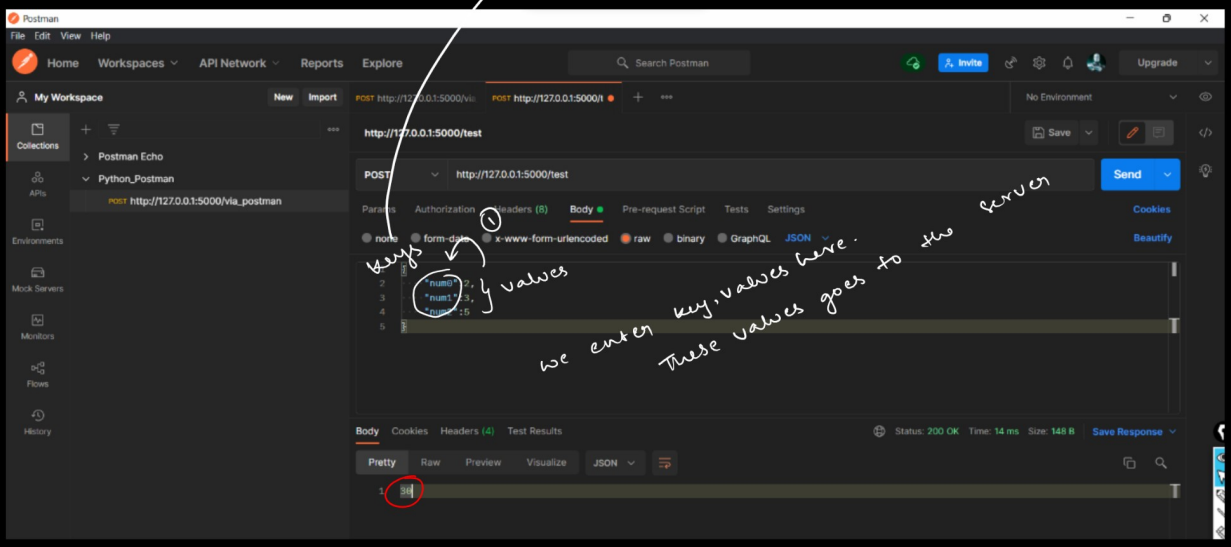

Postman
Home
Workspaces
API Network
Reports
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Postman Echo
Python_Postman
New Request
Python_Postman / New Request
Save
No Environment
POST
http://127.0.0.1:5000/via_postman
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Response
Click Send to get a response
local ip + port
→ used to reach the system/server
'app.route('path')' → used to reach the function
complete URL
postman is used to test APIs.
ip address of local system
→ 127.0.0.1
flask port number
→ 5000

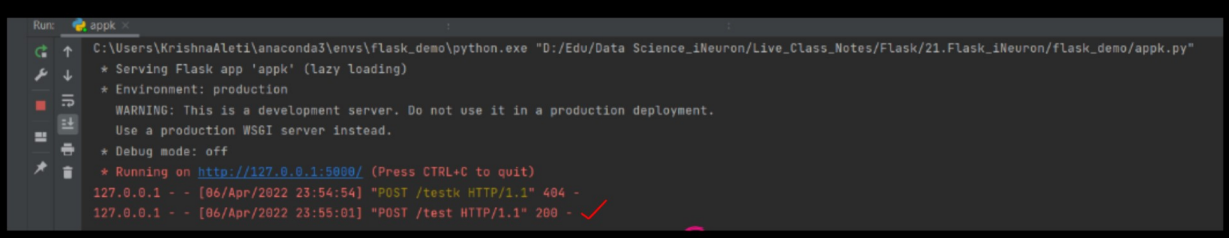

Postman
Home
Workspaces
API Network
Reports
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Postman Echo
Python_Postman
New Request
Python_Postman / New Request
Save
No Environment
POST
http://127.0.0.1:5000/via_postman
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"operation": "add",
"num1":"3",
"num2":"5"
}
keys
values
As we had used POST method, we are sending data in the body → in JSON format (key:value) (Same as dictionary)

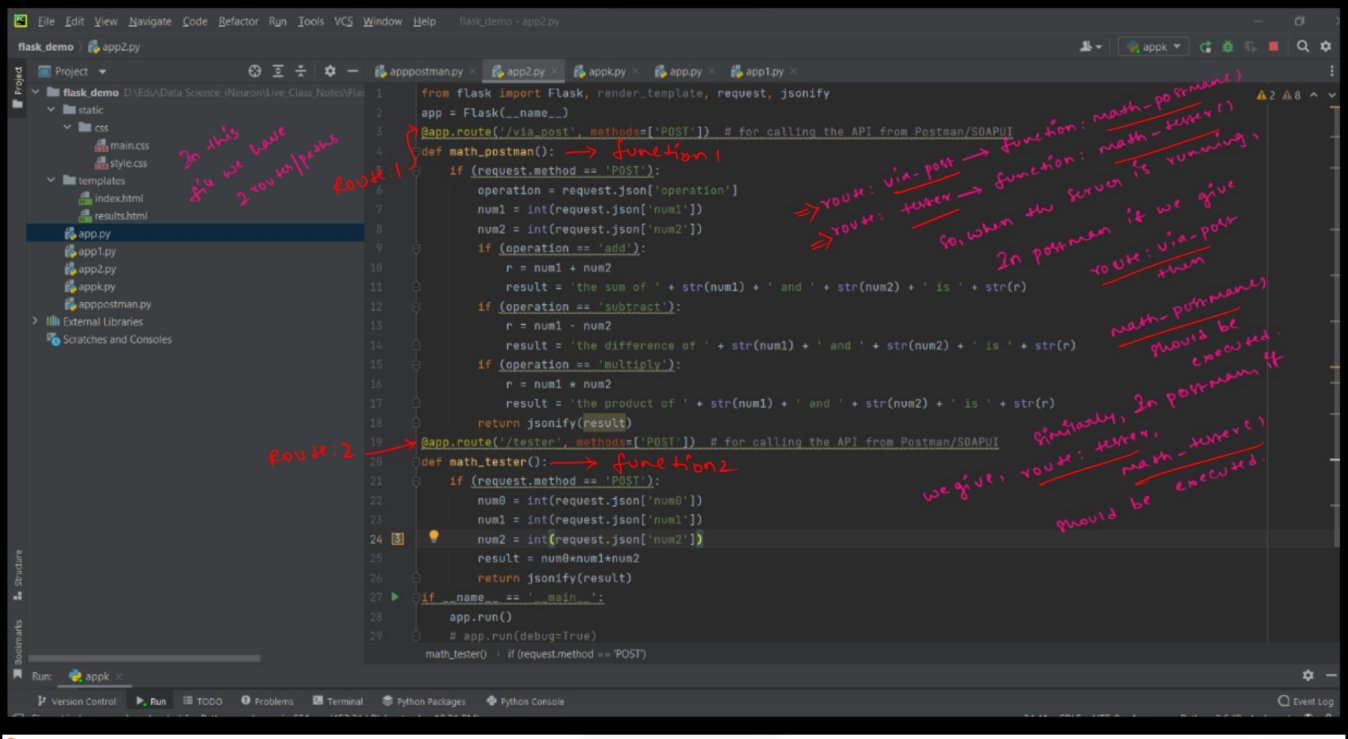

Postman
Home
Workspaces
API Network
Reports
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Postman Echo
Python_Postman
New Request
Python_Postman / New Request
Save
No Environment
POST
http://127.0.0.1:5000/via_postman
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"operation": "add",
"num1":"3",
"num2":"5"
}
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 24 ms
Size: 172 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
"the sum of 3 and 5 is 8"
o/p

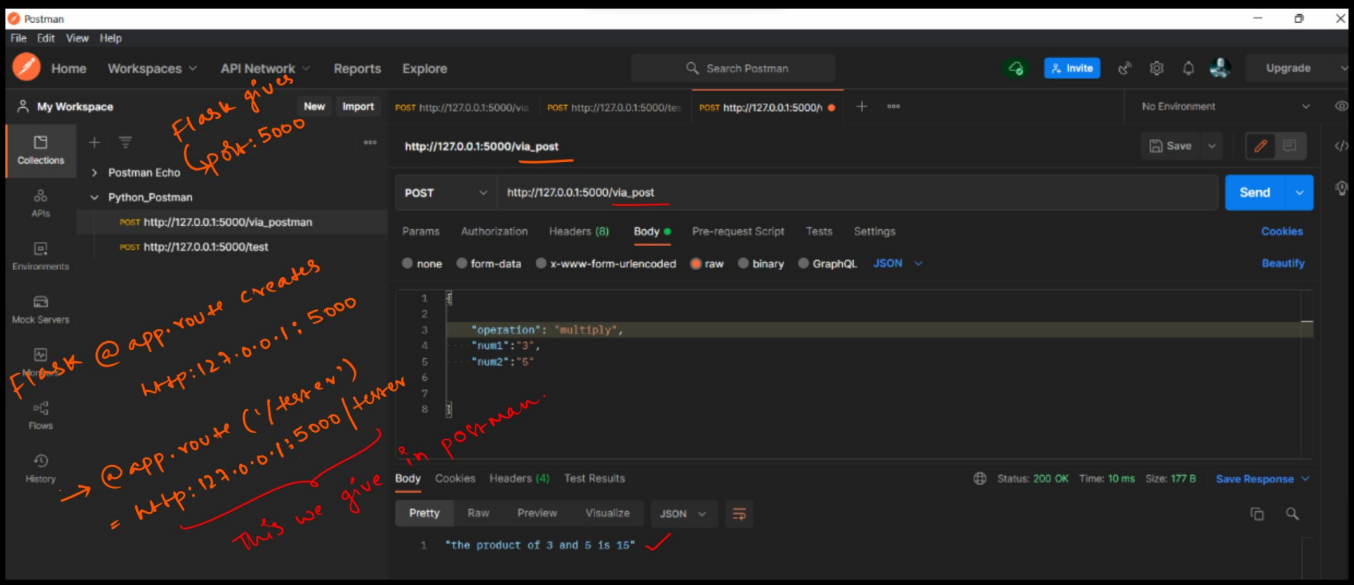

from flask import Flask, render_template, request, jsonify
app = Flask(__name__)
@app.route('/via_postman', methods=['POST'])  # for calling the API from Postman/SOAPUI
def math_operation_via_postman():
if (request.method == 'POST'):
operation = request.json['operation']
num1 = int(request.json['num1'])
num2 = int(request.json['num2'])
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 -
As we got the o/p properly in postman we get code 200 here.

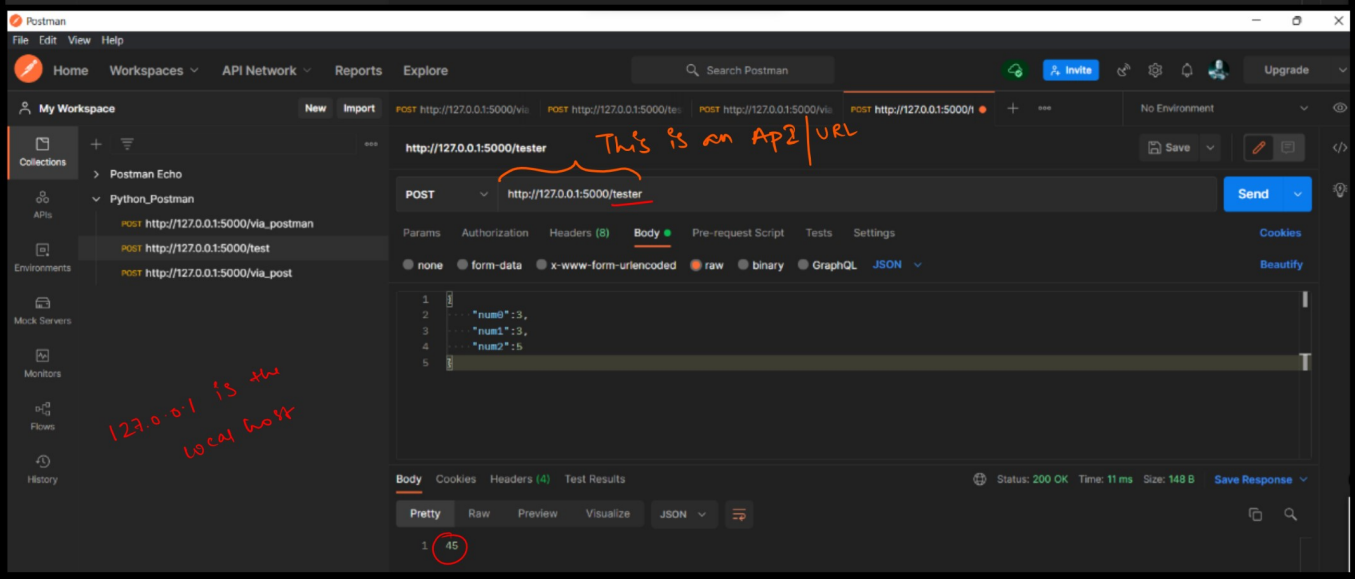

from flask import Flask, render_template, request, jsonify
app = Flask(__name__)
Given other path
@app.route('/test', methods=['POST'])  # for calling the API from Postman/SOAPUI
def math_test():
if (request.method == 'POST'):
num0 = int(request.json['num0'])
num1 = int(request.json['num1'])
num2 = int(request.json['num2'])
result = num0+num1+num2
return jsonify(result)
if __name__ == '__main__':
app.run()
# app.run(debug=True)
if __name__=='__main__'
These names should match the keys in postman.
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 -
Server running.
1
2
3

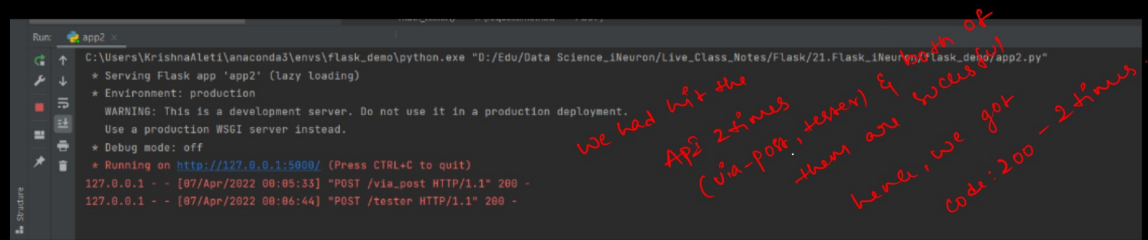

Postman
Home
Workspaces
API Network
Reports
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Postman Echo
Python_Postman
POST http://127.0.0.1:5000/via_postman
http://127.0.0.1:5000/test
No Environment
Save
POST
http://127.0.0.1:5000/test
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
"num0":2,
"num1":3,
"num2":5
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 14 ms
Size: 148 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
keys
values
we enter key, values here.
These values goes to the server
1
2

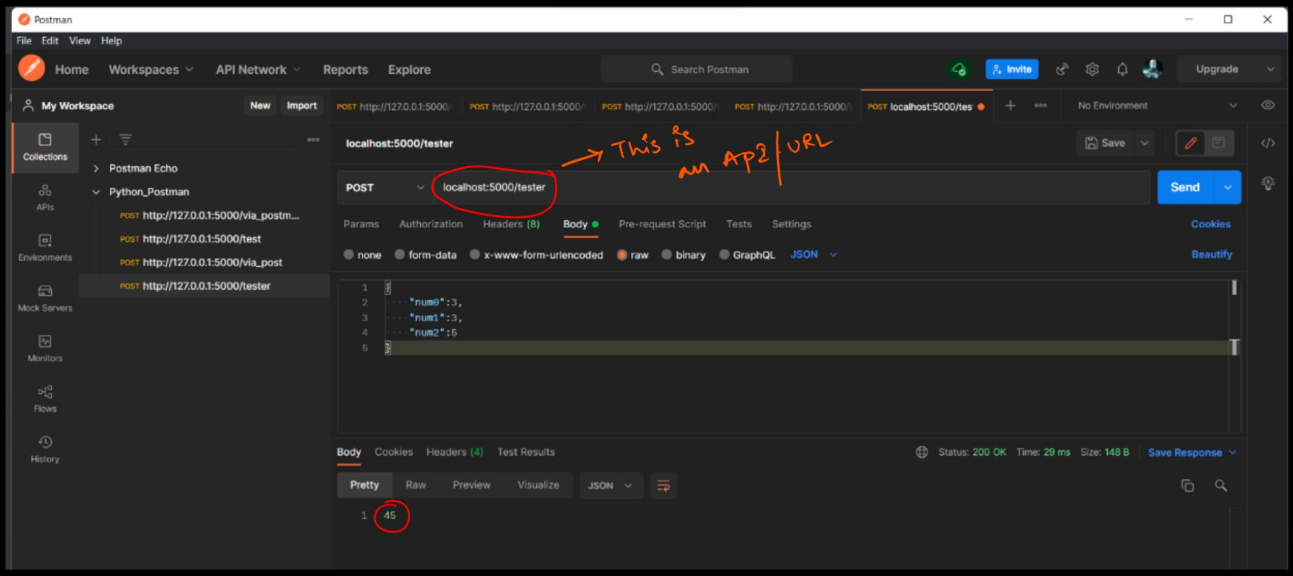

C:\Users\KrishnaAleti\anaconda3\envs\flask_demo\python.exe "D:/Edu/Data Science_iNeuron/Live_Class_Notes/Flask/21.Flask_iNeuron/flask_demo/appk.py"
* Serving Flask app 'appk' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use 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 -


File Edit View Navigate Code Refactor Run Tools VCS Window Help
flask_demo - app2.py
flask_demo > app2.py
Project
apppostman.py
app2.py
appk.py
app.py
app1.py
flask_demo D:\Edu\Data Science_iNeuron\Live_Class_Notes\Flas
static
css
main.css
style.css
templates
index.html
results.html
app.py
app1.py
app2.py
appk.py
apppostman.py
External Libraries
Scratches and Consoles
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)
@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_tester() › if (request.method == 'POST')
Run: appk
Version Control
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
In this file we have 2 routes/paths
Route 1
Route 2
function 1
function 2
=> route: via_post → function: math_postman()
=> route: tester → function: math_tester()
So, when the server is running,
In postman if we give route: via_post then math_postman() should be executed.
Similarly, In postman, if we give, route: tester, math_tester() should be executed.


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
My Workspace
New Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Postman Echo
Python_Postman
POST http://127.0.0.1:5000/via_postman
POST http://127.0.0.1:5000/test
http://127.0.0.1:5000/via_post
Save
POST
http://127.0.0.1:5000/via_post
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"operation": "multiply",
"num1":"3",
"num2":"5"
}
Body Cookies Headers (4) Test Results
Status: 200 OK Time: 10 ms Size: 177 B
Save Response
Pretty Raw Preview Visualize JSON
"the product of 3 and 5 is 15"
Flask gives (post: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.


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
My Workspace
New Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Postman Echo
Python_Postman
POST http://127.0.0.1:5000/via_postman
POST http://127.0.0.1:5000/test
POST http://127.0.0.1:5000/via_post
http://127.0.0.1:5000/tester
This is an API URL
Save
POST
http://127.0.0.1:5000/tester
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"num0":3,
"num1":3,
"num2":5
}
Body Cookies Headers (4) Test Results
Status: 200 OK Time: 11 ms Size: 148 B
Save Response
Pretty Raw Preview Visualize JSON
45
127.0.0.1 is the local host


Run: app2
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 00:05:33] "POST /via_post HTTP/1.1" 200 -
127.0.0.1 - - [07/Apr/2022 00:06:44] "POST /tester HTTP/1.1" 200 -
We had hit the API 2 times (via_post, tester) & both of them are successful kernel, we got code: 200 - 2 times




⇒ 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

This helps us reach the particular function & execute.

This helps to reach the particular system using the Ipaddress (http://127.0.0.1 or 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.

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.

→ Flask - Micro web Framework ⇒ Flask is used to create RESTful APIs.

↳ Is a REST FUl API web Framework.

↳ Pythonic Framework - we can use Flask with Python Code only.

↓ Uses REST

↓ Easy to use/install

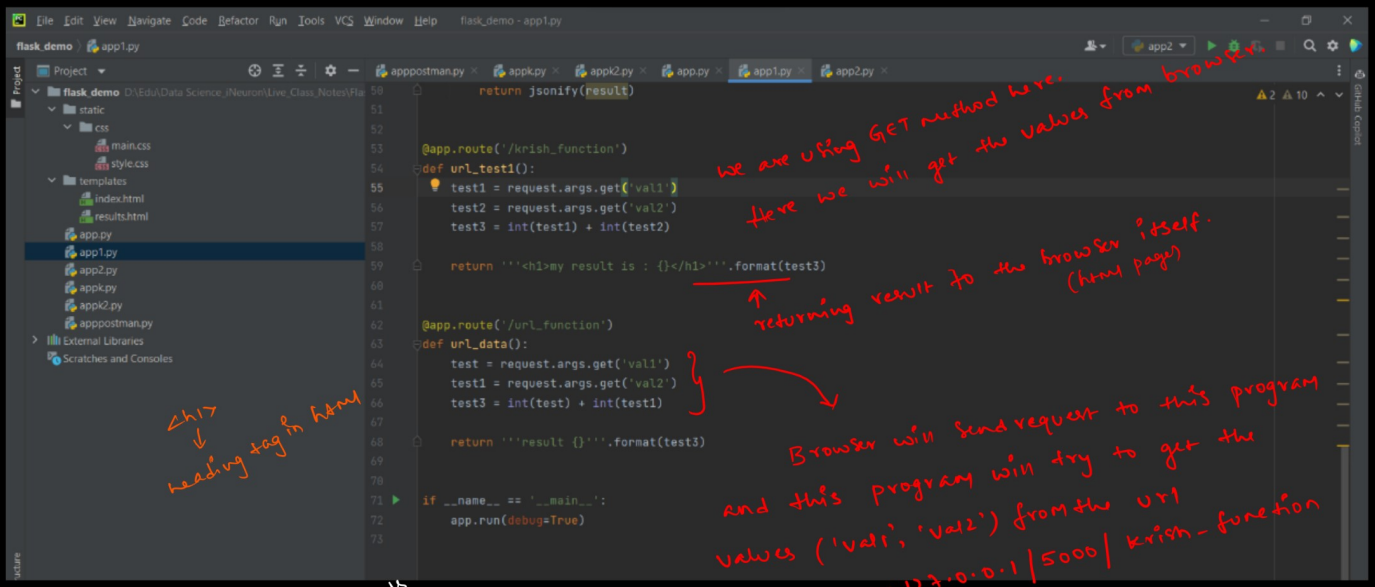
→ API is created by Flask, so API is a Microservice.

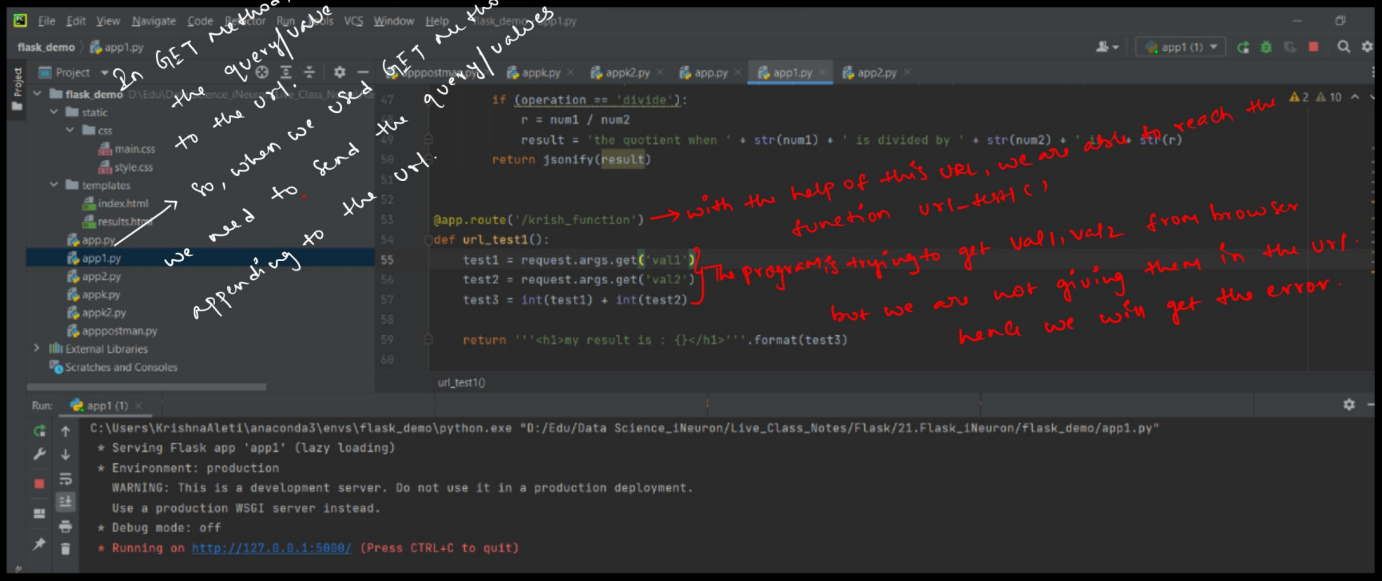
↓ uses REST, hence REST API

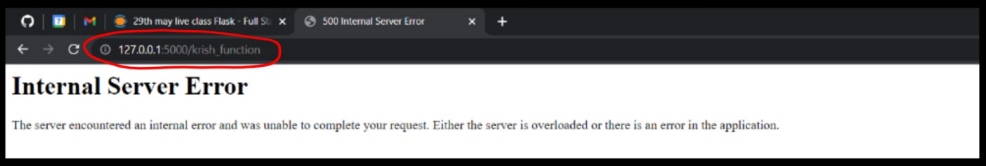
↓ Can integrate in web, hence it is web Framework.

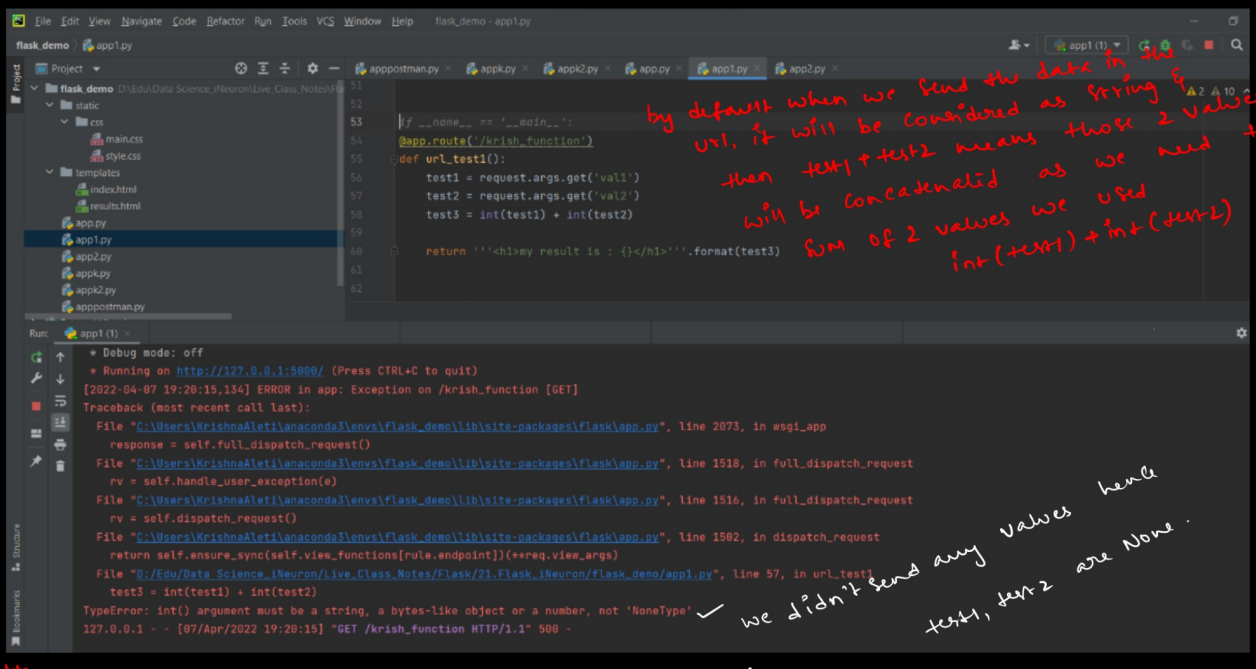
Flask is a pythonic Framework - Flask can be used with Python only. but the API that we had created using Flask can be exposed to other programs for usage. For eg:- A program written in JAVA can consume the API created by Flask (python code).

# Flask

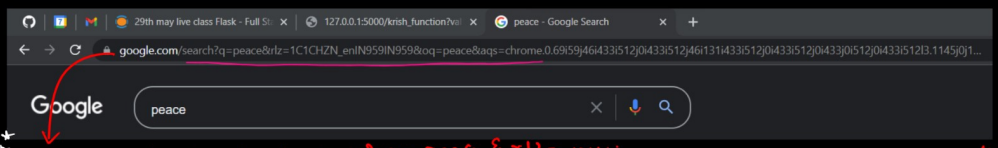


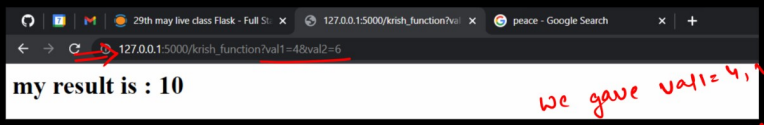

→ To resolve this issue, we need to send data with the url.




⇒ Notation for search → url ? q = peace & rlz = .....

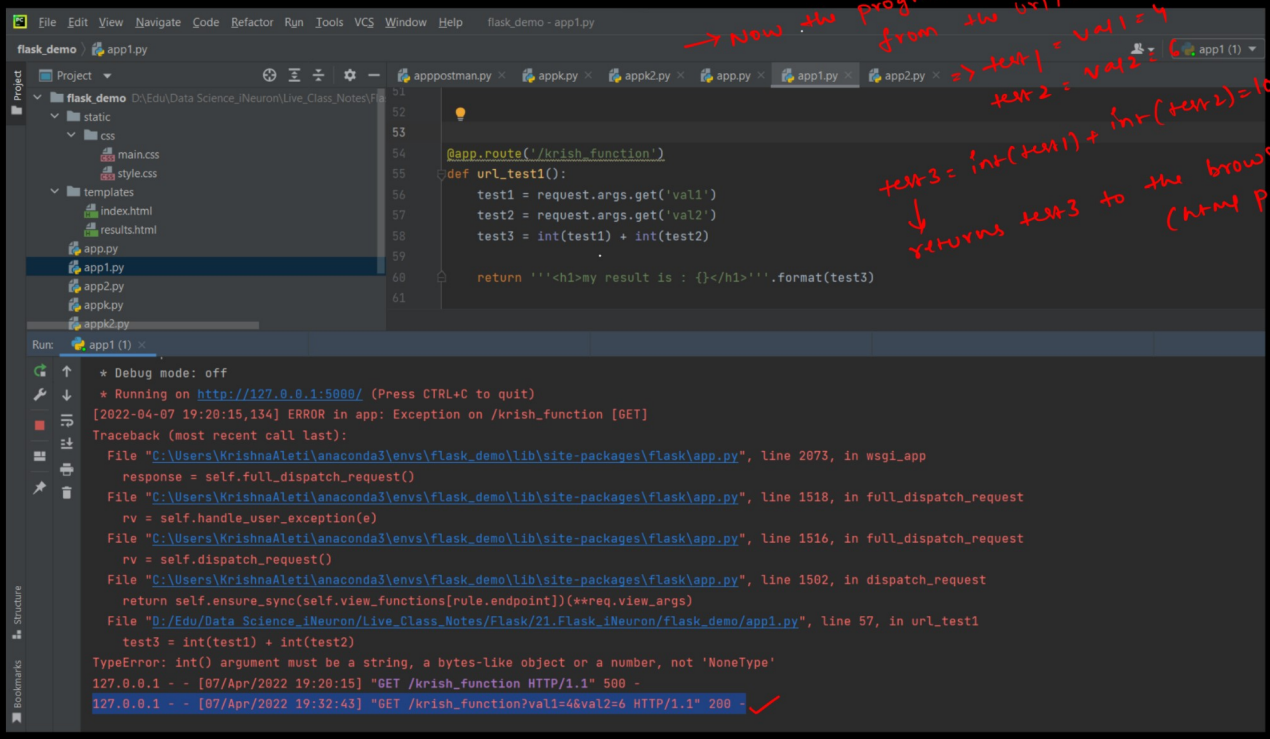
Similarly now we want to pass data in the browser → http://127.0.0.1:5000/krish_function?val1=4&val2=6
(required for programs)

⇒ once 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).

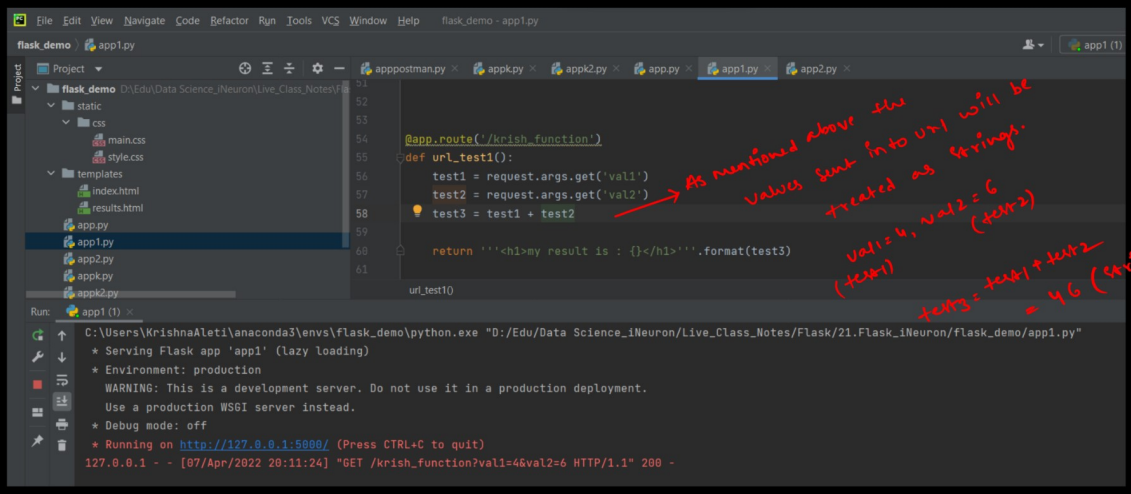



we gave val1=4, val2=6 in the url.

→ Now the program will get those values from the url / browser.



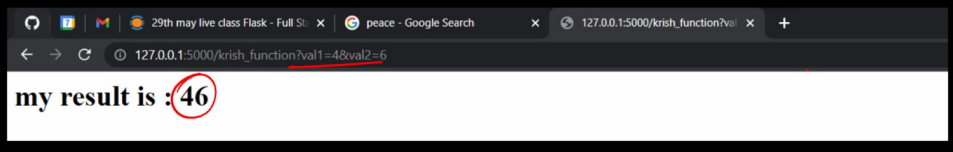

test1 = val1 = 4
test2 = val2 = 6
test3 = int(test1) + int(test2) = 10
↓
returns test3 to the browser (htmp page)

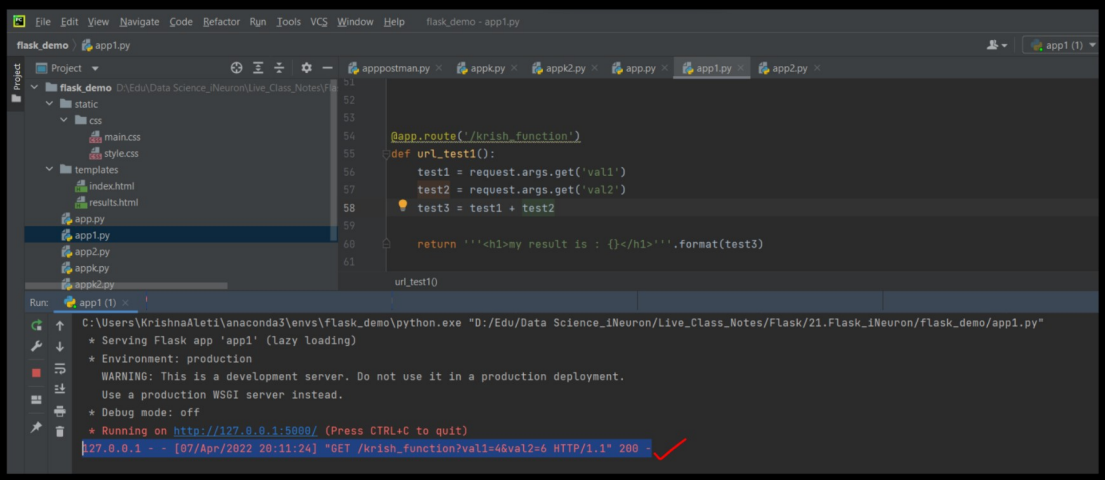



As mentioned above the values sent into url will be treated as strings.

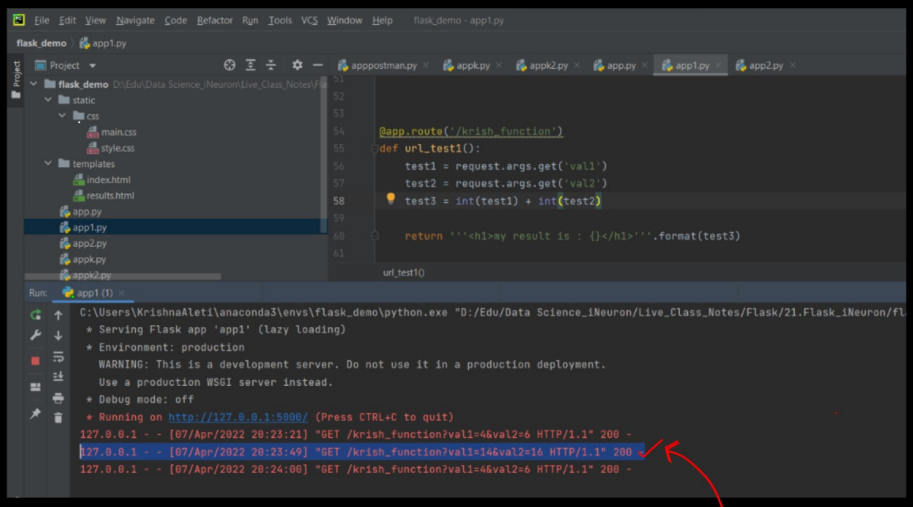
val1 = 4, val2 = 6
(test2)
(test1)

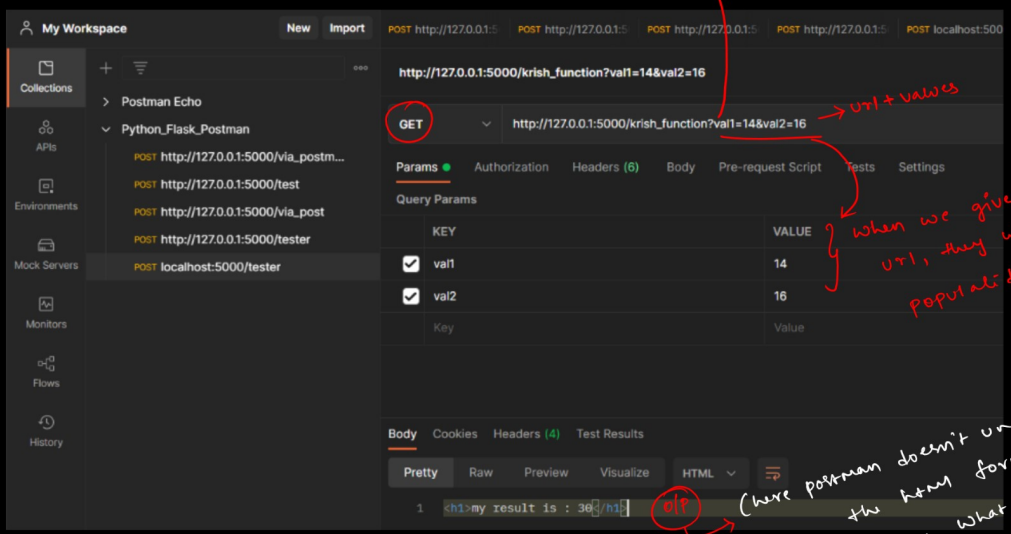
test3 = test1 + test2
= 46 (strings concatenated)






Let us try the same from the postman:

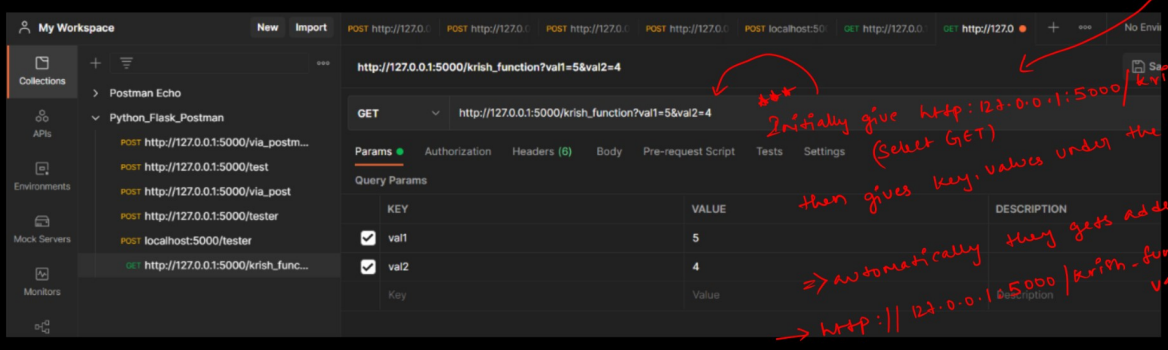



Url + values

When we gives 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).




Initially give http://127.0.0.1:5000/krish_function (Select GET)

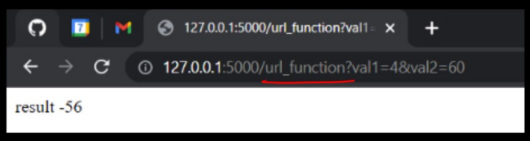
then gives key, values under the params

=> automatically they gets added to url

→ http://127.0.0.1:5000/krish_function?val1=5&val2=4

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

app = Flask(__name__)

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

    return '''<h3>my result is : {}</h3>'''.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)
    app.run()
```

2 routes - 2 functions


can use any of the methods GET/POST

```python
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('/mathops', 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)
if __name__ == '__main__':
    # app.run(debug=True)
    app.run()
```

```
index.html
{% block head %}
<title>MathOperations</title>
<link rel="stylesheet" href="{{ url_for('static',
{% endblock %}
{% block body %}
<div class="content">
    <h1 style="text-align: center">Calculator</h1>
    <div class="form">
        <form action="/mathops" method="POST">
```

@app.route('/') means the root url http://127.0.0.1:5000/ when I 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

whatever mentioned in index.html will be displayed in the homepage.

In the homepage, we have submit/calculate button when it is hit, the path routes to '/mathops' & the function in it is executed.

because in index.html we had defined that when calculate click action='/mathops'

need to give templates

we can also change the name but then we need to given full path instead of just index.html

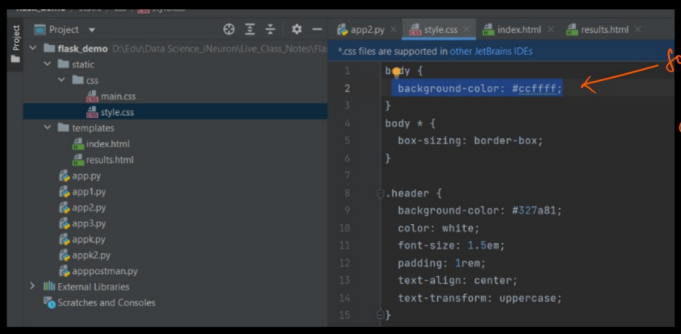
(html behaves like the interpreter, it doesn't execute the entire code, it will just parse & whatever we had written it will populate all and will display)

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

→ static > css > main.css, style.css ( css - cascading style sheet)

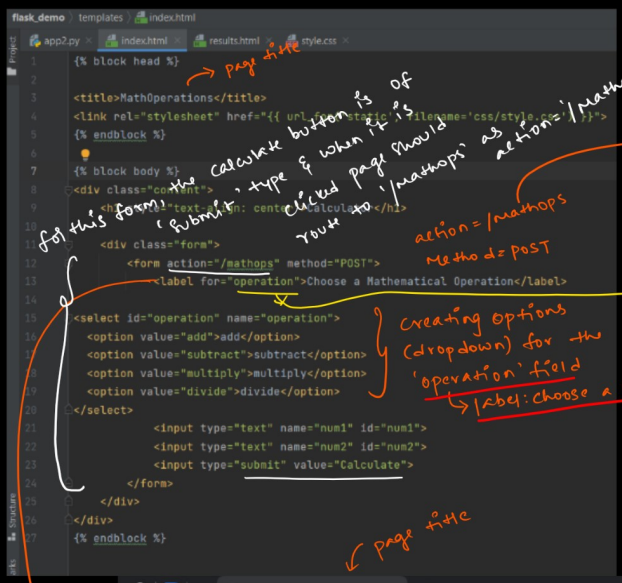
⇒ render_template('index.html')

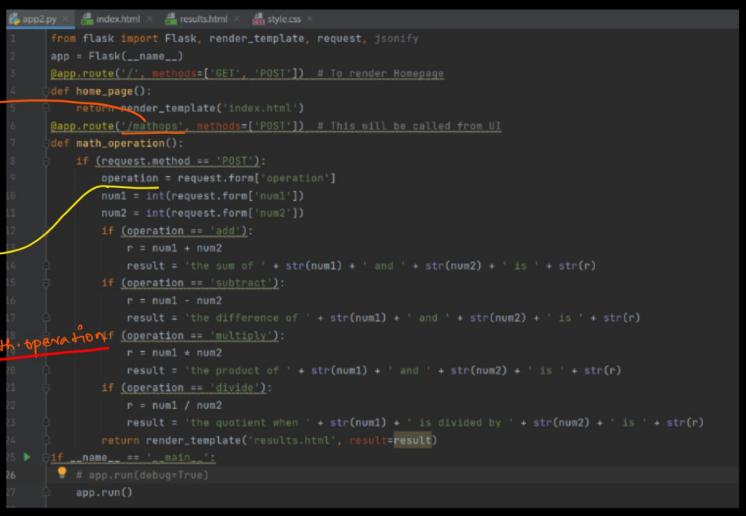
↳ It just renders the html file - shows whatever is there in html file.

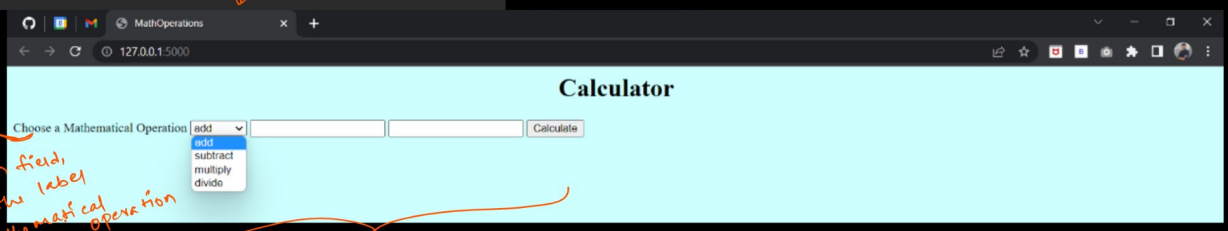



for the background color of the html page.

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

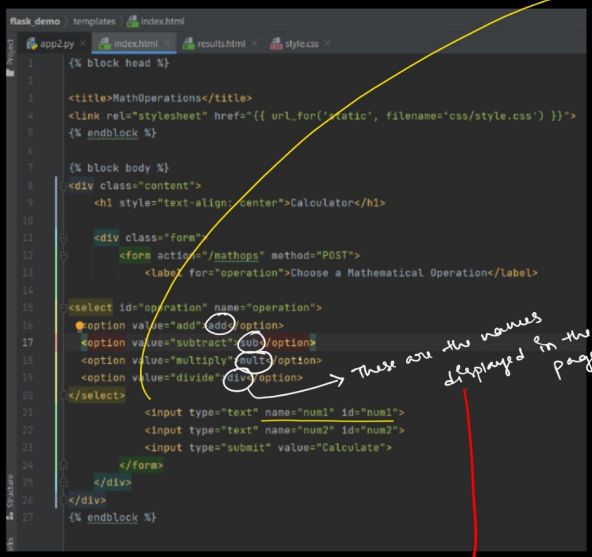


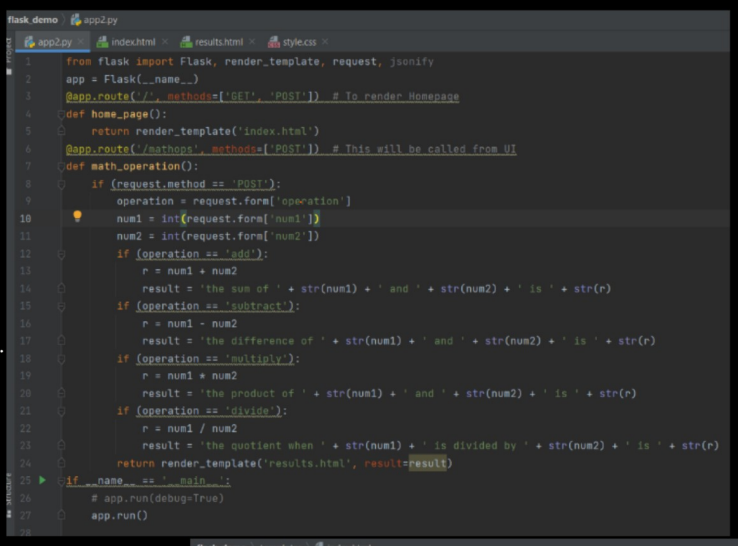



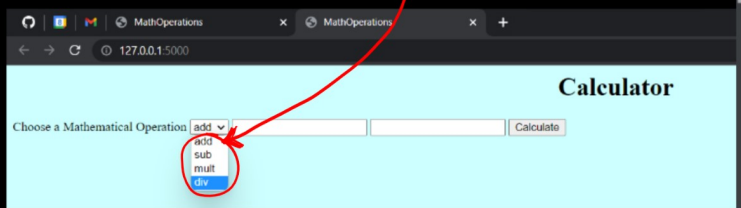



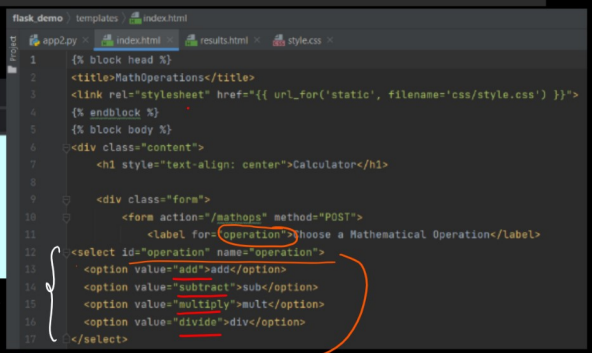

for operation field, we gave the label choose a Mathematical operation
It is having a dropdown & the value that we select from dropdown is the value for 'Operation'
This is a form

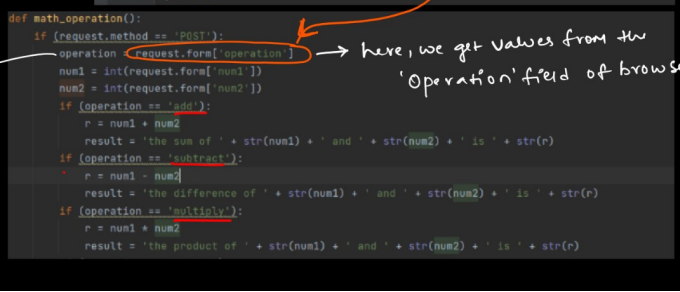
- id is used to identify the HTML element through the Document Object Model (via JavaScript or styled with CSS). id is expected to be unique within the page.
- name corresponds to the form element and identifies what is posted back to the server.




for the field 'operation', we gave the label: Choose a Math operati 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.

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

```html
{% block head %}
<title>MathOperations</title>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
{% endblock %}
{% block body %}
<div class="content">
    <h1 style="text-align: center">Calculator</h1>

    <div class="form">
        <form action="/mathops" method="POST">
            <label for="operation">Choose a Mathematical Operation</label>
<select id="operation" name="operation">
  <option value="add">add</option>
  <option value="subtract">sub</option>
  <option value="multiply">mult</option>
  <option value="divide">div</option>
</select>
            <input type="text" name="num1" id="num1">
            <input type="text" name="num2" id="num2">
            <input type="submit" value="Calculate">
        </form>
    </div>
</div>
{% endblock %}
```

Here the 'calculate' button is to submit the values entered.

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

```python
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('/mathops', 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)
if __name__ == '__main__':
    # app.run(debug=True)
    app.run()
```

we get these values from the browser.

127.0.0.1:5000

**Calculator**

Choose a Mathematical Operation [add] [ ] [ ] Calculate

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)

*** 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 drop down, enter the num1, num2 values & click calculate.

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

⇒ the function defined in the '/mathops' path => Math-Operations() 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

① As per the operation, we get the result.

② that result is stored in the 'result' variable.

(variable name)

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

After execution, the server returns the results.html page having the result.


```html
<!DOCTYPE html>
<html lang="en" >

<head>
  <meta charset="UTF-8">
  <title>Result Page</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/5.0.0/normalize.min.css">
    <link rel="stylesheet" href="./style.css">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
  <div class="table-users">
    <div class="header">Calculation Result</div>

    {{result}}
</div>
</body>
</html>
```

page title (for page routing)

result variable

browser routes to '/mathops' → performs operation & the result will be displayed with the help of result.html in the browser

page title

MathOperations | Result Page

127.0.0.1:5000/mathops

CALCULATION RESULT (header)

the sum of 2 and 23 is 25

Imp:

```python
"""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

```python
"""from the browser and also from postman"""
@app.route('/krish_function')
def url_test1():
    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=46val2=6
```

Method = GET

← The data will be appended to the url

```python
"""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.