Article
Muhammad Waseem · Sep 20 8m read

Getting to know Python Flask Web Framework

Hi Community,

In this article, I will introduce Python Flask Web Framework. Together we will create a minimal web application to connect to IRIS and get data from it.

Below you can find the steps we will need to follow:

- **Step 1 : Introduction to Python Flask Web Framework**
- **Step 2 : Installation of Flask module**
- **Step 3 : Creation of web application using Flask**
- **Step 4 : Use of HTML Templates**
- **Step 5 : Installation of IRIS Python Native module**
- **Step 6 : Establishment of a connection with IRIS**
- **Step 7 : Transferring data from IRIS to Flask and displaying it**

So Let's start with step 1

Step1-Introduction to Python Flask Web Framework

Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since it allows to build a web application quickly using only a single Python file. Flask is also extensible and doesn ' t requires a particular directory structure or complicated boilerplate code before getting started.

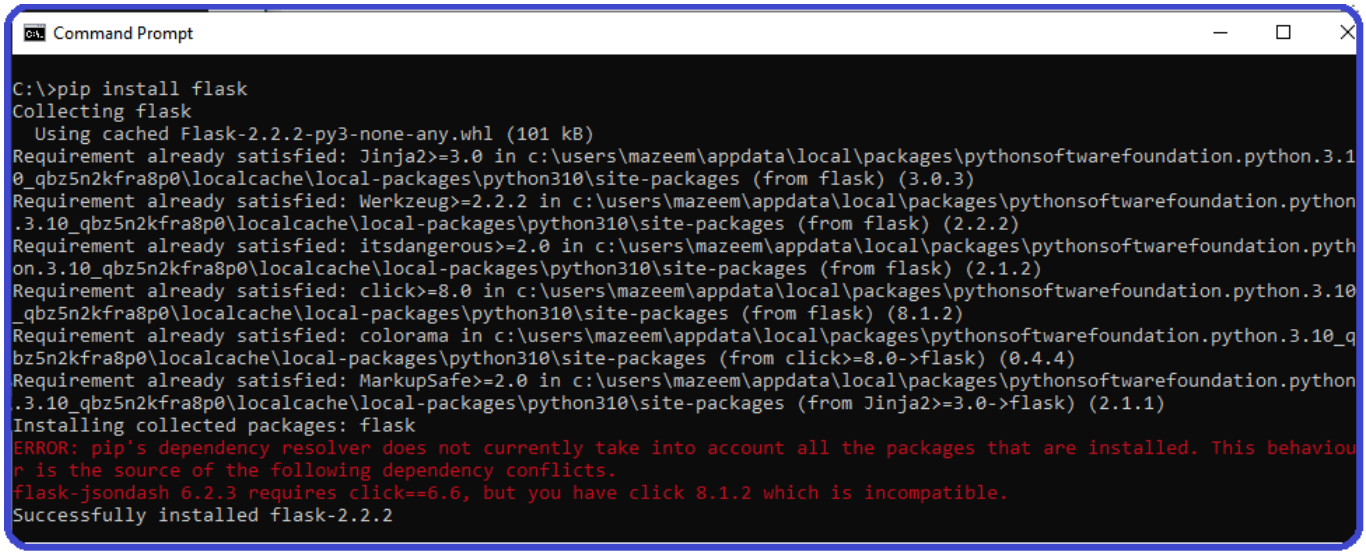
For more details please view Flask Documentations

Step 2 : Installation of Flask module

Before we start building our Flask Web Application, we need to install the Flask module using the pip package installer.

To install Flask, run the following command:

pip install flask



Flask latest version is install successfully.

Step3 : Creation of web application using Flask

Now that we have installed the flask module, we will make a minimal web application inside a Python file and run it. This will start the server, which will display some information on the browser.

Create a folder and create a python file inside. Let's name our folder flask2022 and the python file - app.py

Below you can see an app.py file which will serve as a minimal example of how to handle HTTP requests

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, This is Flask Application!</p>"
if __name__ == '__main__':
    app.run(debug=True)
```

Code Explained:

```
#First we import the Flask class. An instance of this class will be our WSGI applicat
ion
from flask import Flask

#Next we create an instance of this class. The first argument is the name of the appl
ication module or package. __name__ is a convenient shortcut for this which is approp
riate for most cases. This is needed so Flask to know where to look for resources suc
h as templates and static files.
app = Flask(__name__)

#Once you have created the app instance, you will be able to use it to handle incomin
g web requests and send responses to the user. @app.route is a decorator that turns a
 regular Python function into a Flask view function, which converts the function retu
rn value into an HTTP response to be displayed by an HTTP client, such as a web brows
er. You pass the value '/' to @app.route() to signify that this function will respond
 to web requests for the URL /, which is the main URL
@app.route("/")
#The function returns the message we want to display in the user’s browser. The defau
lt content type is HTML. That is why  HTML in the string will be rendered by the brow
ser. We can define any unique function name as
def hello_world():
    return "<p>Hello, World!</p>"
#Now start server and Run the application with debug = True to display errors in brow
ser if any
if __name__ == '__main__':
    app.run(debug=True)
```

From the app.py file location, run the command mentioned below to start the application at the command prompt:

```
python app.py
```

```
D:\flask2022>python app.py
 * 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: 226-609-762
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

The preceding output has several pieces of information, such as:

- **The name of the application you ' re running.**
- **The environment in which the application is being run.**
- **Debug mode: on, which signifies that the Flask debugger is running. This is useful when developing because it gives us detailed error messages when things go wrong, which makes troubleshooting easier.**
- **The application is running locally on the URL http://127.0.0.1:5000/, 127.0.0.1 is the IP that represents your machine ' s localhost and :5000 is the port number.**

Open a browser and type in the URL http://127.0.0.1:5000/, you will receive the string Hello, World! as a response. This confirms that your application is running successfully.

Congratulation our first flask application is running

Step 4 : Use of HTML Templates

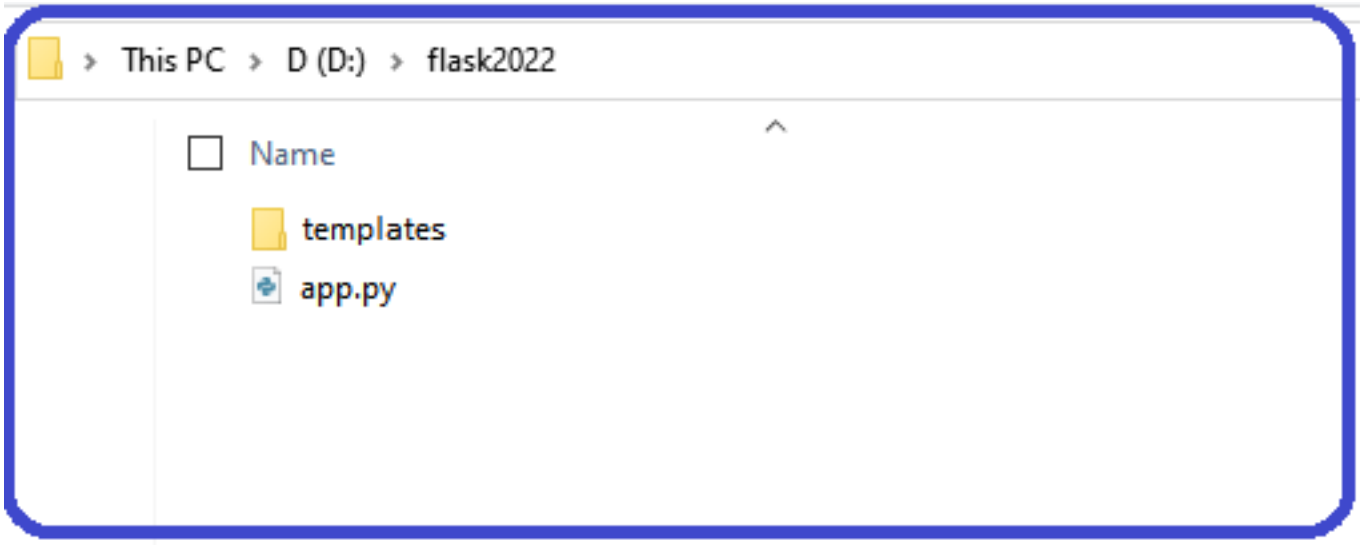
Currently our application only displays a simple message without any HTML. Web applications mainly use HTML to display information for the visitor, so we will now work on incorporating HTML files in our app to display it on the web browser.

Flask provides a rendertemplate() helper function that allows us to use the Jinja template engine. It makes managing HTML much easier by writing your HTML code in .html files and using logic in your HTML code.

Let's create "templates" folder in our flask2022 folder. Please note that folder name must be "templates", and it

should have the same location where our "app.py" file is located.

Below you can see the directory structure once we have created our templates folder:



Let us create an index.htm file in the templates folder. Check the example below.

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

After that, let us modify the app.py file to render the template instead of displaying simple text.

```
from flask import Flask,render_template

app = Flask(__name__)

@app.route("/")
def hello_world():
    return render_template('index.html')

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

Please note two modification in above-mentioned code. The The first one is that we are importing rendertemplate from the flask module. The second one is that we are returning the index.html page in the helloworld function which we saved in the templates folder by using rendertemplate() function.

Let us run the application and navigate to http://127.0.0.1:5000

This time we are getting HTML which we saved in templates folder.

Step 5 : Installation of IRIS Python Native module

Now in order to communicate with IRIS, we need to install IRIS python wheel file. For more details please read Native SDK for Python Quick Reference

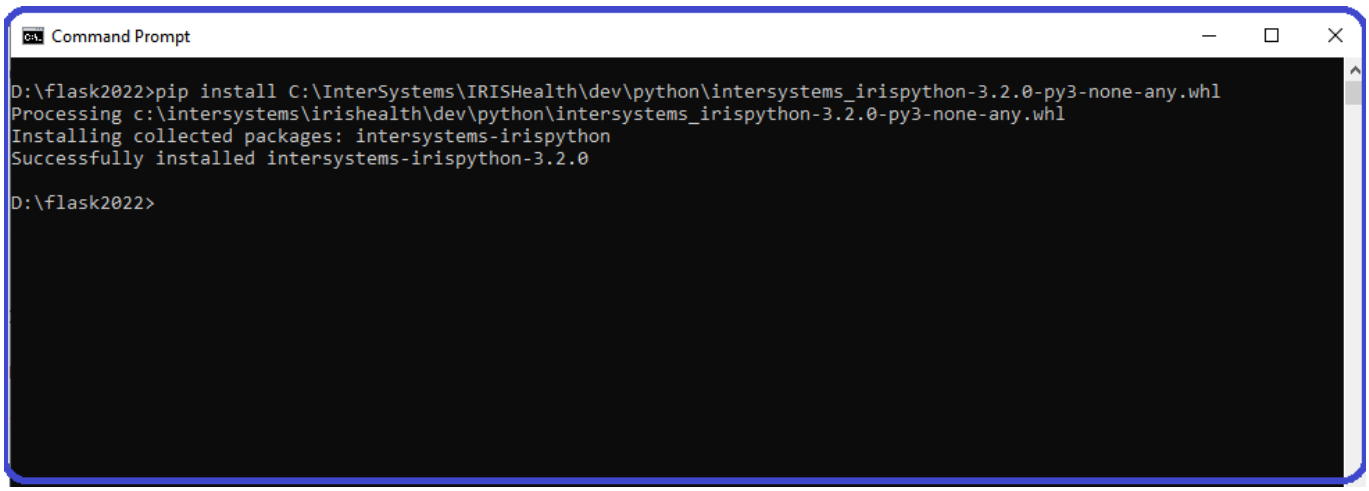
Depending on the operating system and installation path, the WHL is located at <InterSystems installation directory> \dev \python folder

Since I am using windows, and my installation directory is c: \InterSystems, so the whl file path will be:

C: \InterSystems \IRISHealth \dev \python \intersystemsirispython-3.2.0-py3-none-any.whl

Let's install the IRIS python native module by using below command quoted below:

```
pip install C:\InterSystems\IRISHealth\dev\python\intersystems_irispython-3.2.0-py3-none-any.whl
```



Now the IRIS python native module has been installed successfully

Step 6 : Establishment of a connection

First, we need to import irisnative module in python by using below command

```
import irisnative
```

then we will use irisnative module createConnection() function by passing following parameters

```
# Create connection to InterSystems IRIS
connection = irisnative.createConnection(ip, port, namespace, username, password)
```

After that, we will create an IRIS object by using irisnative module createIris() function by using below command

```
# Create an iris object
iris_native = irisnative.createIris(connection)
```

For more details please read Python Native API documentation

Below you can find the code of the python file:

```
from flask import Flask,render_template
import irisnative

app = Flask(__name__)

### Native API connection's parameters
ip = "localhost"
port = 1972
namespace = "USER"
username = "_SYSTEM"
password = "SYS"
###Create database connection and IRIS instance
try:
    #Return a new open connection to an IRIS instance.
    connection = irisnative.createConnection(ip, port, namespace, username, password)
    #Return a new irisnative.iris object that uses the given connection.
    myIris = irisnative.createIris(connection)
except Exception as e:
    print(e)


@app.route("/")
def hello_world():
    return render_template('index.html')

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

Step 7 : Transferring data from IRIS to Flask and displaying it

In this final step we will:
(1) get data from Global
(2) pass the data to index.html
(3) display the data in HTML

(1)Getting the data from global:

```
myGlobal = myIris.get("myGlobal")
```

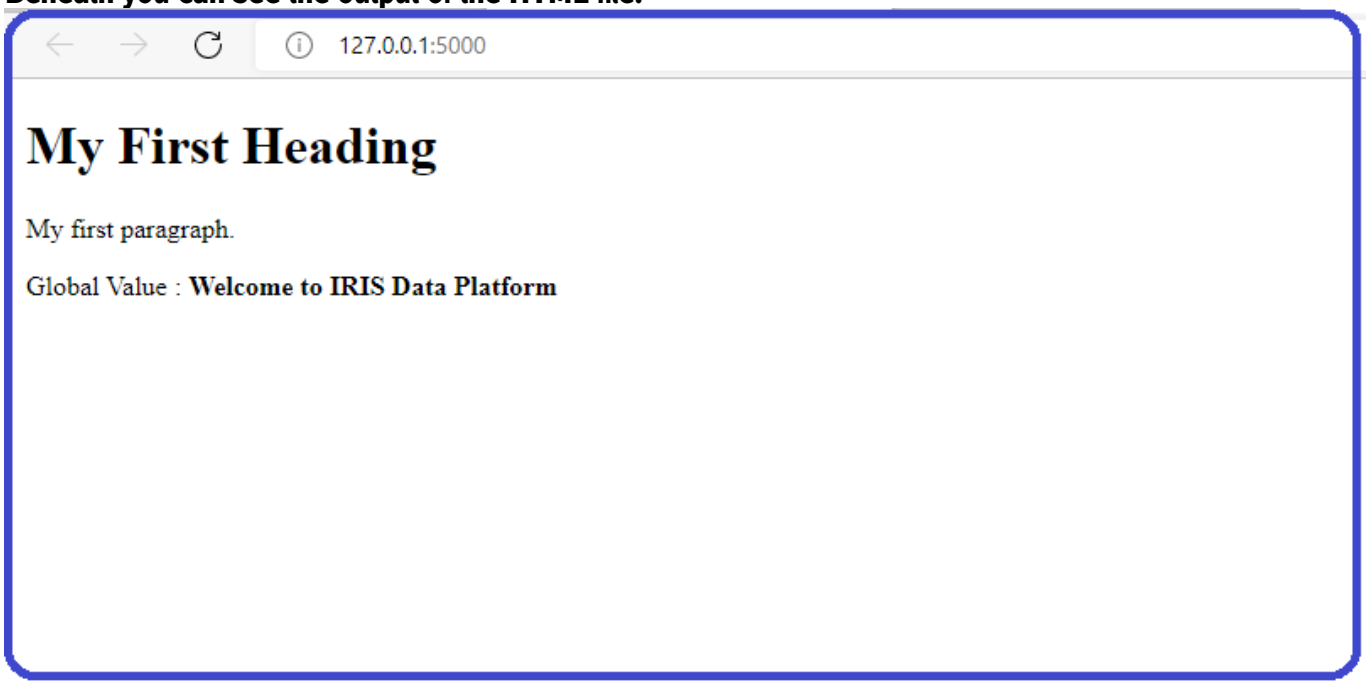
(2)The following code will pass myGlobal data to the index.html and render the template:

```
return render_template('index.html',MyGlobal = myGlobal)
```

(3) Below HTML code will display MyGlobal data:

```
<p>Global Value : <b>{{ MyGlobal }}</b></p>
```

Beneath you can see the output of the HTML file:



Summary

In this article after defining the Flask web framework, I demonstrated how to install the Flask web framework, create a web application, use HTML templates, install the IRIS python native module, establish a connection with IRIS, transition data from IRIS to Flask and display it.

Check out the final python and HTML files below:

```
from flask import Flask,render_template
```

```
import irisnative

app = Flask(__name__)

### Native API connection's parameters
ip = "localhost"
port = 1972
namespace = "USER"
username = "_SYSTEM"
password = "SYS"
###Create database connection and IRIS instance
try:
    #Return a new open connection to an IRIS instance.
    connection = irisnative.createConnection(ip, port, namespace, username, password)
    #Return a new irisnative.iris object that uses the given connection.
    myIris = irisnative.createIris(connection)
except Exception as e:
    print(e)

#getting the data from global
myGlobal = myIris.get("myGlobal")

@app.route("/")
def hello_world():
    #Pass myGlobal data and render HTML
    return render_template('index.html',MyGlobal = myGlobal)

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

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
<p>Global Value : <b>{{ MyGlobal }}</b></p>
</body>
</html>
```

In the next article, I will cover Routing in Flask Framework, the use of Static folder, Bootstrap, getting and displaying table data.

Thanks

#Globals #HTML #Python #Caché #InterSystems IRIS for Health