Lab # 21

Getting Familiar with Flask in Python

Objective:

Getting Familiar with Flask in Python.

Theory: Flask

- ✓ Flask is a lightweight and flexible web framework for **Python.**
- ✓ It's designed to make getting started with **web development** quick and easy, while still being powerful enough to build complex web applications.
- ✓ It is an API of Python that allows us to build web applications.
- ✓ It was developed by Armin Ronacher.
- ✓ Flask's framework is more explicit than **Django's framework** and is also easier to learn because it has less base code to implement a simple web application.
- ✓ **Microframework:** Flask is considered a "micro" web framework because It doesn't come with the full set of tools like Django provide,
- ✓ Flask is built on top of **two powerful libraries:**
    1. **Werkzeug:** WSGI(Web Server Gateway Interface) **web server library** that helps manage the application's request and response cycles
    2. **Jinja2:** A **templating engine** that allows you to use **dynamic HTML** in your application, making it easy to build web pages with variables and loops.

Primary Terminologies:

- **Framework:** A set of tools and code that helps you build programs faster without starting from scratch.
- **Web Framework:** A framework specifically for building websites and web applications.
- **API (Application Programming Interface):** A way for programs to talk to each other or use each other's functions.
- **WSGI (Web Server Gateway Interface):** A standard that helps Python web applications communicate with web servers.
- **Werkzeug:** A tool Flask uses to handle requests from users and send responses back.
- **Jinja2:** A tool that lets you put Python data into HTML pages to make them dynamic.
- **HTML (HyperText Markup Language):** The language used to create and display web pages in a browser

Lab # 21

Flask Web Application Flow

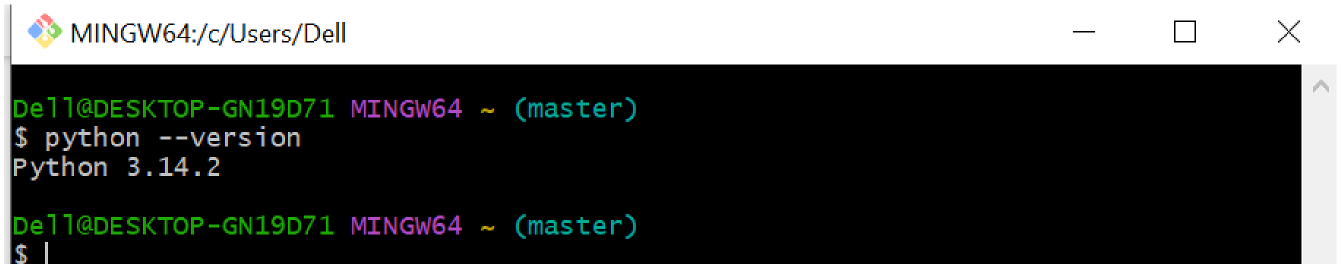
```
         +---------------+
         |   Browser     |
         | (User Request) |
         +-------+-------+
                 |
                 v
         +---------------+
         |   WSGI Server  |  <- Standard interface for Python web apps
         +-------+-------+
                 |
                 v
         +---------------+
         |   Flask App    |  <- Microframework handling routes & logic
         +-------+-------+
                 |
     +-------------+-------------+
     |                           |
     v                           v
+-----------+               +-----------+
| Werkzeug  |               |  Jinja2   |
| (Handles  |               | (Templates|
| requests  |               |  HTML)    |
| & responses)              |           |
+-----------+               +-----------+
     |                           |
     +-------------+-------------+
                   |
                   v
          +---------------+
          |   HTML Page     |
          | (Displayed in   |
          |   Browser)      |
          +---------------+
```

Lab # 21

Step # 01: Install Python

https://www.python.org/downloads/ **(latest version recommended, e.g., 3.12.x).** If It's already installed then check python Version.

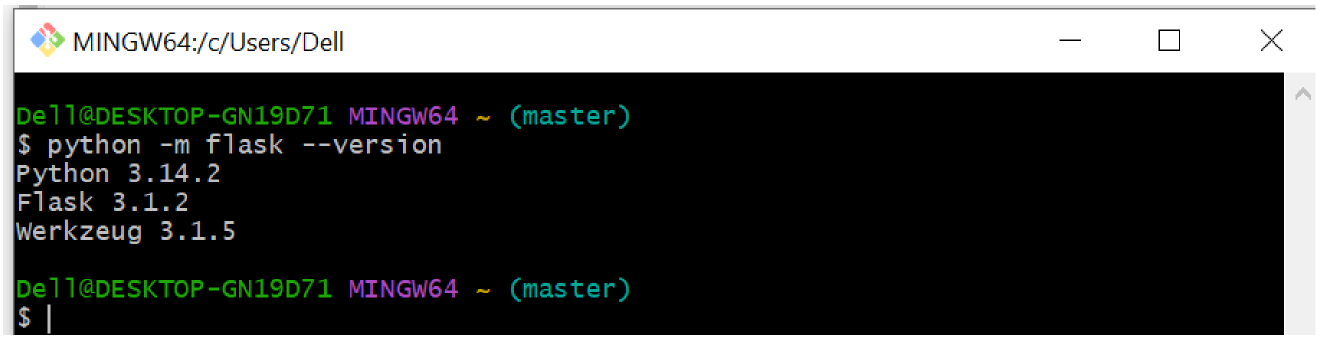
Step # 02: Verify Python Installation



```
Dell@DESKTOP-GN19D71 MINGW64 ~ (master)
$ python --version
Python 3.14.2

Dell@DESKTOP-GN19D71 MINGW64 ~ (master)
$ |
```

Step # 03: Upgrade pip & Install Flask

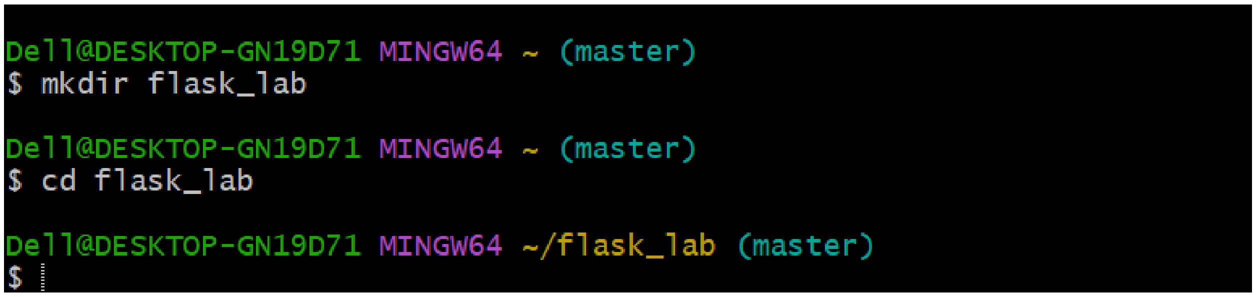
- **python -m pip install --upgrade pip**
- **pip install Flask**
- **python -m flask –version**



```
Dell@DESKTOP-GN19D71 MINGW64 ~ (master)
$ python -m flask --version
Python 3.14.2
Flask 3.1.2
Werkzeug 3.1.5

Dell@DESKTOP-GN19D71 MINGW64 ~ (master)
$ |
```

Step # 04: Create a Basic Flask App - Folder

- **mkdir "file_name"**
- **cd "file_name"**

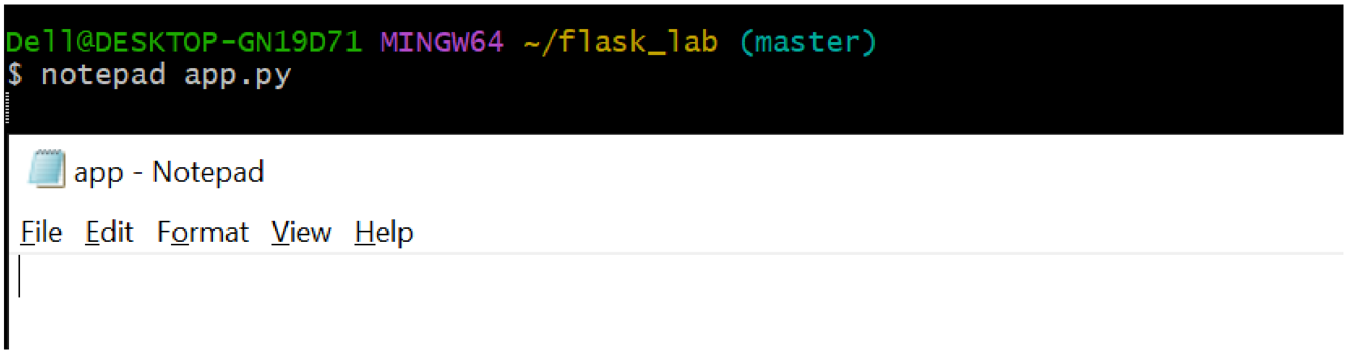


```
Dell@DESKTOP-GN19D71 MINGW64 ~ (master)
$ mkdir flask_lab

Dell@DESKTOP-GN19D71 MINGW64 ~ (master)
$ cd flask_lab

Dell@DESKTOP-GN19D71 MINGW64 ~/flask_lab (master)
$
```

Lab # 21

Step # 04: Create a Python file app.py:

- **notepad app.py**



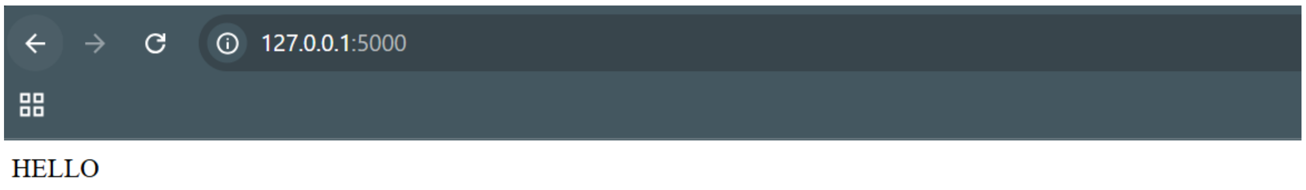
```
Dell@DESKTOP-GN19D71 MINGW64 ~/flask_lab (master)
$ notepad app.py
```

Step # 05: Add this code in Notepad.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'HELLO'

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

Step # 06: Open a browser and see the output.

- **http://127.0.0.1:5000/**



```
HELLO
```

Lab # 21

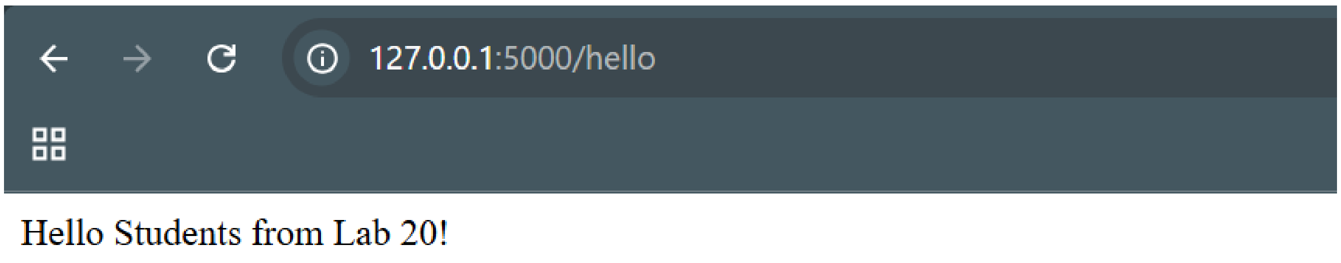
Step # 07: Flask Routes and Variables

1. Static Route

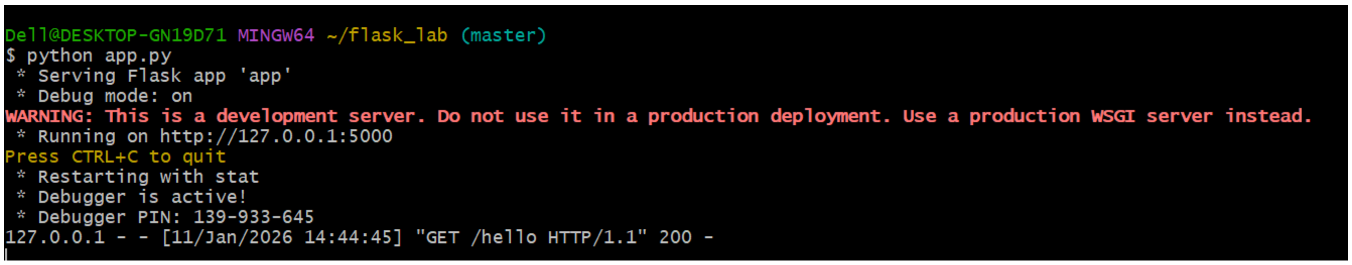
```
from flask import Flask

app = Flask(__name__)

@app.route('/hello')
def hello_world():
    return 'Hello Students from Lab 20!'

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

Output: Browser

- **http://127.0.0.1:5000/hello**



Hello Students from Lab 20!



```
Dell@DESKTOP-GN19D71 MINGW64 ~/flask_lab (master)
$ python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 139-933-645
127.0.0.1 - - [11/Jan/2026 14:44:45] "GET /hello HTTP/1.1" 200 -
```

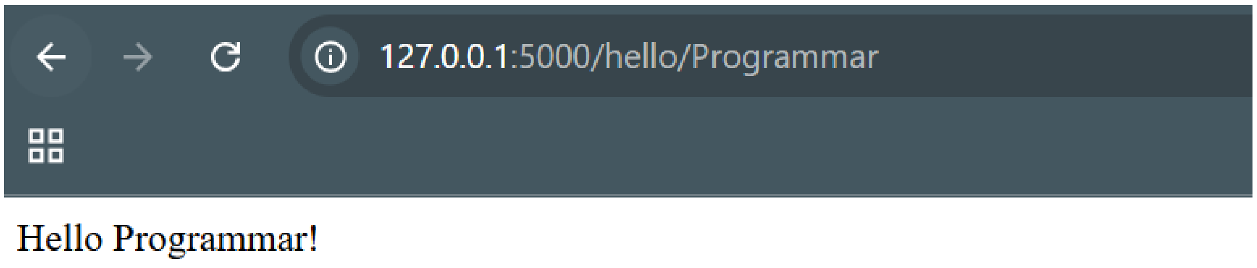
Lab # 21

2. Dynamic Route with Variable

```
from flask import Flask

app = Flask(__name__)

@app.route('/hello/<name>')
def hello_name(name):
    return f'Hello {name}!'

if __name__ == '__main__':

    app.run(debug=True)
```

Output: Browser

- **http://127.0.0.1:5000/hello/Programmar**



Hello Programmar!

Lab # 21

Step # 08: Build a URL in Flask

Dynamic Building of the URL for a specific function is done using url_for() function. The function accepts the name of the function as first argument, and one or more keyword arguments. See this example.

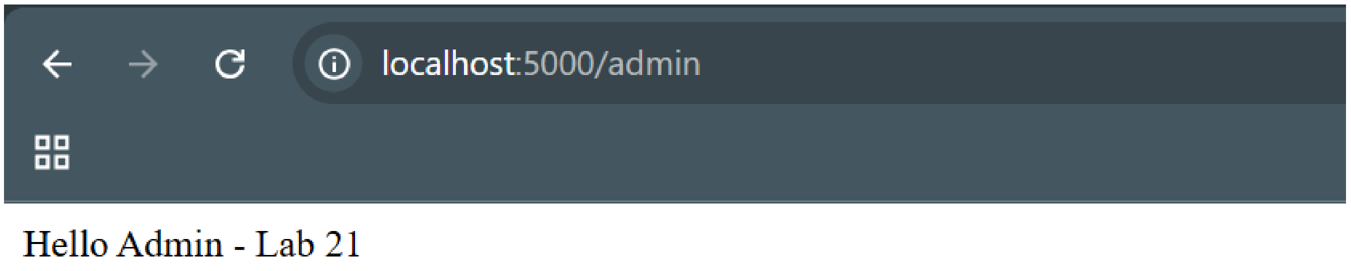
Code

```
from flask import Flask, redirect, url_for

app = Flask(__name__)


@app.route('/admin')  # decorator for route(argument) function

def hello_admin():  # binding to hello_admin call

   return 'Hello Admin'


@app.route('/guest/<guest>')

def hello_guest(guest):  # binding to hello_guest call

   return 'Hello %s as Guest' % guest


@app.route('/user/<name>')

def hello_user(name):

   if name == 'admin':  # dynamic binding of URL to function

      return redirect(url_for('hello_admin'))

   else:

      return redirect(url_for('hello_guest', guest=name))


if __name__ == '__main__':

   app.run(debug=True)
```

Lab # 21

Outputs:

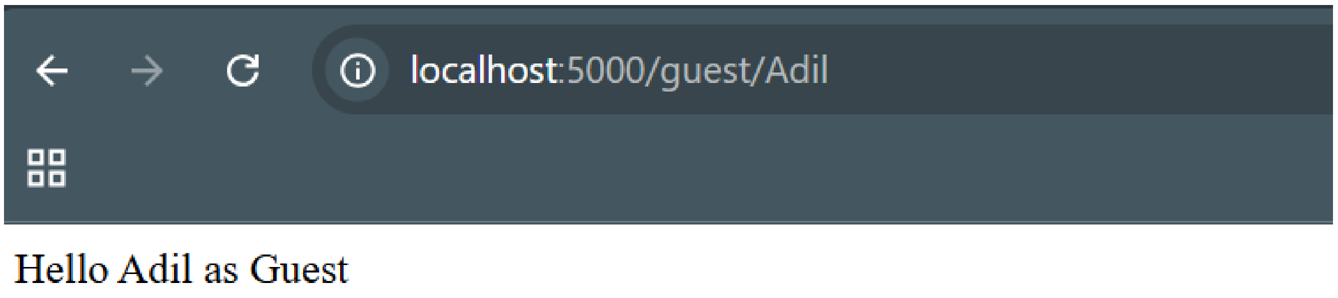
- **http://localhost:5000/admin**



localhost:5000/admin

Hello Admin - Lab 21

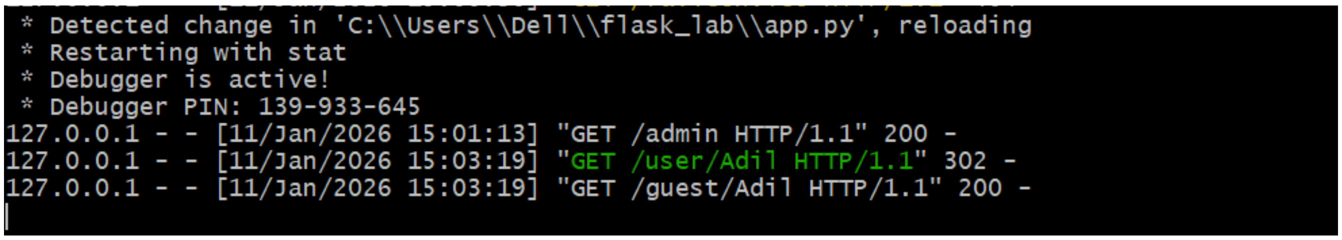
- **http://localhost:5000/guest/Adil**



localhost:5000/guest/Adil

Hello Adil as Guest

Bash / Cmd Screen



```
 * Detected change in 'C:\\Users\\Dell\\flask_lab\\app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 139-933-645
127.0.0.1 - - [11/Jan/2026 15:01:13] "GET /admin HTTP/1.1" 200 -
127.0.0.1 - - [11/Jan/2026 15:03:19] "GET /user/Adil HTTP/1.1" 302 -
127.0.0.1 - - [11/Jan/2026 15:03:19] "GET /guest/Adil HTTP/1.1" 200 -
```

Lab # 21

Exercise

1. **Create a new route /goodbye**
    - Returns the string: "Goodbye World!"
    - Test it in the browser: http://127.0.0.1:5000/goodbye
2. **Create a dynamic route /greet/<name>**
    - Returns "Hello <name>! Welcome to Flask!"
    - Test examples:
        - /greet/Alice → Hello Alice! Welcome to Flask!
        - /greet/Bob → Hello Bob! Welcome to Flask!
3. **Create a new route /age/<int:age>**
    - Returns "You are <age> years old!"
    - Test example: /age/25 → You are 25 years old!
4. **URL Redirection Practice**
    - Create 2 new routes: /teacher and /student/<name>
    - If the user visits /student/admin, redirect to /teacher
    - Otherwise, display "Hello <name>, welcome student!"