Python | Flask

Session 1

- Introduction
- Routing
- Requests

by Mohammad Amin H.B. Tehrani

www.maktabsharif.ir

Introduction

What is Flask?

Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it. Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

Flask is **Micro-Framework**:

"Micro" does not mean that your whole web application has to fit into a single Python file (although it certainly can), nor does it mean that Flask is lacking in functionality. The "micro" in microframework means Flask aims to keep the core simple but extensible. Flask won't make many decisions for you, such as what database to use. Those decisions that it does make, such as what templating engine to use, are easy to change. Everything else is up to you, so that Flask can be everything you need and nothing you don't.

Installation

Install Venv (Virtual Environment):
virtualenv is a virtual Python environment builder. It helps a user to create multiple Python environments side-by-side.

Install venv:

```
pip install virtualenv
```

Create virtual environment:

```
mkdir newproj
cd newproj
virtualenv venv
```

Linux:

```
venv/bin/activate
```

Windows:

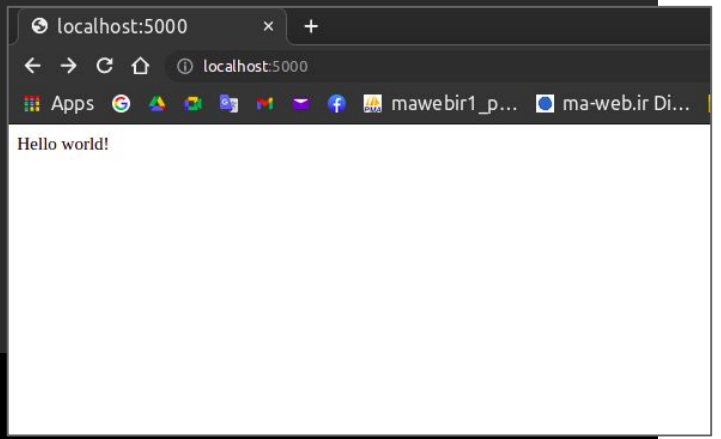
```
venv\scripts\activate
```

Install Flask:

```
pip install Flask
```

Example: Hello world!

```python
from flask import Flask  # Import flask

app = Flask(__name__)  # Create app instance

@app.route('/')
def hello world():
   return "Hello world!"

# Run the app on localhost..
app.run()
```




```
$ python3 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: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Run Application

No.	Parameters & Description
1	**host** Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to ‘0.0.0.0’ to have server available externally.
2	**port** Defaults to 5000
3	**debug** Defaults to false. If set to true, provides a debug information
4	**options** To be forwarded to underlying Werkzeug server.

Routing

Routing

app.route() decorator

App routing is used to map the specific URL with the associated function that is intended to perform some task. It is used to access some particular page.

```
@app.route('/')
def index():
   return "Index page..."

@app.route('/home')
def home():
   return "Home page..."

@app.route('/about-us')
def about_us():
   return "About-us page..."
```

Routing

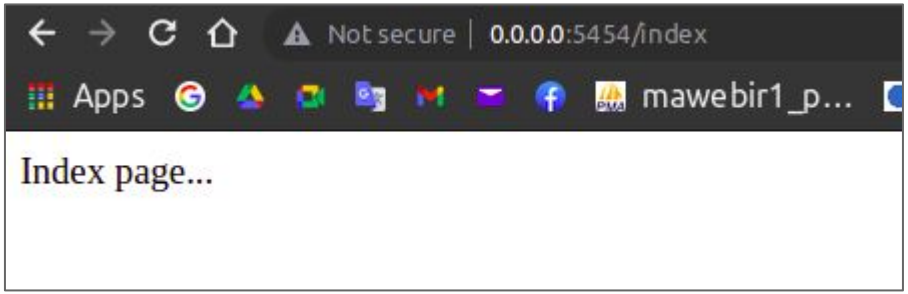
add_url_route() function

There is one more approach to perform routing for the flask web application that can be done by using the add_url() function of the Flask class. The syntax to use this function is given below.

```
def index page():
   return "Index page..."

def home page():
   return "Home page..."

def about_us_page():
   return "About-us page..."

app.add url rule('/', 'index', index page)
app.add url rule('/home', 'home', home page)
app.add_url_rule('/about-us', 'about-us', about_us_page)
```

Multi-URL routing

```
@app.route('/')
@app.route('/index')
def index():
    return "Index page..."
```



Routing

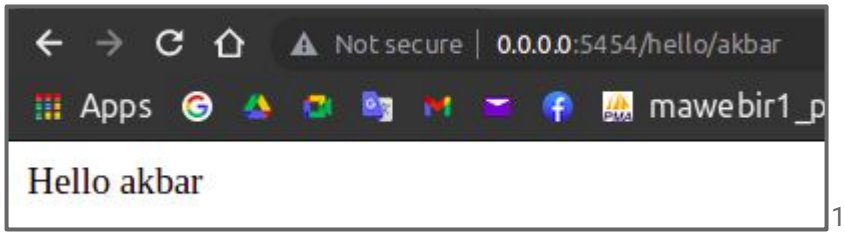
Dynamic URLs

It is possible to build a URL dynamically, by adding variable parts to the rule parameter. This variable part is marked as <variable-name>. It is passed as a keyword argument to the function with which the rule is associated.

Syntax:

```
@app.route('.../<variable_name1>/<variable_name2>/...' )
def view_func(variable_name1, variable_name2, ... ):
   ...
```

```
@app.route('/hello/<name>')
def hello(name):
   return "Hello "+name
```



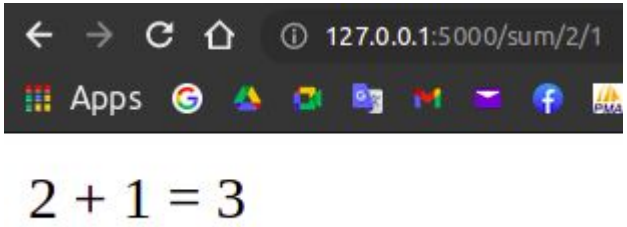
Dynamic URLs with type

"default": UnicodeConverter,
"string": UnicodeConverter,
"any": AnyConverter,
"path": PathConverter,
"int": IntegerConverter,
"float": FloatConverter,
"uuid": UUIDConverter,

Syntax:

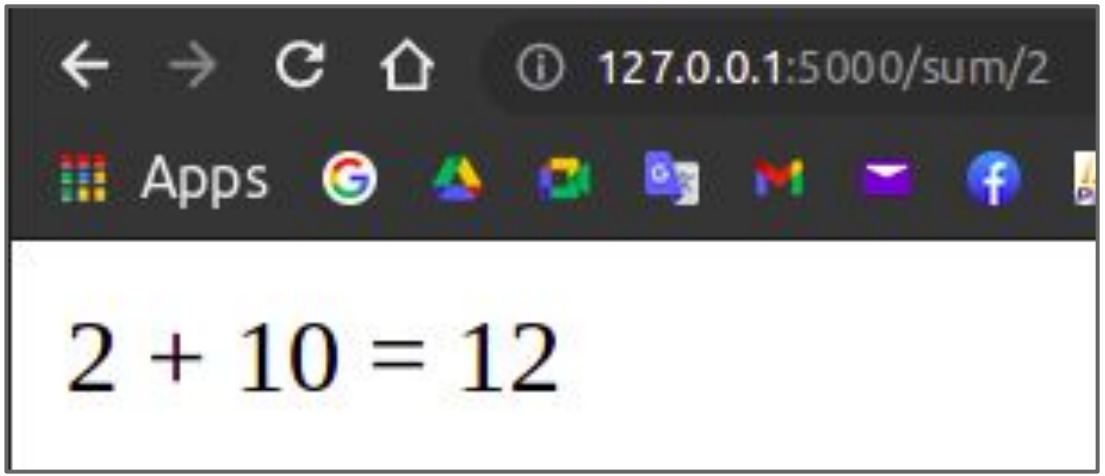
```
@app.route('.../<type:variable_name>')
def view_func(variable_name):
    ...
```

```
@app.route('/sum/<int:a>/<int:b>')
def sum(a, b):
    return f"{a} + {b} = {a+b}"
```



Dynamic URLs with Optional Parameters

```
@app.route('/pow/<int:a>', defaults={'b': 2})
@app.route('/pow/<int:a>/<int:b>')
def pow(a, b):
    return f"a**b = {a**b}"
```

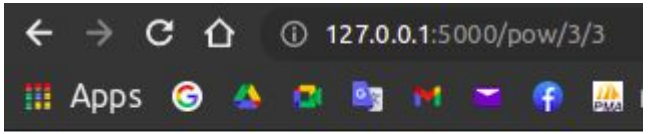


Practice: Power

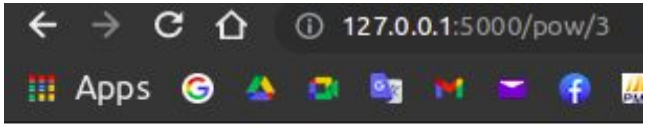
Power

Write a flask app view, with following functionality:

ex.:
/pow/3/3 -> 3 ** 3 = 27
/pow/3 -> 3 ** 2 = 9



3 ** 3 = 27



3 ** 2 = 9

Routing

Redirect() function

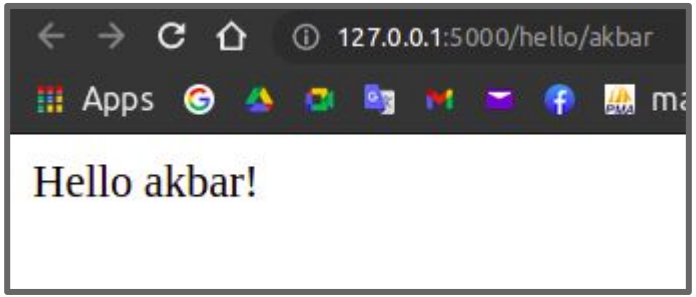
Flask class has a redirect() function. When called, it returns a response object and redirects the user to another target location with specified status code.

```
from flask import Flask, redirect

...


@app.route('/')
def index():
   return redirect('https://www.google.com')
```

url_for() function

The **url_for**() function is very useful for dynamically building a URL for a specific function. The function accepts the name of a function as first argument, and one or more keyword arguments, each corresponding to the variable part of URL.

```
from flask import Flask, redirect, url_for

...


@app.route('/')
def index():
   return redirect(url_for('hello', name='akbar'))

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



Routing

HTTP methods

By default, the Flask route responds to the GET requests. However, this preference can be altered by providing methods argument to route() decorator.

```
@app.route('/')
def get_view():
   return "get request!"


@app.route('/', methods=['POST'])
def post_view():
   return "POST request!"


@app.route('/', methods=['PATCH', 'PUT'])
def patch_put_view():
   return "PATCH or PUT request!"
```

```
curl -X get http://127.0.0.1:5000/
get request!
```

```
curl -X post http://127.0.0.1:5000/
POST request!
```

```
curl -X patch http://127.0.0.1:5000/
PATCH or PUT request!
```

```
curl -X put http://127.0.0.1:5000/
PATCH or PUT request!
```

Requests

Intro

You can simple import and use **`request`**, to handle http requests, and pass it into your view functions as **`request`**:

```
from flask import Flask, request

...


@app.route('/')
def index():
    print("Method", request.method)
    print("Args", request.args)
    print("Url", request.url)
    print("Headers", request.headers)
    return request.args
```

Some request attributes

1. request.args: the key/value pairs in the URL query string
2. request.form: the key/value pairs in the body, from a HTML post form, or JavaScript request that isn't JSON encoded
3. request.files: the files in the body, which Flask keeps separate from form. HTML forms must use enctype=multipart/form-data or files will not be uploaded.
4. request.values: combined args and form, preferring args if keys overlap
5. request.json: parsed JSON data. The request must have the application/json content type, or use request.get_json(force=True) to ignore the content type.

Some request attributes

All of these are MultiDict instances (except for json). You can access values using:

- **request.form['name']:** use indexing if you know the key exists
- **request.form.get('name'):** use get if the key might not exist
- **request.form.getlist('name'):** use getlist if the key is sent multiple times and you want a list of values. get only returns the first value.

Requests

Example

```
@app.route('/', methods=['POST', 'GET'])
def index():
    print("Method", request.method)
    print("Args", request.args)
    print("Form", request.form)
    print("Files", request.files)
    print("Values", request.values)
    print("Json", request.json)
    print("Data", request.data)
    return {
        "Args": request.args,
        "Form": request.form,
        "Files": list(request.files.keys()),
        "Values": request.values,
        "Json": request.json,
        "data": request.data,
    }
```

Practice: User registration page

User registration page

First create an User class with attributes like id, first_name, last_name, phone, email, ...
Then create a Flask app:

A. **POST** request to **/ (index page):**
   Register users using form data.
B. **GET** request to **/ (index page):**
   Show list of registered users
C. **GET** request to **/<int:id>**
   Show information of user #id

Hint: use python dict as output for **JSON Response**