

Flask - Python

Flask is back-end web framework, born in 2010, that can browser requests, and user session between requests; route HTTP request to the controllers, evaluate form data, respond to HTML and JS request, and so on.

First web app with flask

In this session, we will write our first app explained line by line; we will also cover how to set up our environment, what tools to use for development, and how to work with HTML in our app.

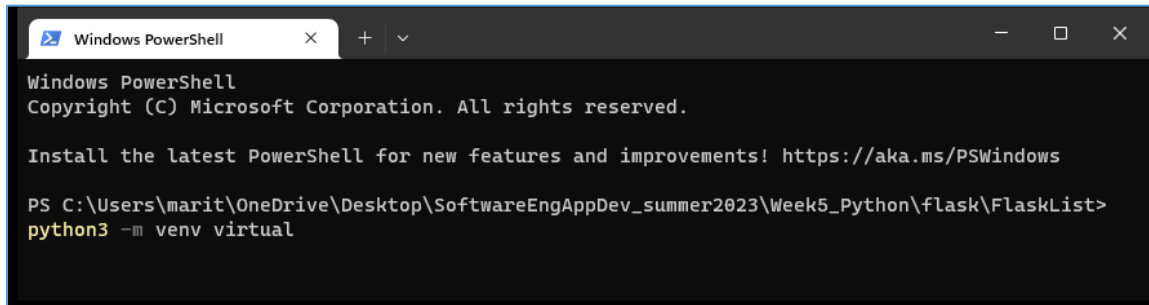
Installation and tools

Step 1) install virtual environment

One way to create a virtual environment is used *venv* instead of *virtualenv*.

To do so, create a project folder and open the Terminal with the path to the project folder. After it, type the following code at the Terminal:

Windows and Mac: **python3 -m venv virtual**

A screenshot of a Windows PowerShell terminal window. The title bar says 'Windows PowerShell'. The text inside the terminal shows the copyright notice for Microsoft Corporation, a link to update PowerShell, and the current directory path 'PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\FlaskList>'. The command 'python3 -m venv virtual' is entered at the prompt.

Alternative step 1)

In the command window, let's install the virtual environment for python to work with flask. In the command terminal, go to your project folder, or root folder, and type:

Windows: `pip install virtualenv`

mac: `sudo pip install virtualenv`

`pip` is the default Python package management tool and helps us install the `virtualenv` package.

```
Microsoft Windows [Version 10.0.17134.1365]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Student>cd C:\Users\Student\Desktop\JavaScript

C:\Users\Student\Desktop\JavaScript>pip install virtualenv
Collecting virtualenv
  Downloading https://files.pythonhosted.org/packages/ef/a1/4e1700f25211b3851e6be6675061e0c8eae7585d80177a40e9b02d1105d8/virtualenv-20.13.0-py2.py3-none-any.whl (6.5MB)
    100% |#####| 6.0MB 232kB/s
Collecting importlib-metadata>=0.12; python_version < "3.8" (from virtualenv)
  Downloading https://files.pythonhosted.org/packages/a0/a1/b153a0a4caf7a7e3f15c2cd56c7702e2cf3d89b1b359d1f1c5e59d68f4ce/importlib_metadata-4.8.3-py3-none-any.whl
Collecting filelock<4,>=3.2 (from virtualenv)
  Downloading https://files.pythonhosted.org/packages/84/ce/8916d10ef537f3b046843255f9799504aa41862bfa87844b9bdc5361cd/filelock-3.4.1-py3-none-any.whl
Collecting platformdirs<3,>=2 (from virtualenv)
  Downloading https://files.pythonhosted.org/packages/b1/78/dcf8d4d3aabd46a9c77260fb47ea5d244806e4daef83aa6fe5d83adb182c/platformdirs-2.4.0-py3-none-any.whl
Collecting distlib<1,>=0.3.1 (from virtualenv)
  Downloading https://files.pythonhosted.org/packages/ac/a3/8ee4f54d5f12e16eeda6b7df3dfdbda24e6cc572c86ff959a4ce110391b/distlib-0.3.4-py2.py3-none-any.whl (461kB)
    100% |#####| 471kB 2.6MB/s
Collecting importlib-resources>=1.0; python_version < "3.7" (from virtualenv)
  Downloading https://files.pythonhosted.org/packages/24/1b/33e489669a94da3ef4562938cd306e8fa915e13939d7b8277cb5569cb405/importlib_resources-5.4.0-py3-none-any.whl
Collecting six<2,>=1.9.0 (from virtualenv)
  Downloading https://files.pythonhosted.org/packages/d9/5a/e7c31adbe875f2abbb91bd84cf2dc52d792b5a01506781dbcf25c91daf11/six-1.16.0-py2.py3-none-any.whl
Collecting zipp>=0.5 (from importlib-metadata>=0.12; python_version < "3.8"->virtualenv)
  Downloading https://files.pythonhosted.org/packages/bd/df/d4a4974a3e3957fd1c1fa3082366d7fff6e428ddb55f07bf64876f8e8ad/zipp-3.6.0-py3-none-any.whl
Collecting typing-extensions>=3.6.4; python_version < "3.8" (from importlib-metadata>=0.12; python_version < "3.8"->virtualenv)
  Downloading https://files.pythonhosted.org/packages/05/e4/baf0031e39cf545f0c9edd5b1a2ea12609b7fcb2d58e118b11753d68cf0/typing_extensions-4.0.1-py3-none-any.whl
Installing collected packages: zipp, typing-extensions, importlib-metadata, filelock, platformdirs, distlib, importlib-resources, six, virtualenv
Successfully installed distlib-0.3.4 filelock-3.4.1 importlib-metadata-4.8.3 importlib-resources-5.4.0 platformdirs-2.4.0 six-1.16.0 typing-extensions-4.0.1 virtualenv-20.13.0 zipp-3.6.0

C:\Users\Student\Desktop\JavaScript>
```

A virtual environment is the way Python isolates full package environments from one another. This means you can easily manage dependencies. Imagine you want to define the minimum necessary packages for a project; a virtual environment would be perfect to let you test and export the list of needed packages.

Once you install *virtualenv*, the warning message will display:

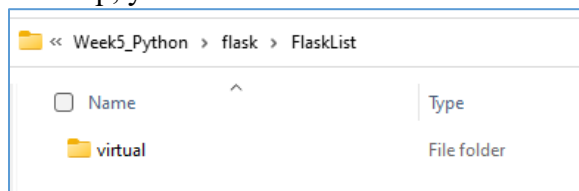
WARNING: The script virtualenv.exe is installed in 'C:\Users\marit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfr8a8p0\LocalCache\local-packages\Python311\Scripts' which is not on PATH. Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.

Once the first step is completed, type:

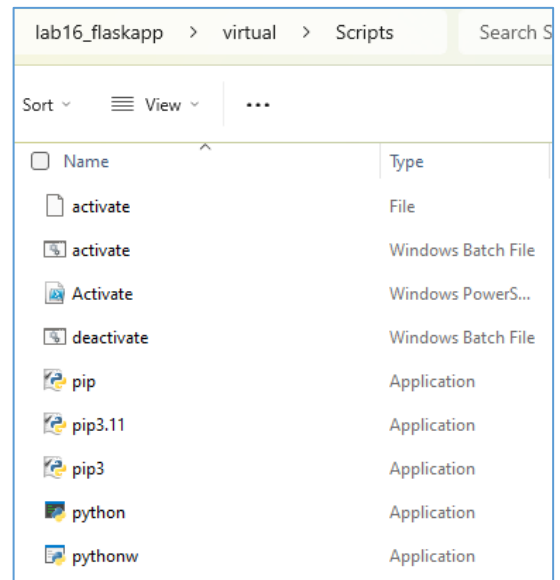
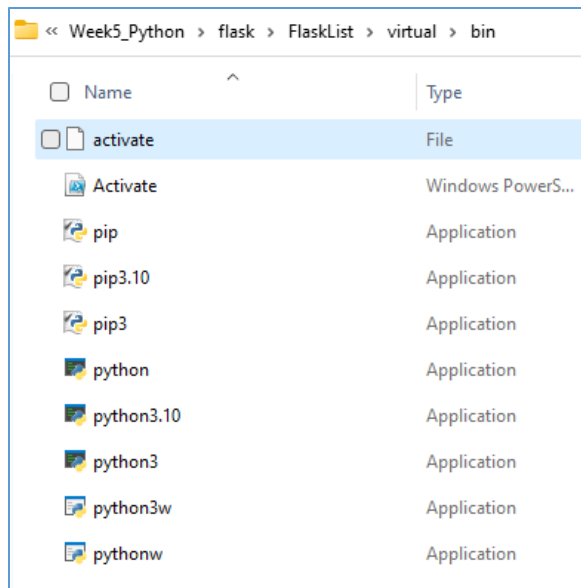
Windows and Mac: **python3 -m virtualenv virtual**

Step 2) Checking the virtual environment directory

This step specifies which folder you want to install virtual environment to be created in. After this step, you should see **virtual** folder created under your app project root folder.



If we open the virtual folder, it should have the virtual files. We can find the path to the **activate** file, which is the file that will connect us to the virtual machine, which is in the 'bin' or 'Scripts' folder:



Step 3) Activate virtual environment

activate will trigger your virtual environment under 'bin' or 'Scripts' folder. To connect to the virtual environment, we need to type the path to the activate file:

For windows

*(root folder) > **virtual\bin\activate***

or

*(root folder) > **virtual\Scripts\activate***

For Mac and Linux

source ./virtual/bin/activate

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\FlaskList>
python3 -m venv virtual
virtual^Cers\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\FlaskList>
PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\FlaskList>
virtual\bin\activate
(virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\
FlaskList>
```

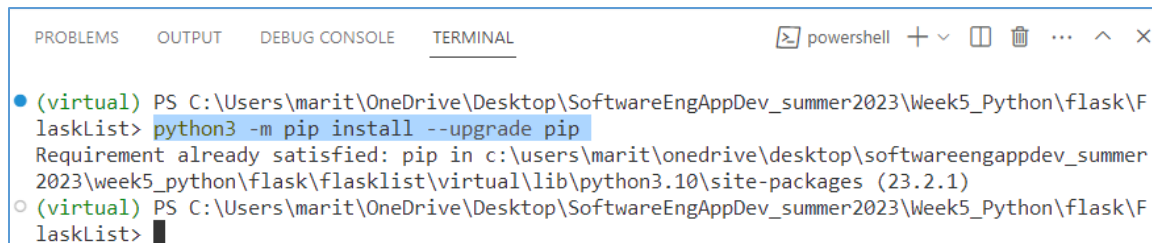
Step 4) Installing Flask

After step 3, point to virtual folder, *bin*, then we can install flask. Before installing Flask, we can update pip first by typing the following code in the terminal:

```
python3 -m pip install --upgrade pip
```

depending on the system, the Terminal will automatically set the command to upgrade pip. Some system might show the command:

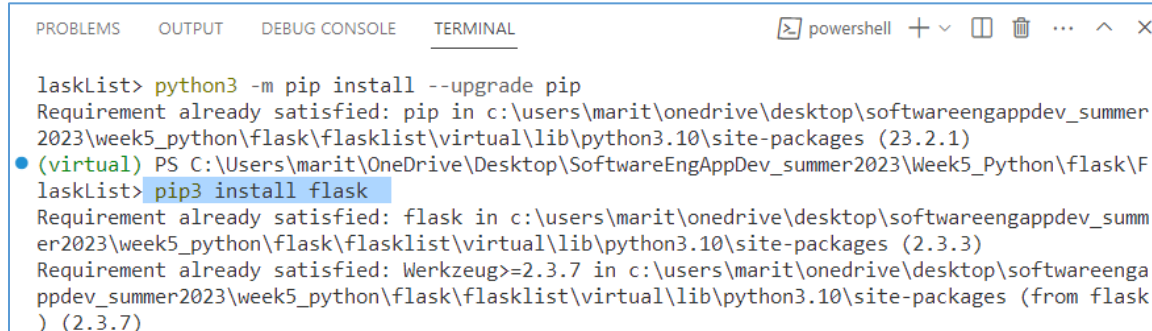
```
python.exe -m pip install --upgrade pip
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL powershell + - [ ] [ ] ... ^ X
• (virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\F
  lasklist> python3 -m pip install --upgrade pip
  Requirement already satisfied: pip in c:\users\marit\onedrive\desktop\softwareengappdev_summer
  2023\week5_python\flask\flasklist\virtual\lib\python3.10\site-packages (23.2.1)
○ (virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\F
  lasklist> █
```

Now, we can install Flask by typing:

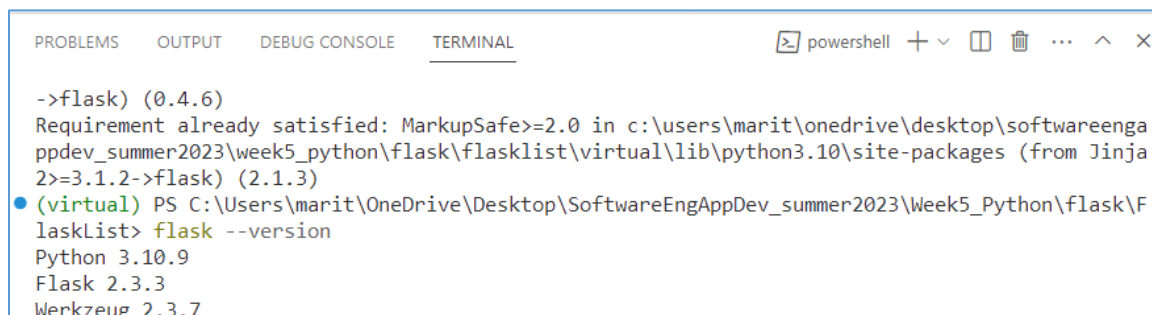
```
(virtual)(root folder) > pip3 install flask
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL powershell + - [ ] [ ] ... ^ X
lasklist> python3 -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\marit\onedrive\desktop\softwareengappdev_summer
2023\week5_python\flask\flasklist\virtual\lib\python3.10\site-packages (23.2.1)
• (virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\F
  lasklist> pip3 install flask
  Requirement already satisfied: flask in c:\users\marit\onedrive\desktop\softwareengappdev_summ
  er2023\week5_python\flask\flasklist\virtual\lib\python3.10\site-packages (2.3.3)
  Requirement already satisfied: Werkzeug>=2.3.7 in c:\users\marit\onedrive\desktop\softwareenga
  ppdev_summer2023\week5_python\flask\flasklist\virtual\lib\python3.10\site-packages (from flask
  ) (2.3.7)
```

We can check if *Flask* is installed by typing:

```
flask --version
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL powershell + - [ ] [ ] ... ^ X
->flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\marit\onedrive\desktop\softwareenga
ppdev_summer2023\week5_python\flask\flasklist\virtual\lib\python3.10\site-packages (from Jinja
2>=3.1.2->flask) (2.1.3)
• (virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\F
  lasklist> flask --version
  Python 3.10.9
  Flask 2.3.3
  Werkzeug 2.3.7
```

We can visit [Python and Flask Tutorial in Visual Studio Code](https://code.visualstudio.com/docs/python/tutorial-flask) <https://code.visualstudio.com/docs/python/tutorial-flask> to check more about Flask installation.

After having the environment development set-up to work with Flask, we can start building our app project. The app that we are creating will add an item to a list.

Setting up Flask environment variables

Flask environment variables are defined as a set of parameters that facilitates the running of application developed in Flask. An environment for flask applications is essentially a directory or a placeholder that contains all the pre-requisite for successfully running an application. It is assumed to be a surrounding created in order for an application to run. The environment variables come in handy as it allows users to tweak parameters enabling them download tools for building flask application to run, test, and operationalize only for the sake of running the application without hampering the system-wide variables. This also allows users to categorize the type of environment for the flask application, the details of which we will learn in the forthcoming sections!

How Environment Variables work in Flask?

Before we start learning about environment variables, it is necessary for us to know about the environments which Flask supports. It is the availability of different environments that reason to having multiple configurations because every configuration file is environment-related. The several categories in which the environment is segregated are:

- **Development:** This environment consists of a set of processes and programming tools that facilitate the developer to build the program or the software product. This environment acts like an experimental playground for the developer so that one can test, debug, experiment, and then finalize on the tools that will go into the final program.
- **Testing:** This environment relates to the system having configurations built for enabling developers to run test cases enhancing the confidence in the actual product. The development environment might consist of various tools as a result of experimentation, and we don't know the dependencies, if applicable, on any of the tools. As a result, the testing environment helps create an identical environment every time one needs to test the product so that the integration in the production environment is seamless.
- **Production:** This environment consists of a set of processes and programming tools that facilitate the end-user for their operations. The software developed in the development environment and after getting tested in the testing environment gets actually put into the production environment for operation of the end-users. This is a real-time setting that has all the required hardware installed and relied on for any commercial purposes.

Now that we have a fair idea of all the environment applicable in the context of Flask, we would now look at what environment variables are and their working. In the introduction session, we have extensively gone through the definition of the environment variables, which we can in short describe as a set of parameters that facilitates the running of application in the prescribed environment we intend the application to run.

While we use these parameters to ease off some repetitive coding for our flask application, we should keep in mind that the usage is totally voluntary and one can easily switch to either of the ways as per the requirement of the application developed. (Ghoshal, 2023)

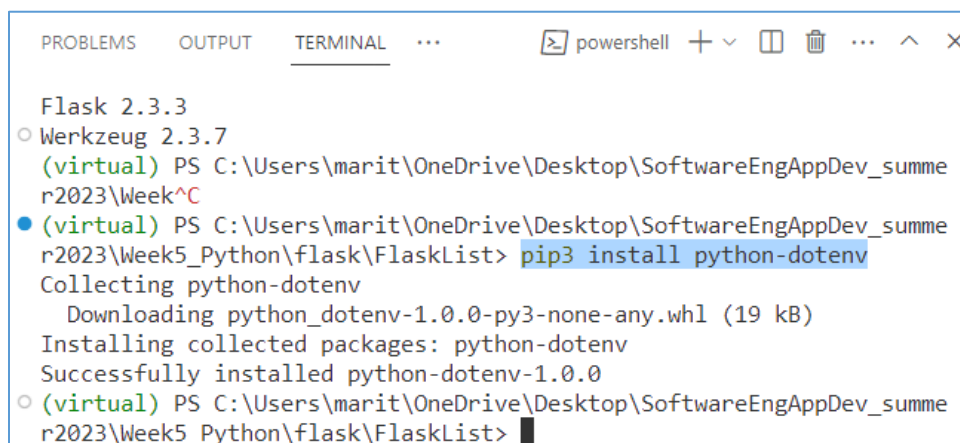
For our app, we are going to create a file with *.flaskenv* and set the variables inside *.flaskenv*. The two key variables we need to set up are *FLASK_ENV* and *FLASK_APP*. It is important to set the *FLASK_ENV* because if you had it in production, which is the default, then every time you make a change, you have to restart the server every time.

The next environmental variable is the application module *FLASK_APP*, which is the one that runs the application.

Once you have set this environment variables inside the *.flaskenv* file, you will need to install a special package in order to invoke these variables. The package to install is:

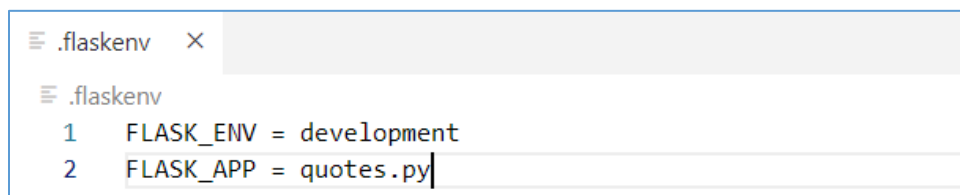
pip3 install python-dotenv

Without this package, the variables will not be invoked. So, the *-dotenv* is a module that loads your environment variables.



```
Flask 2.3.3
○ Werkzeug 2.3.7
(virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summe
r2023\Week^C
● (virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summe
r2023\Week5_Python\flask\FlaskList> pip3 install python-dotenv
Collecting python-dotenv
  Downloading python_dotenv-1.0.0-py3-none-any.whl (19 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.0.0
○ (virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summe
r2023\Week5_Python\flask\FlaskList> █
```

After it, open Visual Studio Code text editor, create **.flaskenv** file under your App Root Folder. In the *.flaskenv* file, we are going to set the variables environment as:



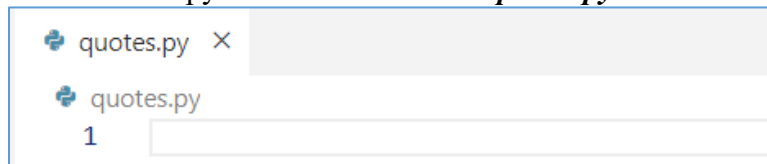
```
.flaskenv
1 FLASK_ENV = development
2 FLASK_APP = quotes.py
```

Flask app template

For our Flask app project, we are going to:

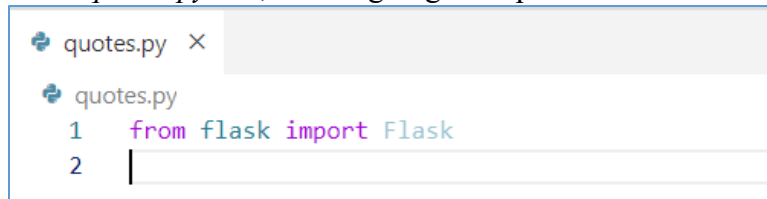
- set a python module file and save it with **.py** extension.
- Import class Flask.
- Instantiate the class.
- Create a default route.
- Run the app

Let's create a python file and name it **quotes.py**



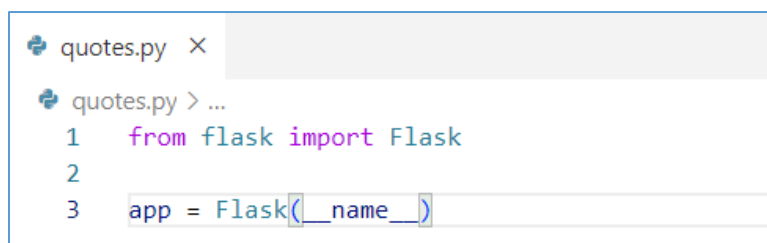
```
quotes.py X
quotes.py
1
```

In the **quotes.py** file, we are going to import the class Flask:



```
quotes.py X
quotes.py
1 from flask import Flask
2
```

The next line we are going to create an instance of this class and then store it in a variable called **app**. Inside the parenthesis for the class code Flask, the first argument is the name of the application.



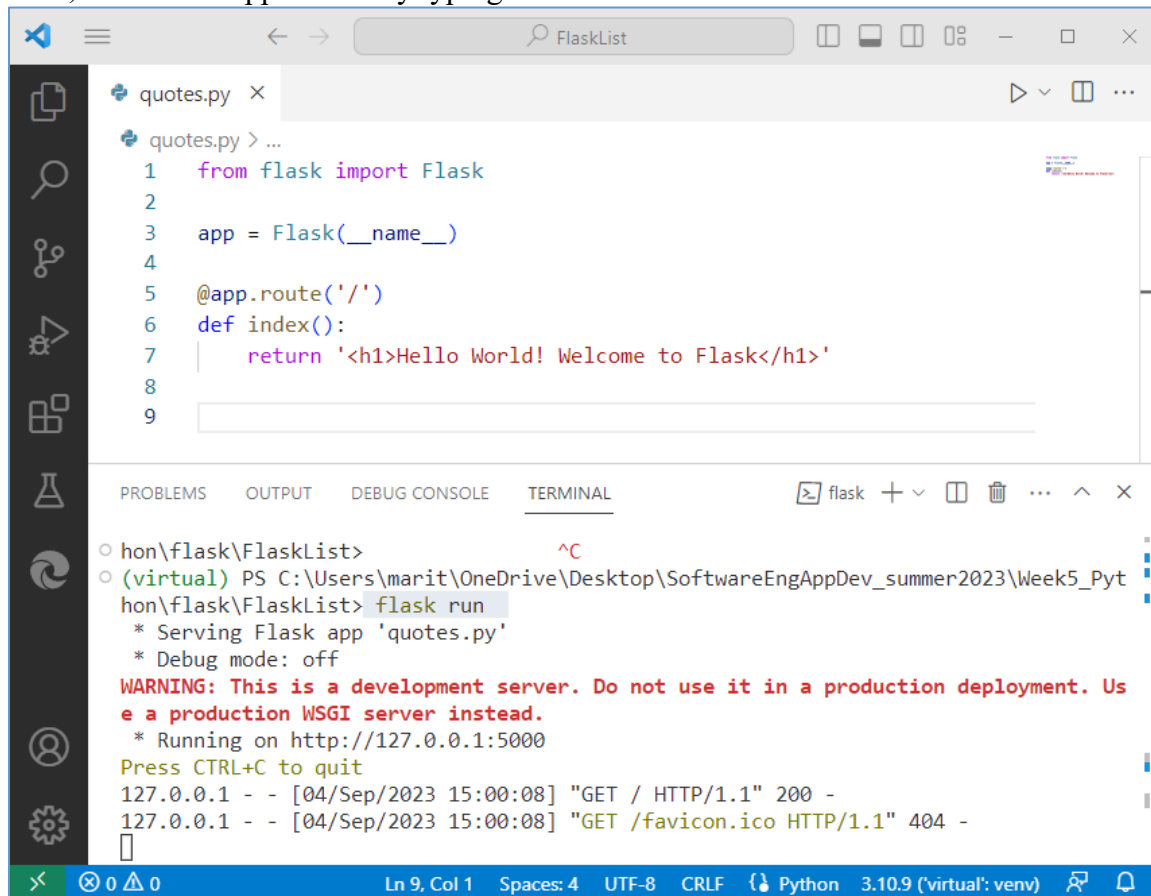
```
quotes.py X
quotes.py > ...
1 from flask import Flask
2
3 app = Flask(__name__)
```

The next thing we are going to do is to use a root decorator to create a root for the application. So, the root decorator is basically going to tell the Flask application what we can use to trigger the function. Every root decorator must have a function that it decorates that function as a function that is triggered when someone visits that particular root:

```
quotes.py x
quotes.py > index
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return '<h1>Hello World! Welcome to Flask</h1>'
```

The default root of the route has the `'/'`

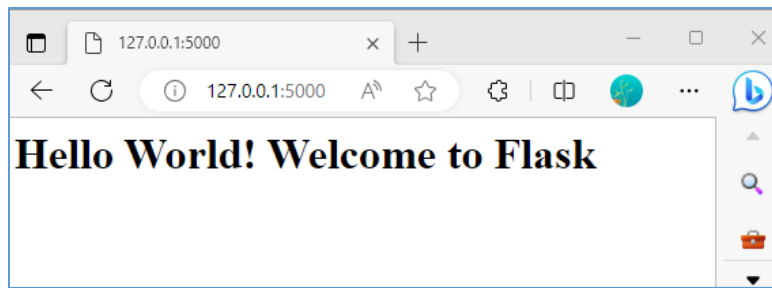
Now, we run our application by typing **flask run** in our terminal:



The screenshot shows the Visual Studio Code editor with a file named `quotes.py` open. The code defines a Flask application with a single route at the root. Below the editor, the terminal window is active, showing the command `flask run` being executed. The output indicates that the application is running on `http://127.0.0.1:5000` and shows some initial HTTP requests.

```
quotes.py x
quotes.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return '<h1>Hello World! Welcome to Flask</h1>'
8
9
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
hon\flask\FlaskList> ^C
(virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Pyt
hon\flask\FlaskList> flask run
* Serving Flask app 'quotes.py'
* Debug mode: off
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
127.0.0.1 - - [04/Sep/2023 15:00:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2023 15:00:08] "GET /favicon.ico HTTP/1.1" 404 -
[]
```

Opening the browser to `http://127.0.0.1:5000` we have:

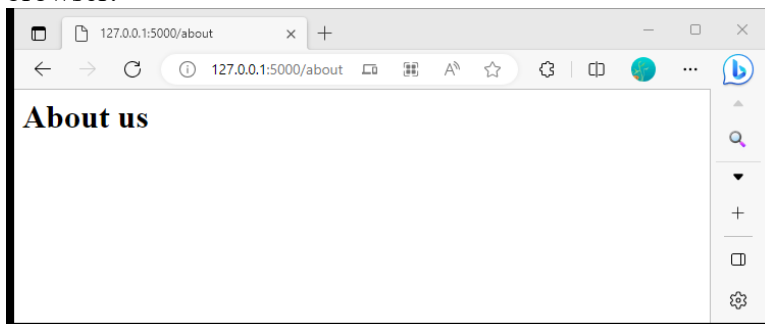


endpoints

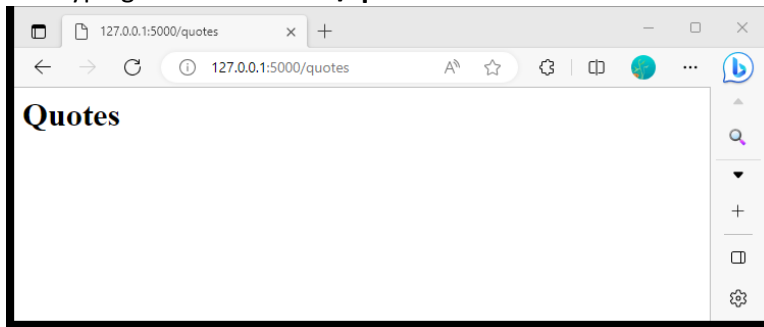
Now we are going to add an *endpoint* to our app. *endpoints* refer to the name of the view function. A view function is the function that responds to requests from your application. For our app, we can create two endpoints *about* and *quotes*:

```
quotes.py x
quotes.py > about
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return '<h1>Hello World! Welcome to Flask</h1>'
8
9  @app.route('/about')
10 def about():
11     return '<h1>About us</h1>'
12
13 @app.route('/quotes')
14 def quotes():
15     return '<h1>Quotes</h1>'
```

To test our endpoints, we are doing to run the app again and type **127.0.0.1:5000/about** at the browser:



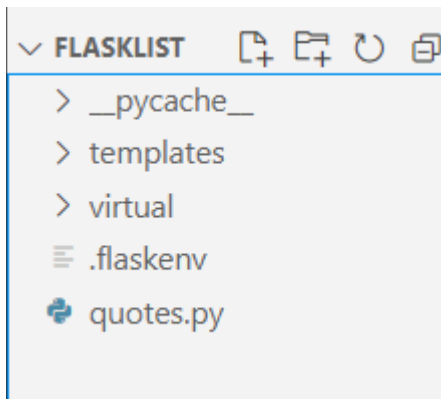
And typing **127.0.0.1:5000/quotes**



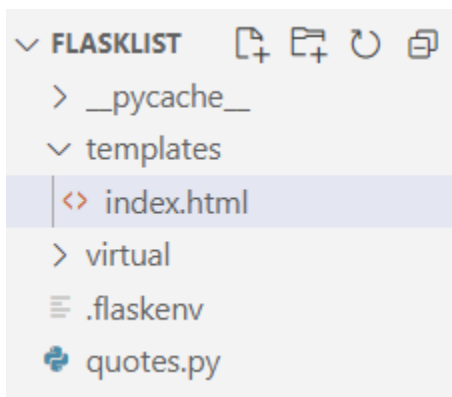
Rendering templates

Templates help programs to separate the Flask code and the HTML code.
Flask looks for templates in a folder named **templates** in the app project root folder.
The method named **render_template** is used to render templates to your Flask application.

So, for our app, we are going to take out the HTML and render it from a template HTML file.
To do so, we are going to create a **templates** directory (folder).



Now, inside of the **templates** directory, we are going to create an **index.html** file:

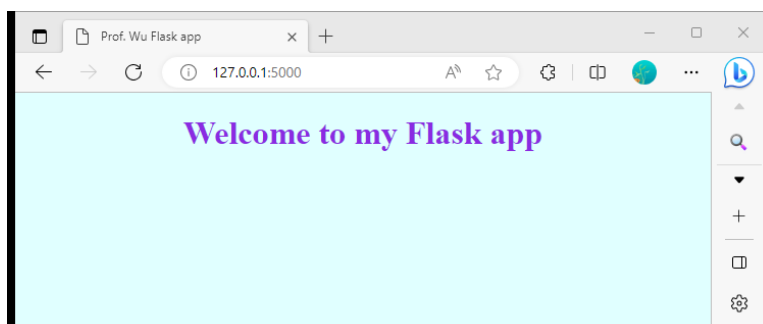


```
index.html x
templates > index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Prof. Wu Flask app</title>
7      <style>
8          body{background-color: lightcyan;}
9          .title{text-align: center; color: blueviolet;}
10     </style>
11 </head>
12 <body>
13     <h1 class="title">Welcome to my Flask app</h1>
14 </body>
15 </html>
```

Now, we can go back to the *quotes.py* file and import the `render_template` to the python file and modify the return of the `@app.route('/')` to `render_template`:

```
quotes.py x
quotes.py > ...
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template('index.html')
8
```

Running Flask again and refreshing the browser we will have:



In order to pass in variables inside the HTML using Jinja in Python. Jinja is a template engine for Python. It is similar to the Django template engine, where the information is passed using the `{% %}` syntax.

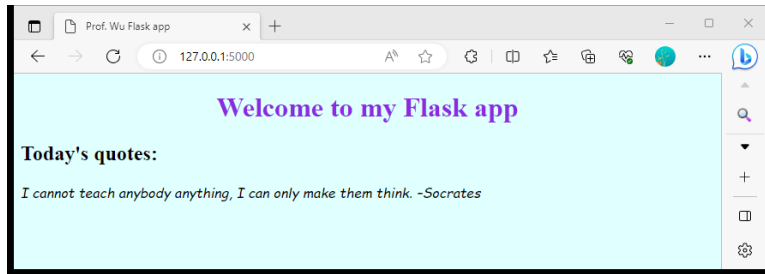
For our app, to pass Jinja template in variables into each HTML in order to parsing variables to the HTML method:

```
quotes.py x
quotes.py > index
1 from flask import Flask, render_template
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def index():
7     return render_template('index.html', quote1 = "I cannot teach anybody anything, I can only make them think. -Socrates")
8
9 @app.route('/about')
10 def about():
11     return '<h1>About us</h1>'
12
13 @app.route('/quotes')
14 def quotes():
15     return '<h1>Quotes </h1>'
16
```

After the template is rendered, to inject this variable into the HTML template, we need to use Jinja. To do so, open the `index.html` file, and then to inject that into this template, you do double curly braces, which is Jinja syntax, and then you the name of the variable, which is `quote1`:

```
<> index.html x
templates > <> index.html > html > body
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Prof. Wu Flask app</title>
7     <style>
8         body{background-color: lightcyan;}
9         .title{text-align: center; color: blueviolet;}
10        .quote{font-family: cursive; font-style: italic;}
11    </style>
12 </head>
13 <body>
14     <h1 class="title">Welcome to my Flask app</h1>
15     <h2>Today's quotes: </h2>
16     <p class="quote">{{quote1}}</p>
17 </body>
18 </html>
```

Running Flask again and refreshing the browser we will have:



Making decision in a Flask app

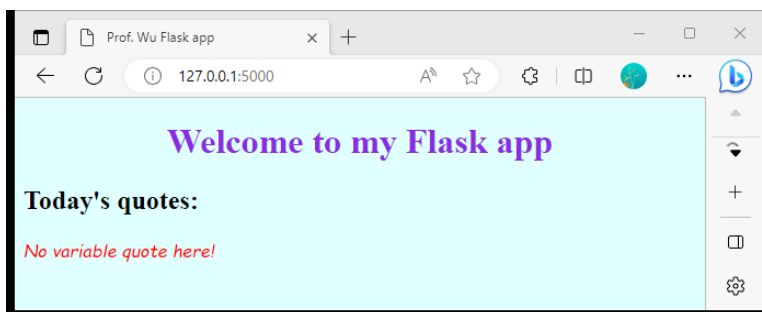
We are going to use the *if* statement to check if a variable *quote1* exists or not: if the variable exists the app will print the *quote1*, otherwise, the app will print out a different quote. To do so, we are going to *index.html* and use Jinja code to make the comparison. The syntax of the Jinja conditional statement is:

```
{% if quote1 %}  
    Display HTML elements if quote1 is true  
{% else %}  
    Display HTML elements if quote1 is false  
{% endif %}
```

```
<body>  
    <h1 class="title">Welcome to my Flask app</h1>  
    <h2>Today's quotes: </h2>  
    <!-- CONDITIONAL STATEMENT -->  
    {% if quote1 %}  
    <p class="quote">{{quote1}}</p>  
    {% else %}  
    <p class="no_quote">No variable quote here! </p>  
    {% endif %}  
</body>
```

If we have a quote in the variable *quote1*, the if statement will be true and paragraph with class *quote* will print. If there is no quote in the variable *quote1*, the if statement will be false and the paragraph with class ***no_quote*** will print.

```
quotes.py ×
quotes.py > index
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template('index.html', quote1 = "")
8
9  @app.route('/about')
10 def about():
11     return '<h1>About us</h1>'
12
13 @app.route('/quotes')
14 def quotes():
15     return '<h1>Quotes </h1>'
```



For loop in Flask

A for loop is used for iterating over a sequence like a list. For example, we are going to create a *color* list in the *quotes.py*

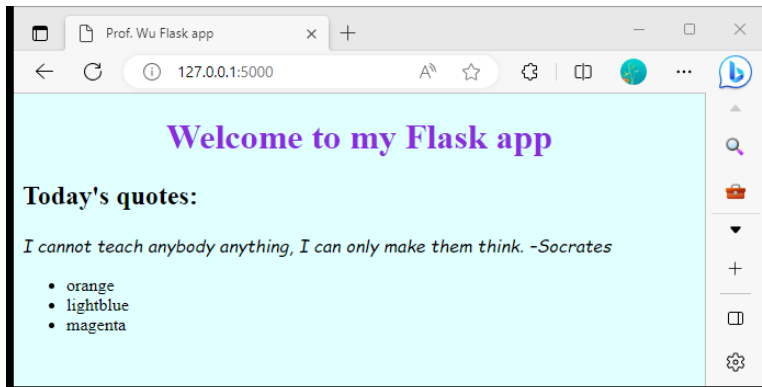
```
quotes.py ×
quotes.py > index
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      color = ['orange', 'lightblue', 'magenta']
8      return render_template('index.html', quote1 = "")
9
10 @app.route('/about')
11 def about():
12     return '<h1>About us</h1>'
13
14 @app.route('/quotes')
15 def quotes():
16     return '<h1>Quotes </h1>'
```

After it, we are going to add a variable named *colors* and makes it equal to the *color* list in the *render_template*:

```
quotes.py x
quotes.py > index
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      color=['orange','lightblue','magenta']
8      return render_template('index.html', quote1 = "I cannot teach anybody anything, I can only
      make them think. -Socrates", colors = color)
9
10 @app.route('/about')
11 def about():
12     return '<h1>About us</h1>'
13
14 @app.route('/quotes')
15 def quotes():
16     return '<h1>Quotes </h1>'
```

Now, we can add the *for* loop at the *index.html* file:

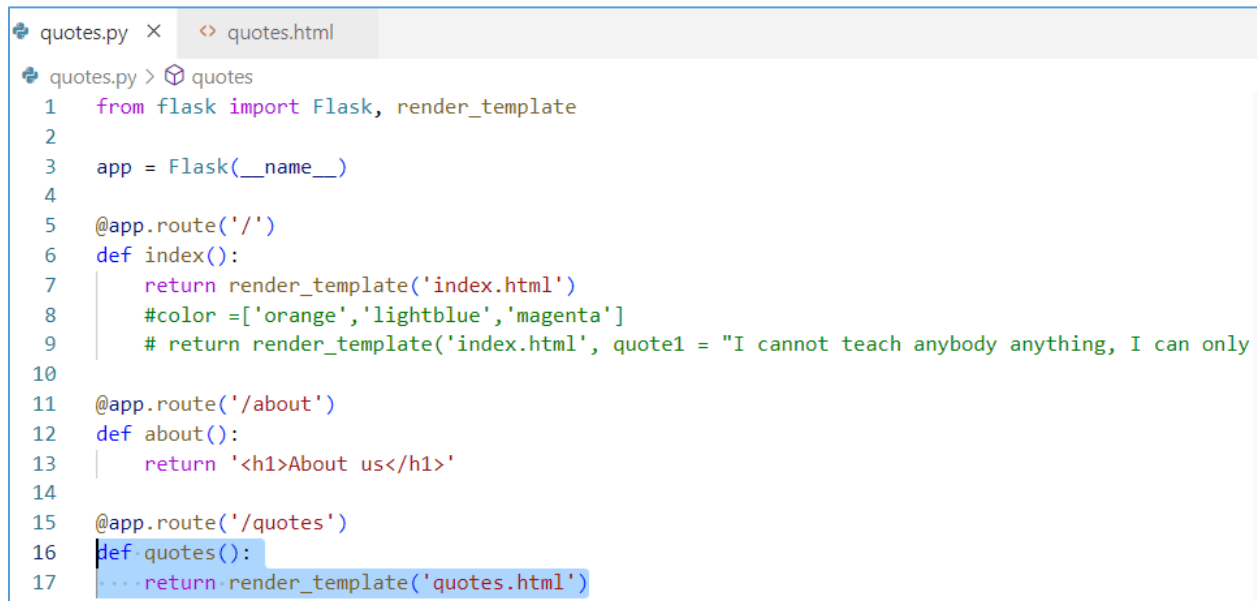
```
<!-- FOR LOOP -->
<ul>
    {%for c in colors %} <!-- print each item in list colors as variable c -->
    <li>{{ c }}</li> <!-- print each item in the list colors -->
    {% endfor %}
</ul>
```



You can also find some Flask tutorial at [Flask Tutorial: Templates --> https://pythonbasics.org/flask-tutorial-templates/](https://pythonbasics.org/flask-tutorial-templates/)

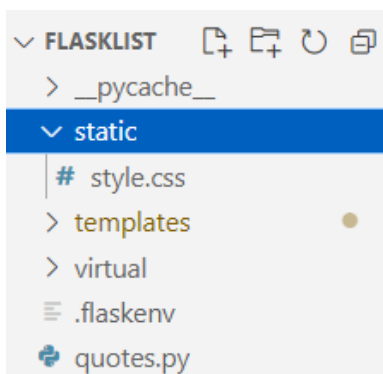
For our app, we are going to:

- Create a new HTML file name *quotes.html* in the *templates* folder.
- Modify *index.html* and *quotes.py* files.
- Styles the elements.



```
quotes.py > quotes
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template('index.html')
8      #color = ['orange', 'lightblue', 'magenta']
9      # return render_template('index.html', quote1 = "I cannot teach anybody anything, I can only
10
11  @app.route('/about')
12  def about():
13      return '<h1>About us</h1>'
14
15  @app.route('/quotes')
16  def quotes():
17      return render_template('quotes.html')
```

We are going to work on modifying and styling the *index.html* and *quotes.html* files. Let's us create a *style.css* file. The *style.css* file must be saved in a new folder in our root directory named **static**:



The 'index.html' file will be as:

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

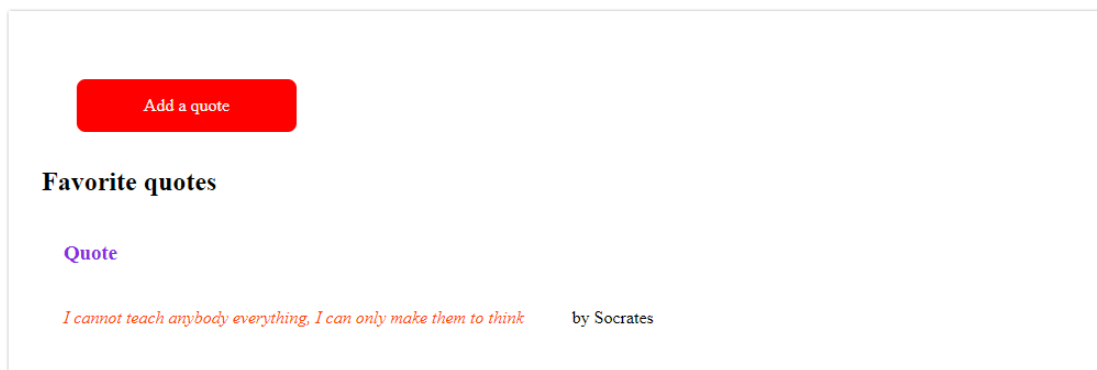
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Welcome to Flask</title>
  <link rel="stylesheet" href="../static/style.css">
</head>

<body>
  <h1>Student's name - Flask</h1>

  <main class="container">
    <nav class="addquotebtn">
      <a href="">Add a quote</a>
    </nav>
    <h2>Favorite quotes</h2>
    <div>
      <h3 class="favquotes">Quote 1</h3>
      <p>
        <em class="favquotes_quote">I cannot teach anybody everything,
        I can only make them to think</em>
        <span class="favquotes_author"> by Socrates</span>
      </p>
    </div>
  </main>

</body>
</html>
```

The localhost view for the 'index.html' webpage will be as:



The style of the file, style.css, will be as:

```
.addquotebtn{
    margin: 2em;
    background-color: red;
    color: whitesmoke;
    padding:15px;
    border-radius: 0.5em;
    width: 200px;
    text-align: center;
}
.addquotebtn a{
    color: whitesmoke;
    text-decoration: none;
}
.addquotebtn:hover{
    box-shadow: 0px 0px 3px 1px rgb(50,50,50);
    background-color: whitesmoke;
    color:black
}
.container{
    width: 1000px;
    margin: 30px auto;
    box-shadow: 1px 1px 3px rgb(50,50,50);
    padding: 30px;
}
.favquotes{
    color: blueviolet;
    padding: 20px;
}
.favquotes_quote{
    color: orangered;
    padding: 20px;
}
.favquotes_author{
    color: black;
    padding: 20px;
}
```

After it, let's prepare the *quotes.html* structure and styling:

```
<> quotes.html x
templates > <> quotes.html > html > body > main.container > form > div.form-group > l
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  |   <title>My quotes</title>
7  |   <link rel="stylesheet" href=" ../static/style.css">
8  </head>
9  <body>
10 |   <main class="container">
11 |     <form action="" method="post">
12 |       <div class="form-group">
13 |         <label for="inputName">Quote Author</label>
14 |         <input type="text" id="inputName" class="form-control"
15 |           placeholder="Add a quote author...">
16 |       </div>
17 |       <div class="form-group">
18 |         <label for="quote">Quote</label>
19 |         <textarea class="form-control" id="quote" rows="5"
20 |           placeholder="Add your quote..."></textarea>
21 |       </div>
22 |       <div class="form-group">
23 |         <input class="btn_submit" type="submit"/>
24 |       </div>
25 |     </form>
26 |   </main>
27 </body>
28 </html>
```

```

.container{
  width: 80%;
  margin: 5em auto;
  box-shadow: 1px 1px 3px ■rgb(50,50,50);
  background-color: □aliceblue;
}

.form-group{
  padding: 1em;
}

.form-group label{
  display: block;
  margin-bottom: 0.3em;
  font-size: 1.1em;
  font-weight: 600;
}

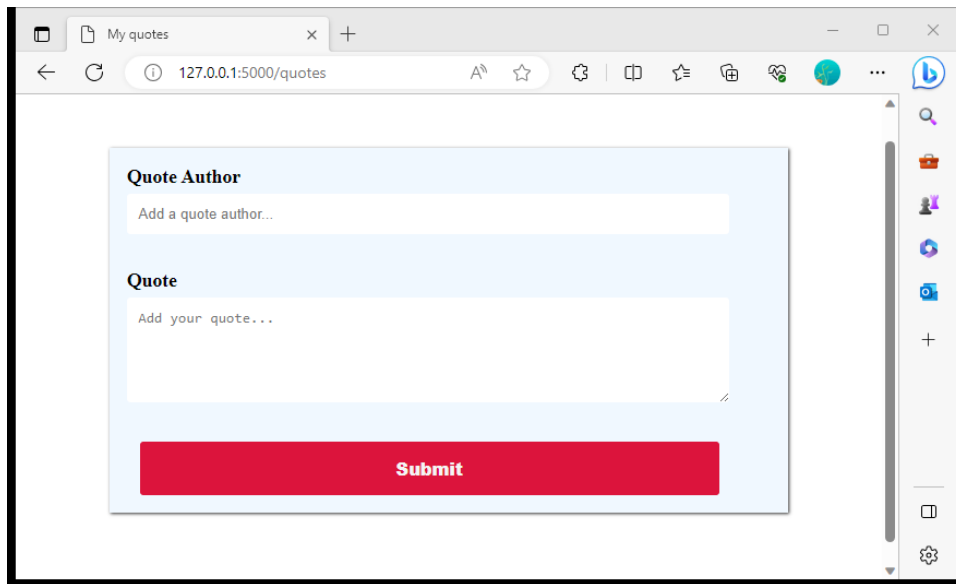
.form-group input, .form-group textarea{
  padding: 0.8em;
  width: 90%;
  border: none;
  border-radius: 3px;
}

.form-group input:hover, .form-group textarea:hover{
  box-shadow: 0px 0px 2px 1px ■lightslategray;
}

.btn_submit{
  color: □whitesmoke;
  text-decoration: none;
  background-color: ■crimson;
  margin: 0% 2%;
  font-size: 1em;
  font-weight: 900;
}

.btn_submit:hover{
  box-shadow: 0px 0px 3px 1px ■rgb(50,50,50);
  background-color: □rgb(250,250,250);
  color: ■crimson;
}

```



After it, now, if the data is clicked on the Submit button, the data in the form will go to the database. In the *quotes.py* file, we are going to add a new endpoint to *process*. The *process* will render back to the *quotes.html* page.

```
quotes.py > process
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template('index.html')
8      #color = ['orange', 'lightblue', 'magenta']
9      # return render_template('index.html', quote1 = "I cannot teach anybody anything,
10
11 @app.route('/about')
12 def about():
13     return '<h1>About us</h1>'
14
15 @app.route('/quotes')
16 def quotes():
17     return render_template('quotes.html')
18
19 @app.route('/process')
20 def process():
21     ...return render_template('quotes.html')
```

Within the *process()* function, we are going to add a redirect, but before we can do that, we need to import the request object. The request object is what is going to give us access to all the data that the form captures.

```
quotes.py X
quotes.py > ...
1  from flask import Flask, render_template, request, redirect, url_for
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template('index.html')
8      #color = ['orange', 'lightblue', 'magenta']
9      # return render_template('index.html', quote1 = "I cannot teach anybody anything,"
10
11  @app.route('/about')
12  def about():
13      return '<h1>About us</h1>'
14
15  @app.route('/quotes')
16  def quotes():
17      return render_template('quotes.html')
18
19  @app.route('/process')
20  def process():
21      return render_template('quotes.html')
```

Now to need redirect our *process* endpoint to specified URL. For this, in the route of process:

- add the methods use to transfer the data from the form. In this case, we want to be explicit and set it to POST, which means post would be my HTTP method for my form.
- change the return to redirect to the homepage *index.html*
- create two variables: one variable is going to save the author information and the other one will save the actual quote information.

```
quotes.py x
quotes.py > process
1  from flask import Flask, render_template, request, redirect, url_for
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template('index.html')
8      #color = ['orange', 'lightblue', 'magenta']
9      # return render_template('index.html', quote1 = "I cannot teach anybody anything,
10
11  @app.route('/about')
12  def about():
13      return '<h1>About us</h1>'
14
15  @app.route('/quotes')
16  def quotes():
17      return render_template('quotes.html')
18
19  @app.route('/process', methods=['POST'])
20  def process():
21      ...author = request.form['author']
22      ...quote = request.form['quote']
23      ...return redirect('index.html')
```

Next thing we need to do is to pass in this route what we have created into the template that contains the form. For this, in the *quotes.html* file, we are going to submit the value for the form action. This action attribute basically is what is going to process the form's data:

```
quotes.py  quotes.html x
templates > <> quotes.html > html > body > main.container
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>My quotes</title>
7      <link rel="stylesheet" href="../../static/style.css">
8  </head>
9  <body>
10     <main class="container">
11         <form action="/process" method="post">
12             <div class="form-group">
13                 <label for="inputName">Quote Author</label>
14                 <input type="text" id="inputName" class="form-control" placeholder="Add a quote au
15             </div>
16
17             <div class="form-group">
18                 <label for="quote">Quote</label>
19                 <textarea class="form-control" id="quote" rows="5" placeholder="Add your quote..."
20             </div>
21
22             <div class="form-group">
23                 <input class="btn_submit" type="submit"/>
24             </div>
25         </form>
26     </main>
27 </body>
28 </html>
```

Database integration with Flask app

To integrate information from the Flask app's form to the database, we will need to install a couple of components: *psycopg2* and *SQLAlchemy*:

- *psycopg2* is basically a database driver or adapter. This is used to enable communication between any Python program and a PostgreSQL database system. To install *psycopg2* we type **pip3 install psycopg2** in the terminal.
- *Alchemy* is a library use as an Object Relational Mapper (ORM) tool that translates Python classes to tables of relational databases and automatically converts function calls to SQL statements. To install *SQLAlchemy* we type **pip3 install Flask-SQLAlchemy** in the terminal.

In the *quotes.py*

```
quotes.py 1 X
quotes.py > index
1  from flask import Flask, render_template, request, redirect, url_for
2  from flask_sqlalchemy import SQLAlchemy
3
4  app = Flask(__name__)
5
6  # connection from form to database
7  app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql+psycopg2://postgres:password@localhost/quotes'
8  app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
9
10 @app.route('/')
11 def index():
12     return render_template('index.html')
13     #color = ['orange', 'lightblue', 'magenta']
14     # return render_template('index.html', quote1 = "I cannot teach anybody anything, I can only mak
15
```

After it, we are going to create an instance of SQLAlchemy and then we are passing the app to the data to the db.

To do this we need to create a class *Favquotes*:

```
quotes.py \ 1 X  quotes.py C:\...example-quotes 1
quotes.py > ...
1  from flask import Flask, render_template, request, redirect, url_for
2  from flask_sqlalchemy import SQLAlchemy
3
4  app = Flask(__name__)
5
6  # connection from form to database
7  app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql+psycopg2://postgres:password@localhost/quotes'
8  app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
9
10 # transferring data from form to database
11 db = SQLAlchemy(app)
12
13 class Favquotes(db.Model):
14     id = db.Column(db.Integer, primary_key=True)
15     author = db.Column(db.String(30))
16     quote = db.Column(db.String(2000))
```

In the terminal, go to type **py** and then **from quotes import db**

```
(virtual) PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\FlaskList> deactivat
PS C:\Users\marit\OneDrive\Desktop\SoftwareEngAppDev_summer2023\Week5_Python\flask\FlaskList> py
Python 3.12.0rc1 (tags/v3.12.0rc1:63bcd91, Aug  5 2023, 13:51:29) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from quotes import db
```

then we are going to create a database table:

```
with app.app_context():db.create_all()
```

to exit the shell, we type **exit()**

Alternatively, if we don't want to run the program in the Python terminal, we can also add Python script in 'quotes.py' file to ensure that certain code such as the import data to our database, is only executed when the script is run directly, not when it is imported as a module in another script:

```
if __name__ == '__main__':  
    with app.app_context():  
        db.create_all()  
    app.run()
```

Let's us understand each line of codes:

- `if __name__ == '__main__':`

This condition checks if the script is being run directly. In Python, every module has a special built-in variable called `__name__`. If the module is being run directly (as opposed to being imported), `__name__` is set to `'__main__'`. This check ensures that the code block beneath it is executed only when the script is run directly.

- `with app.app_context():db.create_all()`

This line enters the application context of a Flask application named `app`. In Flask, the application context is used to make certain objects (like `request`, `g`, and others) available globally within a block of code. This is necessary for operations that interact with the Flask application's context, such as database operations.

- `db.create_all()`

This command creates all the database tables defined by SQLAlchemy models. It is usually used to initialize the database schema.

- `app.run()`

This line starts the Flask web server, allowing the application to handle incoming web requests.

After it, we can log in into Postgresql and check if the table 'favquotes' is created.

WSGI, Web Server Gateway Interface

It is a protocol for web servers to forward requests to web applications or frameworks written in the Python.

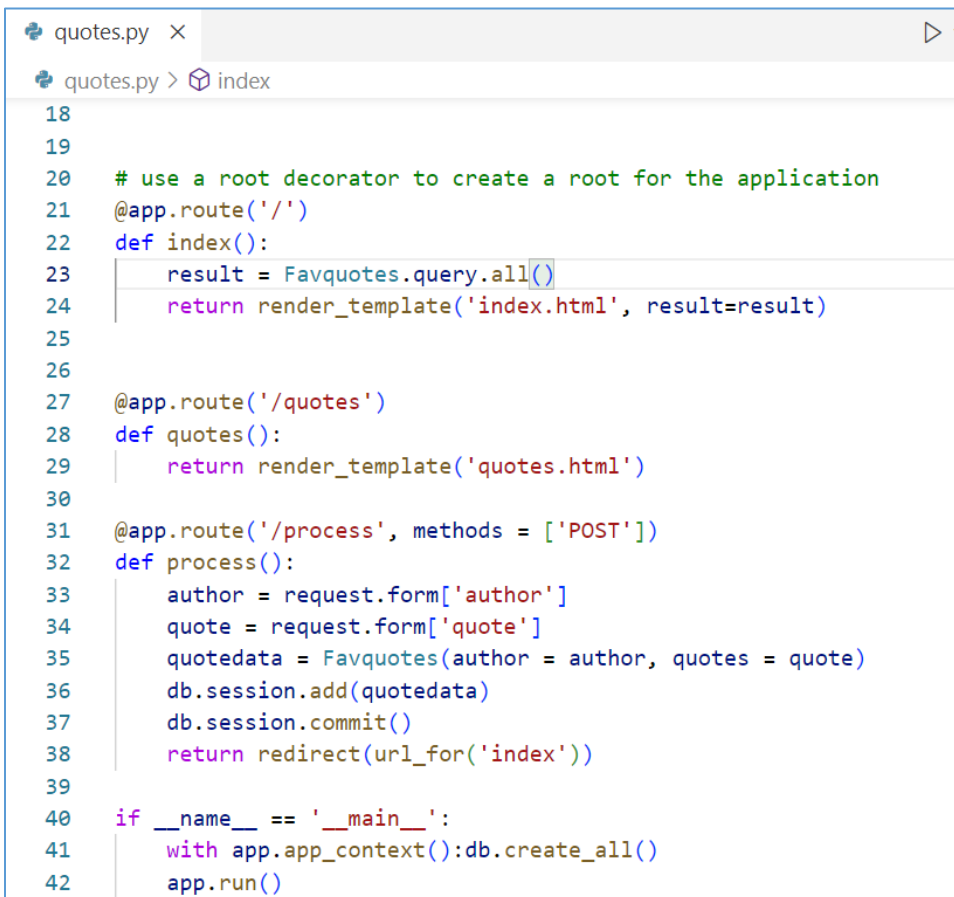
Gunicorn? Gunicorn is a Python WSGI HTTP server that facilitates communication between your web application and the web server:

```
pip install gunicorn
```

Storing form data inside database

```
@app.route('/')
def index():
    result = Favquotes.query.all()
    return render_template('index.html', result=result)
```

After it, we are going to use Jinja to inset the content:



```
quotes.py x
quotes.py > index
18
19
20 # use a root decorator to create a root for the application
21 @app.route('/')
22 def index():
23     result = Favquotes.query.all()
24     return render_template('index.html', result=result)
25
26
27 @app.route('/quotes')
28 def quotes():
29     return render_template('quotes.html')
30
31 @app.route('/process', methods = ['POST'])
32 def process():
33     author = request.form['author']
34     quote = request.form['quote']
35     quotedata = Favquotes(author = author, quotes = quote)
36     db.session.add(quotedata)
37     db.session.commit()
38     return redirect(url_for('index'))
39
40 if __name__ == '__main__':
41     with app.app_context():db.create_all()
42     app.run()
```

In *quotes.html* add the name attributes to textfield as **quote** and the input as **author**

```

<> quotes.html x
templates > <> quotes.html > html > body > main.container > form > div.form-group > input#inputName.form-control
9  <body>
10  <main class="container">
11      <form action="/process" method="post">
12          <div class="form-group">
13              <label for="inputName">Quote Author</label>
14              <input type="text" id="inputName" name="author" class="form-control" placeholder="Add your quote" />
15          </div>
16
17          <div class="form-group">
18              <label for="quote">Quote</label>
19              <textarea class="form-control" id="quote" name="quote" rows="5" placeholder="Add your quote" />
20          </div>
21

```

Insert a for loop to the *index.html*

```

<body>
    <h1>Student's name - Flask</h1>

    <main class="container">
        <nav class="addquotebtn">
            <a href="{{url_for('quotes')}}">Add a quote</a>
        </nav>
        <h2>Favorite quotes</h2>
        <!-- FOR LOOP -->
        {% set list_length = result|length %}
        {% for r in range(list_length) %}
            <div>
                <h3 class="favquotes">Quote {{r}}</h3>
                <p>
                    <em class="favquotes_quote">
                        {{result[r].quotes }}
                    </em>
                    <span class="favquotes_author">{{ result[r].author }}</span>
                </p>
            </div>
        {% endfor %}

    </main>

</body>

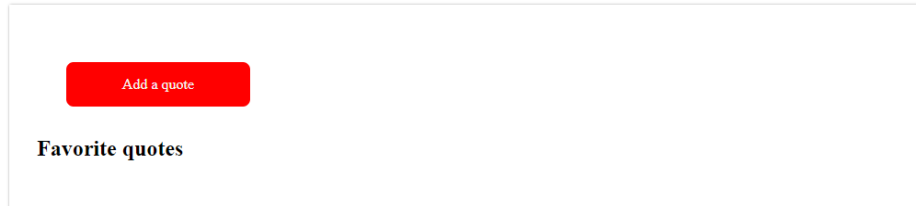
</html>

```

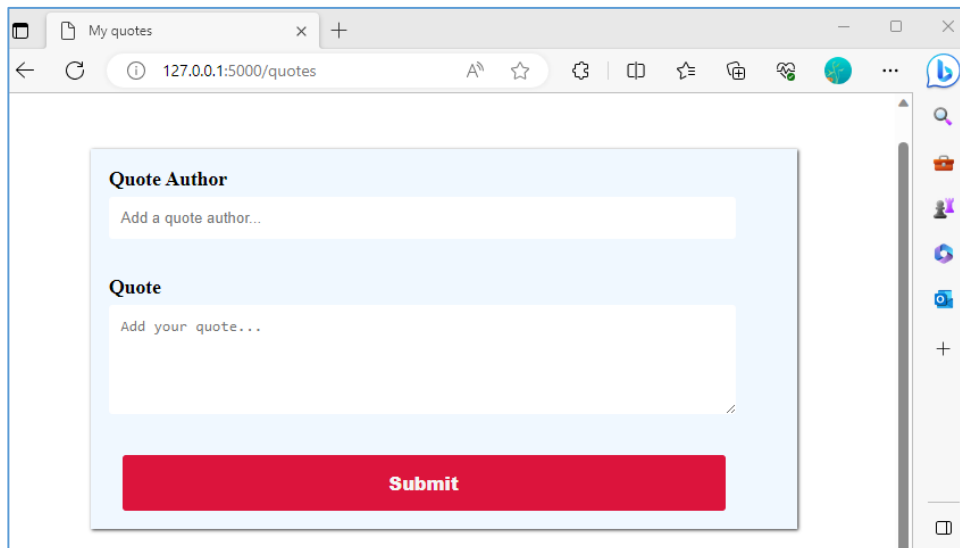
Test the app

To test if the app is transferring data to the database, deactivate and activate the virtual environment again, then run the app again by typing in the terminal 'flask run'.

- After it, go to the localhost and open the app. It should load the index.html webpage



- In the homepage of the app, click on 'Add a quote', which should take you to the 'quotes.html' webpage.



- Type an author's name and the quote, and click on the 'Submit' button.
- Once you clicked on 'Submit' button, it should take you back to the index.html webpage with the entered author and quote displaying in the webpage. It should also transfer the data into the Postgresql.
- Open Postgresql and check for table *favquotes*:

pgAdmin File Object Tools Help

Browser PostgreSQL 14

- Databases (2)
 - demoDB
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - Publications
 - Schemas (1)
 - public
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Operators
 - Procedures
 - Sequences
 - Tables (7)
 - album
 - artist
 - favquotes

public.favquotes/demoDB/postgres@PostgreSQL 14

Query Query History

```

1 SELECT * FROM public.favquotes
2 ORDER BY id ASC

```

Data output Messages Notifications

	id [PK] integer	author character varying (30)	quote character varying (2000)
1	1	Peter	hello wonderland
2	2	Socrates	I cannot teach anybody ...
3	3	Carrie Underwood	Think before he cheats!