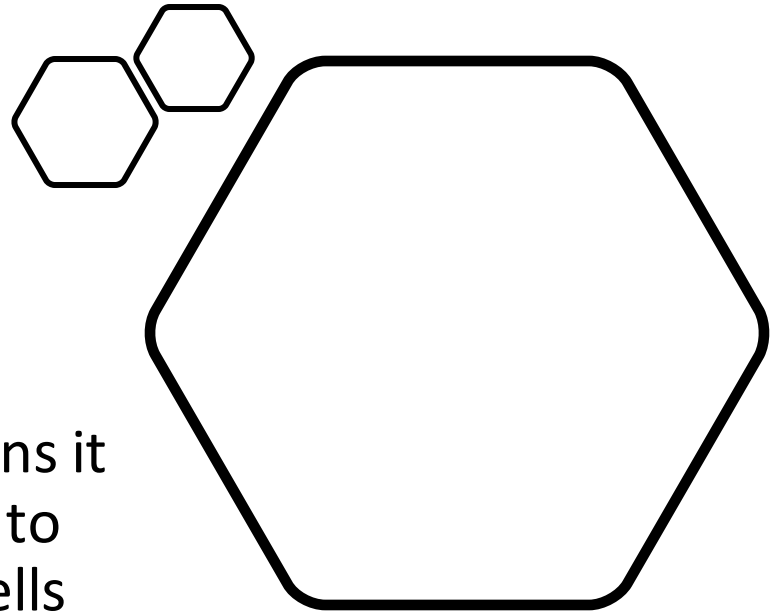
Flask Tutorial

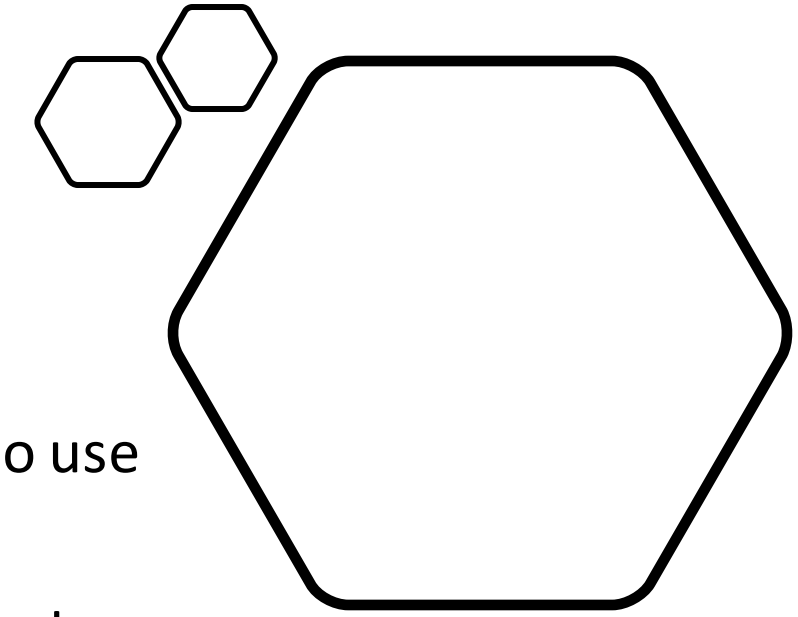
I will show you how to create websites with python using the micro framework flask. Flask is designed for quick development of simple web applications.

What is Flask?



Flask is a known as a **micro web framework**. This means it provides some basic functionality to allow developers to build simple websites. It does not come with all the bells and whistles like some other web frameworks like django have and therefore is typically not used for complex websites. However, there is a benefit to flask's limited features. One of which is it's simplicity. You will see in these tutorial that flask makes it very easy and quick to do things. In fact in under 10 minutes we will have our first website up and running!

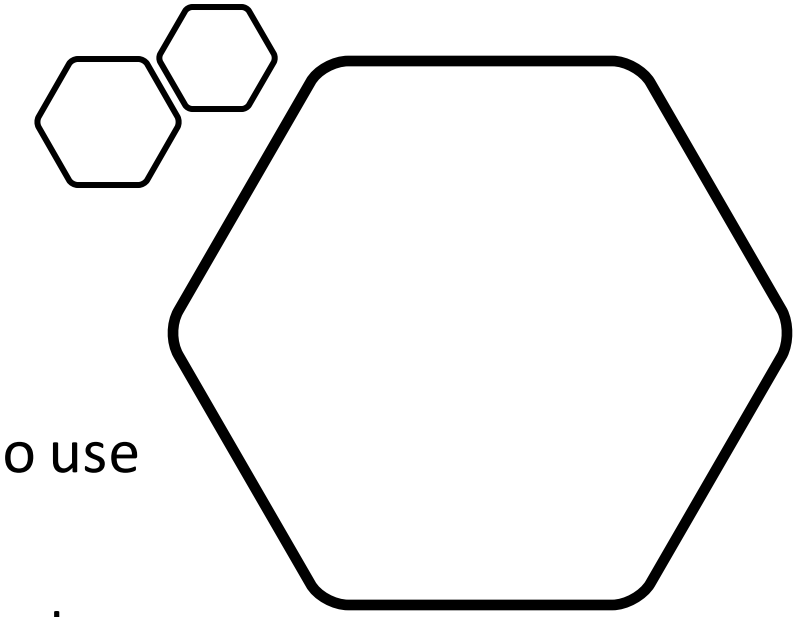
Installing Flask



Installing flask is pretty simple. All that is required is to use the pip command: **pip install flask.**

You can run this command from your command prompt window.

Starting a Project



Installing flask is pretty simple. All that is required is to use the pip command: **pip install flask.**

You can run this command from your command prompt window.

Starting a Project

Creating a website in flask is as easy as **creating a new python script, importing flask and starting the instance.**

I've named my python file ***tutorial1.py*** and put it in it's own folder. You can name yours whatever you'd like.

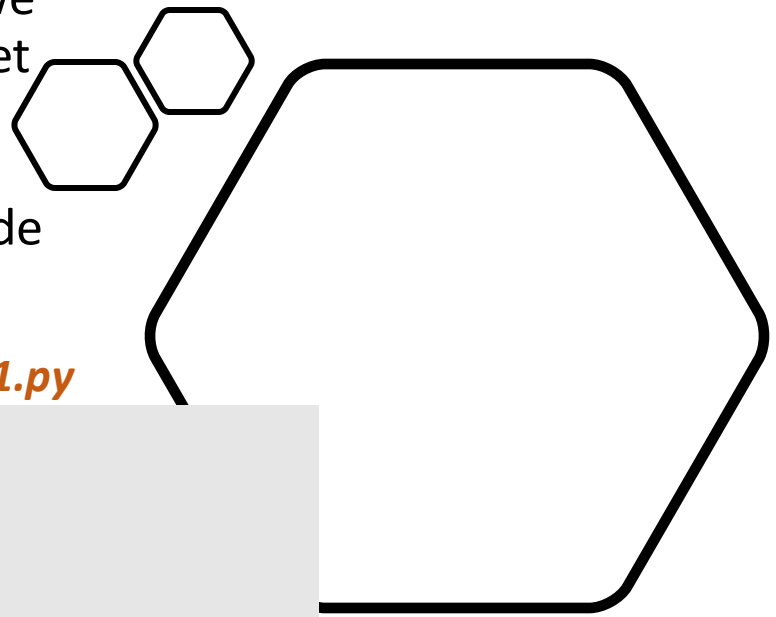
tutorial1.py

```
from flask import Flask

app = Flask(__name__)

if __name__ == "__main__":
    app.run()
```

And now we've created our first flask project!

Creating Web Pages

Time to make our first web page. Flask makes this pretty easy as all we need to do is define a function that will represent each page and set its URL/Path. These functions will perform some operations and return some HTML that should be rendered. For this first video we will just return some very basic HTML that's written as a string inside our python script.
We do this by decorating the function (see below)



tutorial1.py

```
from flask import Flask

app = Flask(__name__)

# Defining the home page of our site
@app.route("/")  # this sets the route to this page
def home():
    return "Hello! this is the main page <h1>HELLO</h1>"  # some basic inline html

if __name__ == "__main__":
    app.run()
```

We've now successfully created our first web page! Time to test it out.

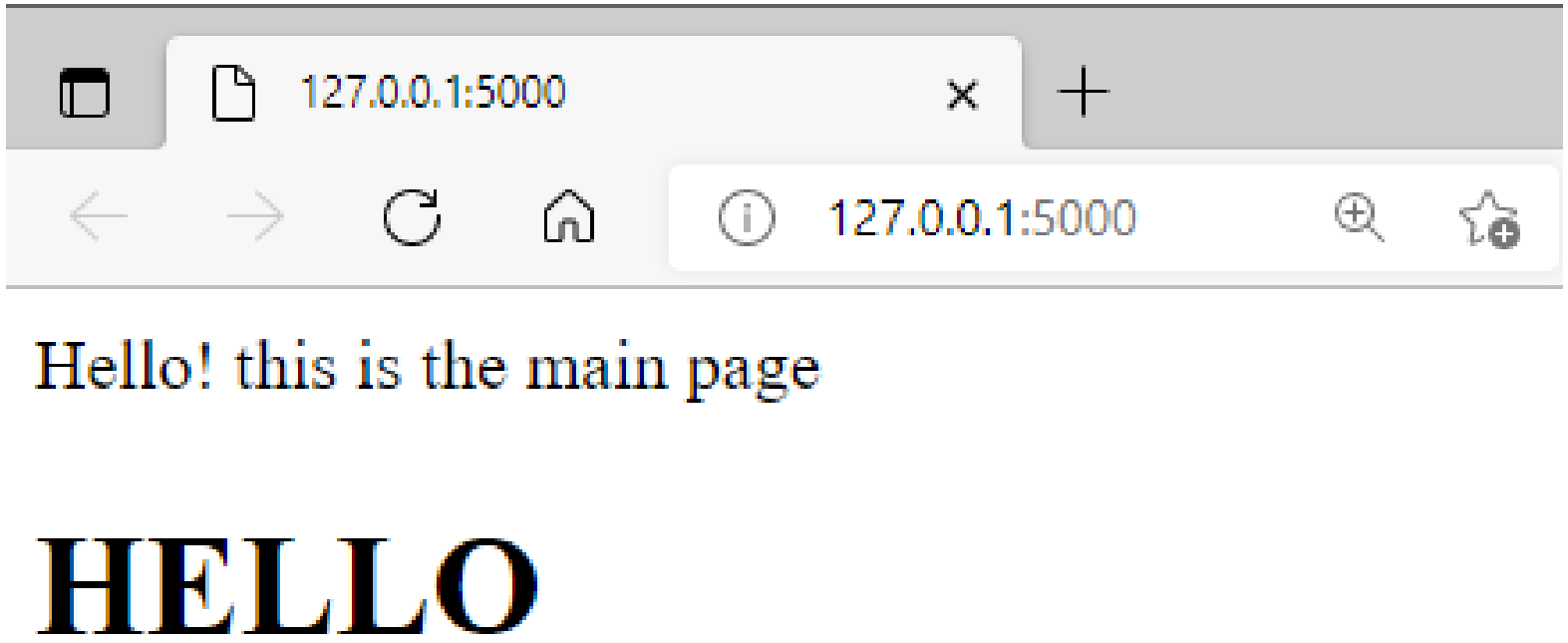
Running Your Website

Time to make our first web page. Flask makes this pretty easy as all we need to do is define a function that will represent each page and set its URL/Path. These functions will perform some operations and return some HTML that should be rendered. For this first video we will just return some very basic HTML that's written as a string inside our python script.
We do this by decorating the function (see below)

```
C:\Users\MarizzaMil\PycharmProjects\FlaskTutorial>python "tutorial1.py"
 * Serving Flask app 'tutorial1' (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)
```

You may notice some warnings when starting your script. Just ignore these for now.

Running Your Website



Once you've connected to your site you should see something like the image above.

Dynamic URLs

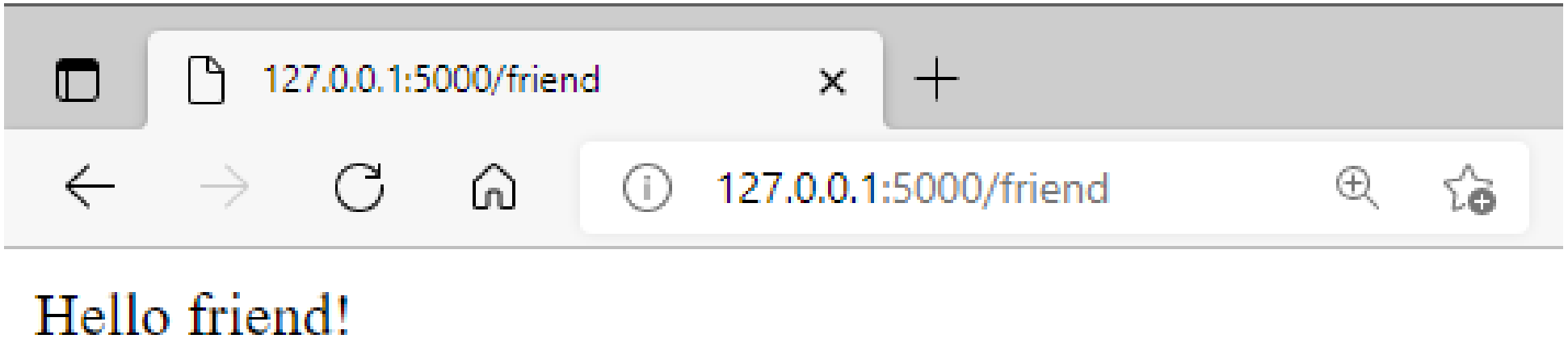
Now let's move to our next web page! This time however we are going to access it using something called a dynamic url/path. This essentially allows us to route any url that matches a specific pattern to a specific web page. This can be useful for things like posts. Where each post may be displayed the exact same way and merely differ in content. By creating a dynamic path to all of our posts we can read the post Id from the address bar and pass that to a function that can display that post. Rather than creating a web page/function for each of our posts.
To define a dynamic path you simply put a variable name inside of . Like this: .

tutorial1.py

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello! this is the main page <h1>HELLO</h1>"

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

if __name__ == "__main__":
    app.run()
```

Dynamic URLs

Have a look at the example above. This will allows us to route any URL to this function, pass the name variable to our function and display "Hello name!" on our web page. Let's give it a try.



Redirects

The last thing to mention is how to redirect users to other pages from within our python code. To do this we will start by importing some more functions from Flask

I'll create another web page called admin to illustrate how to use these functions.

The **url_for** function takes an argument that is the name of the function that you want to redirect to. In this example since we want to redirect to the home page we will put the name of the function for the home page which is "name".

Now whenever we visit /admin we will be redirected home.

tutorial1.py

```
from flask import Flask, redirect, url_for

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello! this is the main page <h1>HELLO</h1>"

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

@app.route("/admin")
def admin():
    return redirect(url_for("home"))

if __name__ == "__main__":
    app.run()
```

HTML Templates

In this flask tutorial I will show you how to render and create HTML templates. I will also discuss how to dynamically create HTML with the use of python code in your html files.

Redirecting Continued

Starting from where we left off in the last tutorial. I wanted to show how to redirect to a function that takes an argument (like our user function). To do this we simply need to define the parameter name and a value in the url_for function, like below.

tutorial1.py

```
from flask import Flask, redirect, url_for

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello! this is the main page <h1>HELLO</h1>"


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

@app.route("/admin")
def admin():
    return redirect(url_for("user", name="Admin!"))
    # Now we when we go to /admin we will redirect to user
with the argument "Admin!"


if __name__ == "__main__":
    app.run()
```

Rendering HTML

Now as beautiful as our website is we probably want to render proper HTML files. To do this we need to follow a few steps.

Step 1: Create new python file called ***tutorial2.py***

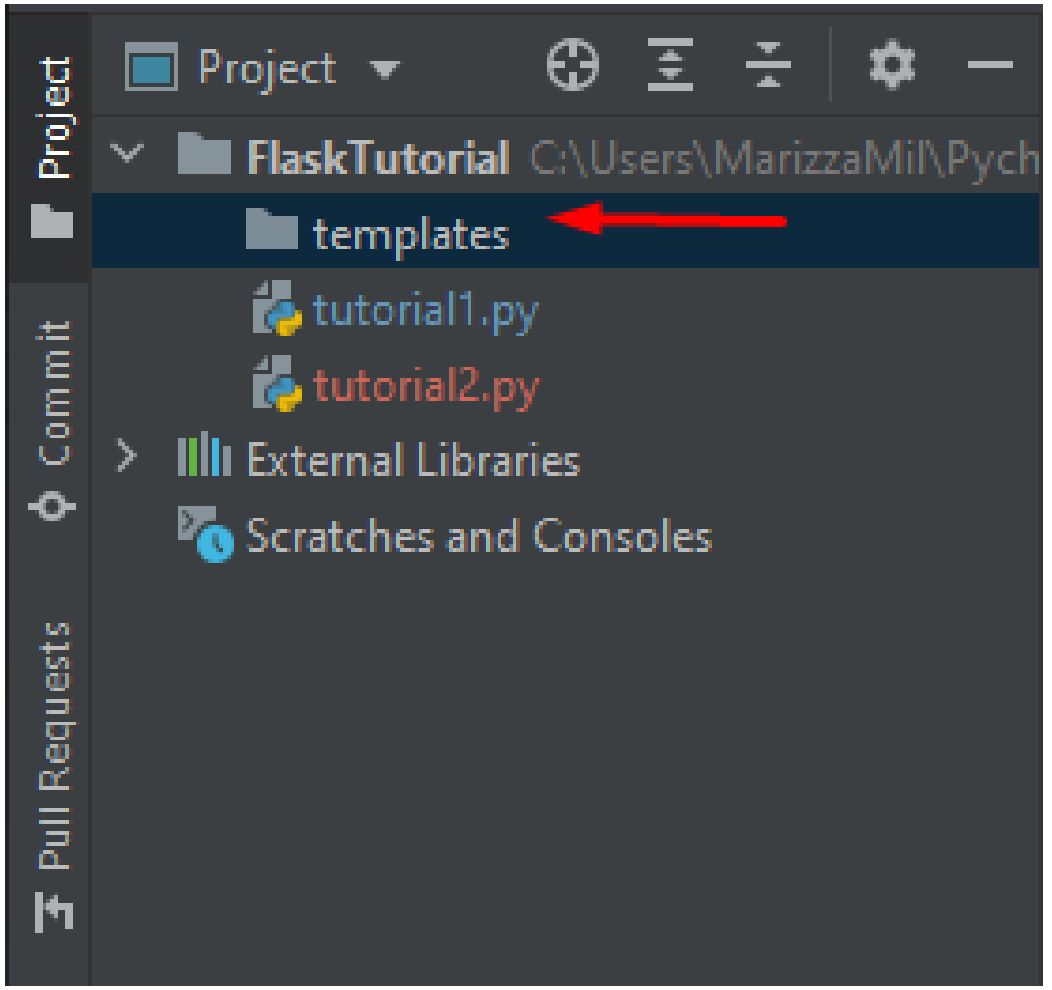
Import the render_template function from flask

tutorial2.py

```
from flask import Flask, render_template


app = Flask(__name__)



@app.route("/")
def home():
    return "Hello! this is the main page <h1>HELLO</h1>"



if __name__ == "__main__":
    app.run()
```

Rendering HTML

Step 2: Create a new folder called *templates* inside the SAME directory as our python script.



Rendering HTML

Step 3: Create an html file, I've named mine *index.html*. Make sure to **put it in the templates folder!**

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Home page</title>
</head>
<body>
   <h1>Home Page!</h1>
</body>
</html>
```

Rendering HTML

Step 4: Render the template from a function in python.

The render_template function will look in the "templates" folder for a file called "index.html" and render it to the screen. Now try running the script and visiting "/". You should see that html rendered.

tutorial2.py

```
from flask import Flask, render_template


app = Flask(__name__)


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


if __name__ == "__main__":
    app.run()
```

Dynamic HTML

Flask uses a templating engine called jinja. This allows you to write python code inside your html files. It also allows you to pass information from your back-end (the python script) to your HTML files.
In your HTML file you can use the following syntax to evaluate python statements.
{{Variable/Statement}} Placing a variable or statement inside of {{}} will tell flask to evaluate the statement inside the brackets and render the text equivalent to it.

Dynamic HTML

Let's look at an example.

Here we are defining that we will have a variable passed to this HTML file called content. So from our back-end, when we render the template we need to pass it a value for content.

index.html

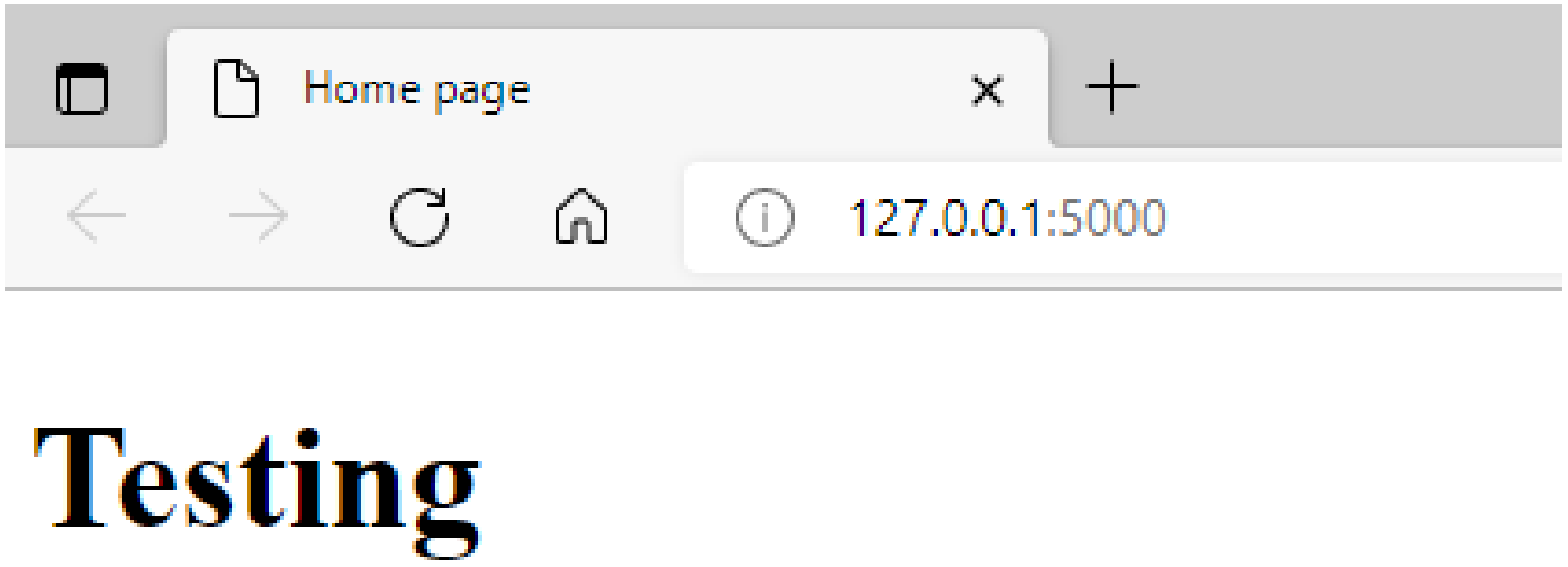
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Home page</title>
</head>

<body>
   <h1>{{content}}</h1>
</body>

</html>
```

tutorial2.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def home():
   return render_template("index.html", content="Testing")

if __name__ == "__main__":
   app.run()
```

Dynamic HTML

When we run the script and navigate to the home page we get the following.



Templates Continued

There are a few other things you can write in your HTML relating to python code. The most popular is to use for loops and if statements.
You can place python expressions inside **{% %}**.

Templates Continued

Here's a quick example of the syntax for if statements

index.html

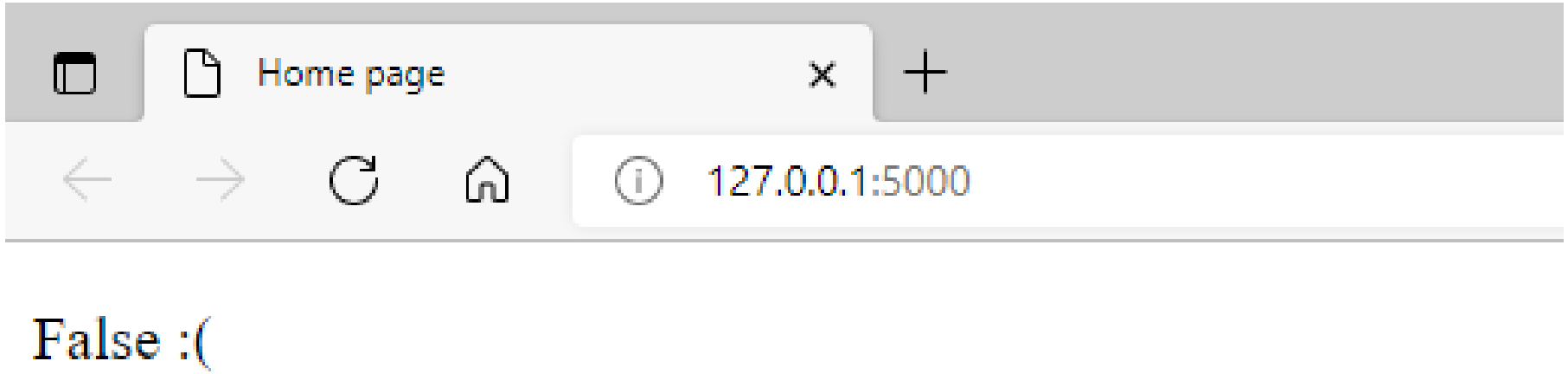
```
<!doctype html>
<html>
<head>
    <title>Home page</title>
    </head>
    <body>
        {% if content == "true" %}
            <p>True!</p>
        {% else %}
            <p>False :(</p>
        {% endif %}
    </body>
</html>
```

tutorial2.py

```
from flask import Flask, render_template


app = Flask(__name__)


@app.route("/")
def home():
   return render_template("index.html", content="Testing")


if __name__ == "__main__":
   app.run()
```

Templates Continued

When we run the script and navigate to the home page we get the following.



Templates Continued

Here's a quick example of the syntax for loops.

index.html

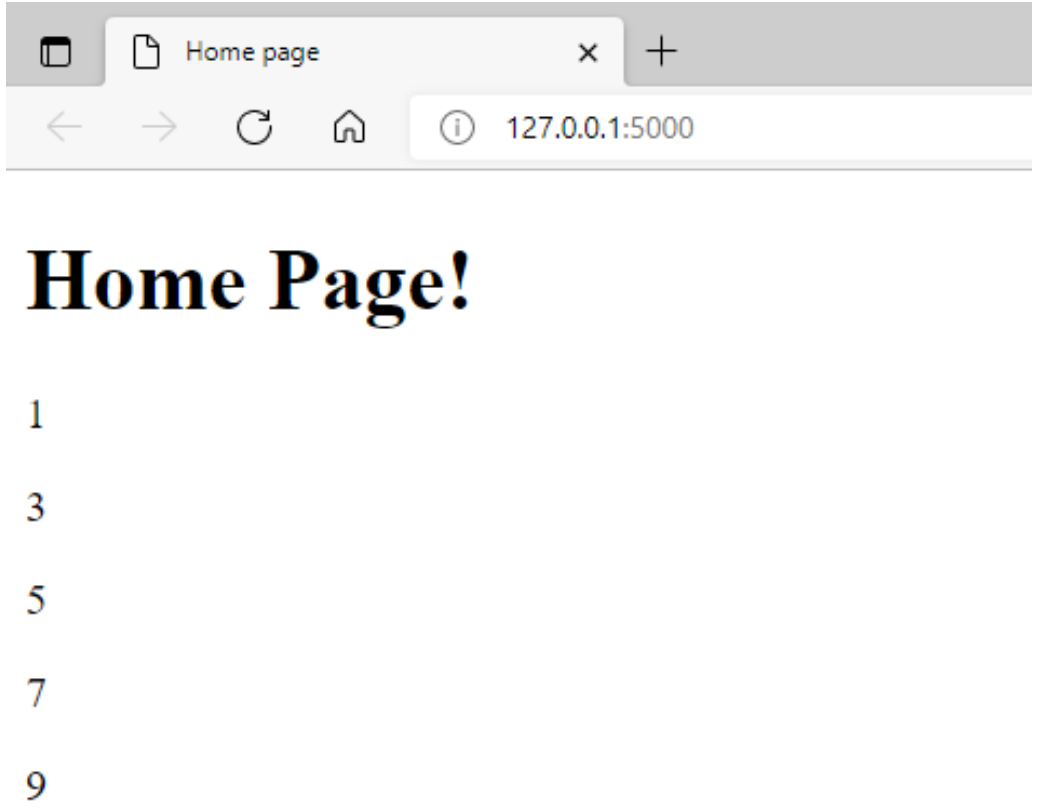
```
<!doctype html>
<html>
<head>
    <title>Home page</title>
    </head>
    <body>
      {% for x in range(10) %}
          {% if x % 2 ==1 %}
             <p>{{x}}</p>
          {% endif %}
       {% endfor %}
    </body>
</html>
```

tutorial2.py

```
from flask import Flask, render_template


app = Flask(__name__)


@app.route("/")
def home():
   return render_template("index.html", content="Testing")


if __name__ == "__main__":
   app.run()
```

Templates Continued

When we run the script and navigate to the home page we get the following.



Templates Continued

Here's a quick example of the syntax for loops.

index.html

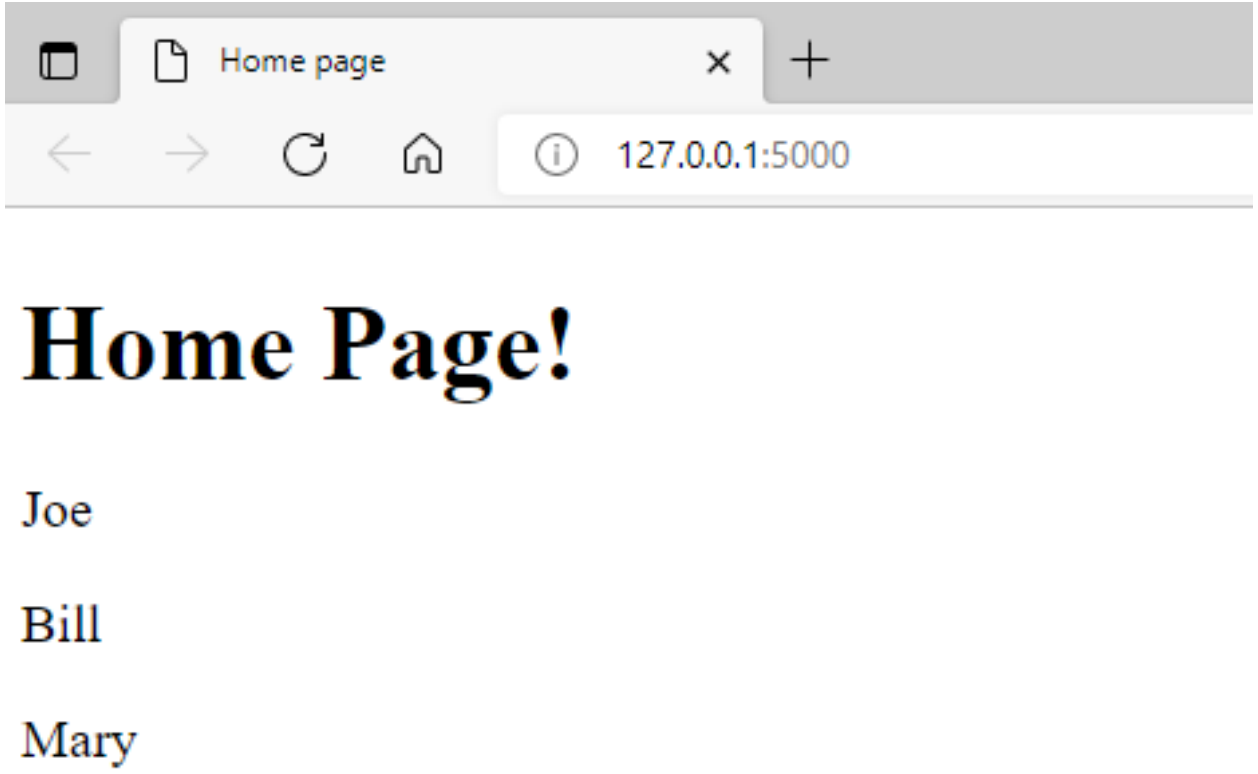
```
<!doctype html>
<html>
<head>
    <title>Home page</title>
    </head>
    <body>
        <h1>Home Page!</h1>
        {% for x in content %}
           <p>{{x}}</p>
        {% endfor %}
    </body>
</html>
```

tutorial2.py

```
from flask import Flask, render_template


app = Flask(__name__)


@app.route("/")
def home():
  return render_template("index.html",
content=['Joe', 'Bill', 'Mary'])

if __name__ == "__main__":
  app.run()
```

Templates Continued

When we run the script and navigate to the home page we get the following.



Passing Multiple Values

Just a quick note here to let you know that you can pass multiple values to your HTML files by defining more keyword arguments in your render_template function or by passing in things like dicts or lists.

Passing Multiple Values

Just a quick note here to let you know that you can pass multiple values to your HTML files by defining more keyword arguments in your render_template function or by passing in things like dicts or lists.

```
@app.route("/")
def home():
    return render_template("index.html", content="Testing", x=4)
```

```
@app.route("/")
def home():
    return render_template("index.html", content={"a":2, "b":"hello"})
```

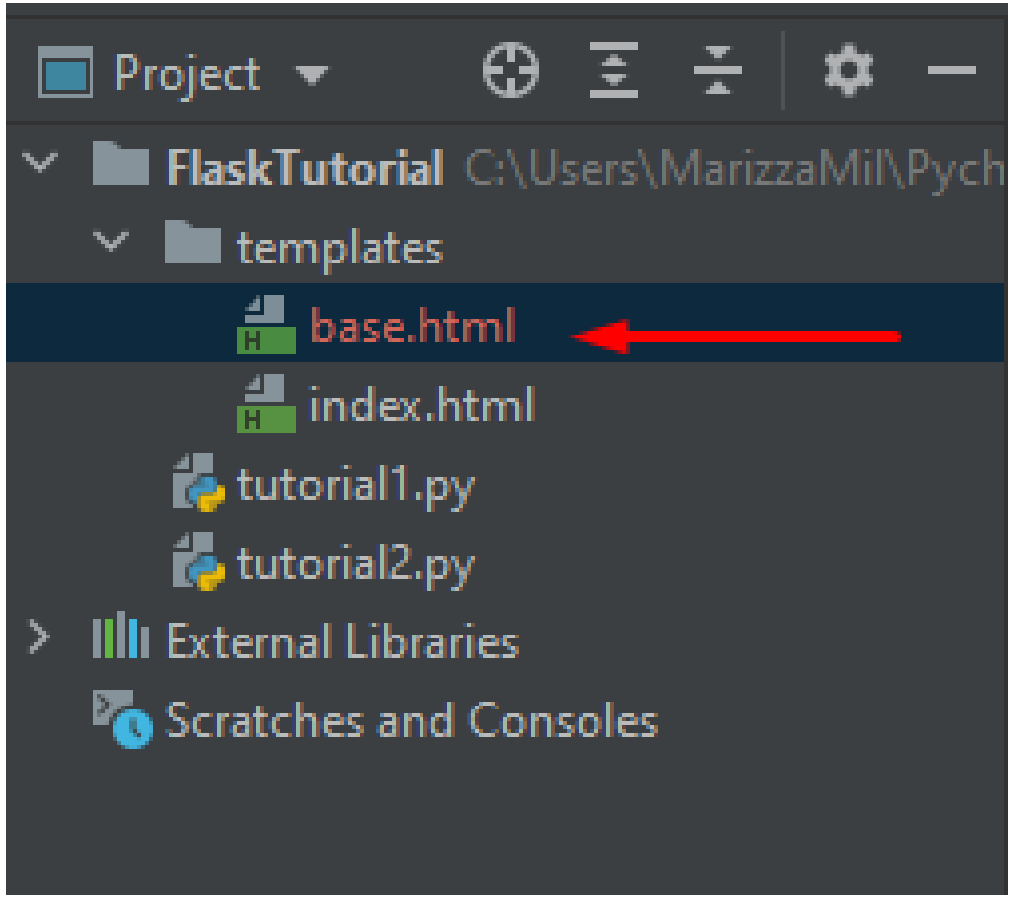
Adding Bootstrap and Template Inheritance

In this flask tutorial I will be showing how to add bootstrap to a flask website and how to use something called template inheritance. This is useful as often times you have the same HTML components that you want to show on all or many different web pages. Template inheritance allows you to make a base template that other templates can inherit from.

Flask

Creating a Base Template

So you may have realized that creating new web pages for every single page on our website is extremely inefficient. Especially when our website follows a theme and has similar elements (like a sidebar) on every page. This is where **template inheritance** comes in. We will talk about how to inherit HTML code later but we are going to start by creating a new HTML document called *"base.html"*. This will hold all of the HTML code that will persist throughout our website. Make sure to put it in the **templates** folder we created previously.



Adding Bootstrap

Bootstrap is a CSS framework that provides some pre-built CSS classes and an easy way to style your website. In the first part of this tutorial I will be showing you how to add this to your website with a few simple steps. We will simply use the CDN (content-delivery-network) to add bootstrap to our website. This involves copying some code from the bootstrap website and adding it in some specific places in our HTML.

Adding Bootstrap

Step 1: Add the following code at the beginning of the **head tags** in the "base.html" file.

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
```

Adding Bootstrap

Step 2: Add the following code at the beginning of the **head tags** in the "base.html" file.

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
```

Adding Bootstrap

This is an example of what your code should look like, directly from bootstrap

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
  </body>
</html>
```

Template Blocks

Now that we've created our base template it is time to talk about template inheritance. Flask uses a templating engine called "Jinja" which allows us to inherit parent templates and override specific parts of them. To define what can be overwritten we need to create something in the base template called **blocks**.
The syntax for a block looks like this.

```
{% block name_of_block %}
{% endblock %}
```

When we write this inside our html we are defining that this area of the code can be overwritten from a child template using the same block name. We can use more than one block in our templates.

Template Blocks

I've created two blocks in my "base.html" template.

base.html

```
<!doctype html>
<html>
<head>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    {% block content %}
    {% endblock %}
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
</body>
</html>
```

Inheriting Templates

Now that we have defined our blocks it's time to inherit them from our child templates.
To do this we will write an extends tag at the begging of our file and define the blocks we want to overwrite. Placing html code inside the blocks will allow us to show unique information on each child template.
Here's an example of another html file I created called ***"index.html"***. It's placed in the same folder as my "base.html" file.

index.html

```
{% extends "base.html" %}
{% block title %}Home Page{% endblock %}
{% block content %}
<h1>Test</h1>
{% endblock %}
```

Adding a Sidebar

Since we added bootstrap to our website let's start using it! I'm going to add a simple sibebar to our base template that will show up on all of our pages. To do this I've merely stole some code from the bootstrap documentation and pasted in in the new HTML file called ***navbar.htm,*** which placed in the same folder as my "base.html" file.

navbar.html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-
expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item active">
          <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Features</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Pricing</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
        </li>
      </ul>
    </div>
  </nav>
```

Adding a Sidebar

And here's my "base.html" template.

base.html

```
<!doctype html>
<html>
<head>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
    <title>{% block title %}{% endblock %}</title>
</head>
<body>

    {% include "navbar.html" %}
    {% block content %}
    {% endblock %}

    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
</body>
</html>
```

Rendering the Page

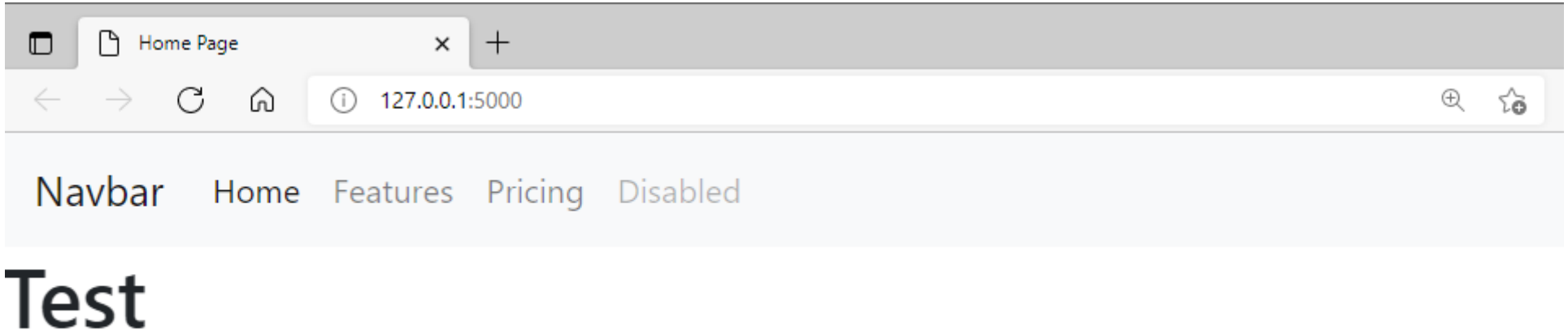
Now it's time to display our website! For this specific tutorial I'll just render the "index.html" template to show you that everything is working.

tutorial3.py

```
from flask import Flask, redirect, url_for, render_template


app = Flask(__name__)


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



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

Rendering the Page

Now when we head to "/" we see the following!



HTTP Methods (GET/POST) & Retrieving Form Data

In this flask tutorial I show you how to use the HTTP request methods Post and Get. The POST method will allow us to retrieve data from forms on our web page. In later videos we'll get into more advanced topics relating to login sessions and using POST methods to retrieve secure information like passwords.

Flask

HTTP Methods

In this tutorial we will talk about HTTP methods. HTTP methods are the standard way of sending information to and from a web server. To break it down, a website runs on a server or multiple servers and simple returns information to a client (web-browser). Information is exchanged between the client and the server using an HTTP protocol that has a few different methods. We will be discussing the post commonly used ones called POST & GET.

GET

GET is the most commonly used HTTP method and it is typically used to retrieve information from a web server.

POST

POST is commonly used to send information to web server. It is more often used when uploading a file, getting form data and sending sensitive data. POST is a secure way to send data to a web server.

Creating a Form

For this example will create a form and discuss how we can use the POST HTTP method to send the information in our form to our server.
Here is a new html file I created called "login.html".

login.html

```
{% extends "base.html" %}
{% block title %}Login Page{% endblock %}

{% block content %}
<form action="#" method="post">
    <p>Name:</p>
    <p><input type="text" name="nm" /></p>
    <p><input type="submit" value="submit"/></p>
</form>
{% endblock %}
```

We've specified the method of the form to be post. Which means that when the form is submitted we will POST data to the web server in the form of a request.
Note the name of the text input field, we will use this to get it's value from our python code.

Back-End

Create new python file called ***tutorial4.py***

What we're going to do is setup a login page that asks the user to type in their name. Once they hit the submit button they will be redirected to a page that displays the name they typed in.

We'll start by importing a new module called "request".

```
from flask import Flask, redirect, url_for, render_template, request
```

Next we'll setup the login page.

To specify that a page works with both POST and GET requests we need to add a method argument to the decorator.

```
@app.route("/login", methods=["POST", "GET"])
```

Back-End

Now from inside the function we'll check if we are receiving a GET or POST request. We'll then define the operation to perform based on the request. If we have received a POST request that means we submitted the form and should redirect to the appropriate page. Otherwise we should simply return and render the login page.

```
def login():
    if request.method == "POST":
    user = request.form["nm"]
    return redirect(url_for("user", usr=user))
    else:
    return render_template("login.html")
```

Now we'll setup the user page. This will simply display the users name in h1 tags. Just for example/testing purposes, we don't need anything fancy.

```
@app.route("/<usr>")
def user(usr):
    return f"<h1>{usr}</h1>"
```

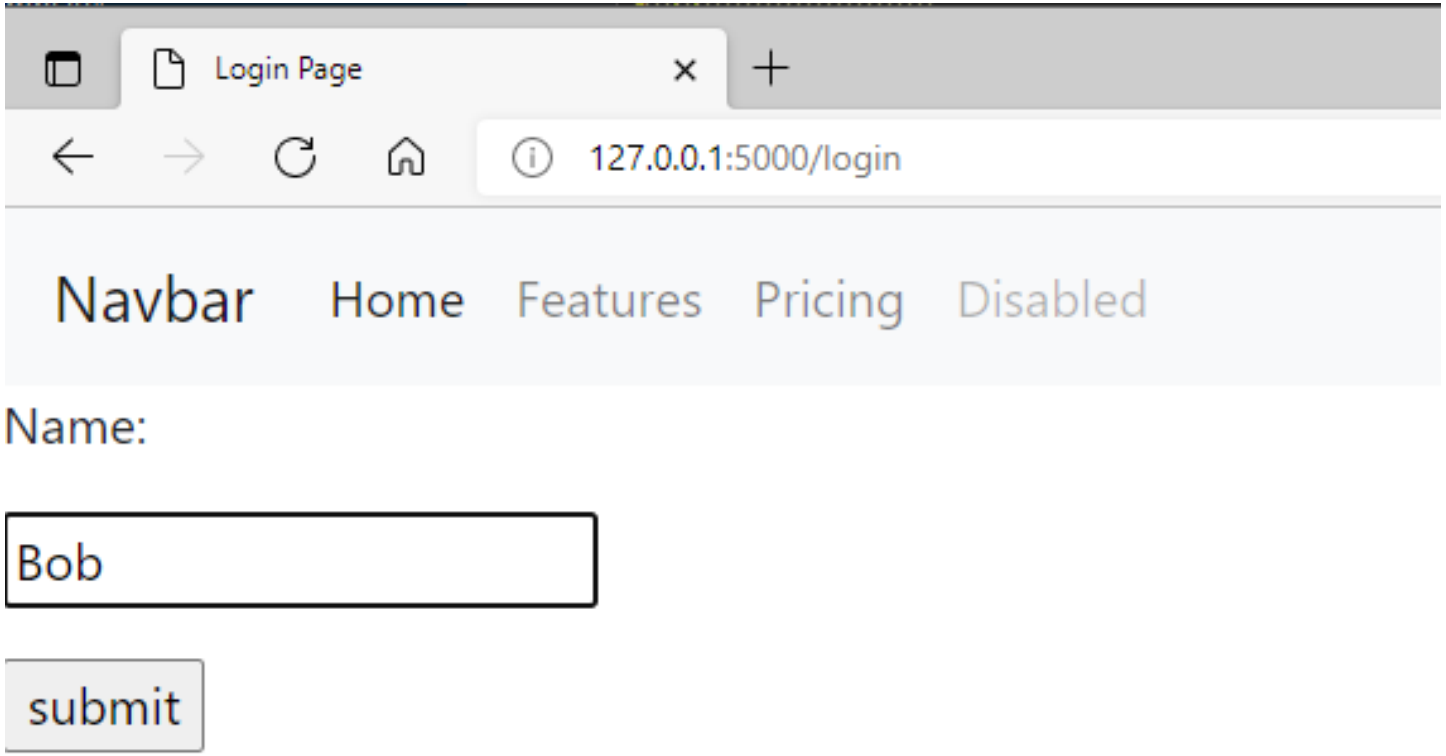
Full Python Code

tutorial4.py

```
from flask import Flask, redirect, url_for, render_template, request

app = Flask(__name__)

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

@app.route("/login", methods=["POST", "GET"])
def login():
   if request.method == "POST":
      user = request.form["nm"]
      return redirect(url_for("user", usr=user))
   else:
      return render_template("login.html")

@app.route("/<usr>")
def user(usr):
   return f"<h1>{usr}</h1>"

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

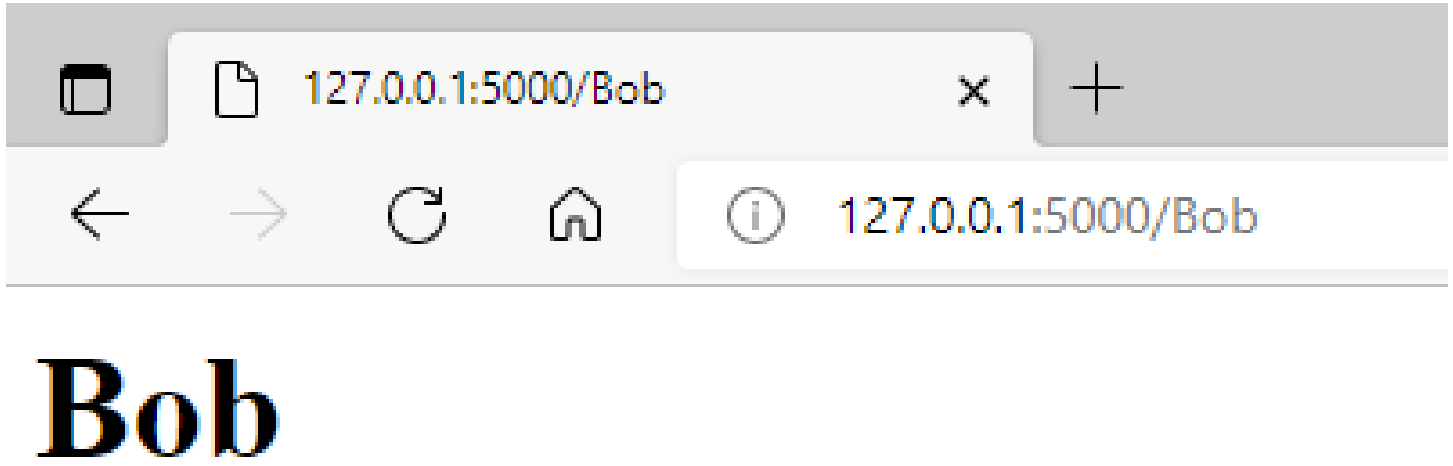
Now let's test out our site!

We'll head to the /login page and type our name.



Now let's test out our site!

And after hitting submit! We get a page with our name on it...



Sessions

In this flask tutorial I show you how to use the HTTP request methods Post and Get. The POST method will allow us to retrieve data from forms on our web page. In later videos we'll get into more advanced topics relating to login sessions and using POST methods to retrieve secure information like passwords.

Sessions vs Cookies

You may have heard of a web feature called *cookies*. I'd just like to quickly explain the difference between a cookie and a session to clear up any confusion.

Cookie: Is stored client side (locally in the web browser) and is NOT a secure way to store sensitive information like passwords. It is often used to store things like where a user left on a page, or their username so it can be auto-filled in the next time they visit the page. It is technically possible for someone to modify cookie data.

Session: Is stored server side (on the web server) in a temporary directory. It is encrypted information and is a secure way to store information. Sessions are often used to store information that should not be seen by the user or should not be tampered with.

Setting up a Session

Create new python file called ***tutorial5.py***

tutorial5.py

```
from flask import Flask, redirect, url_for, render_template, request

app = Flask(__name__)

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

@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        user = request.form["nm"]
        return redirect(url_for("user", usr=user))
    else:
        return render_template("login.html")

@app.route("/<usr>")
def user(usr):
    return f"<h1>{usr}</h1>"

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

Setting up a Session

To illustrate how a session works we will walk through a basic example where a user logs in and their username will be stored in a session until they logout.

We will start by importing *session* from flask.

```
from flask import Flask, redirect, url_for, render_template, request, session
```

While we are at it we will import another module that we'll use later.

```
from datetime import timedelta
```

Since the session information will be encrypted on the server we need to provide flask with a *secret key* that it can use to encrypt the data.

At the top of our program we'll write the following:

```
app.secret_key = "hello"
```

Now we can start saving data!

Creating Session Data

Saving information to a session is actually pretty easy. Sessions are represented in python as dictionaries. This means we can access values using **keys**. To save a new value in a session just create a new dictionary key and assign it a value.

```
session["my key"] = "my value"
```

Getting information can then be done the following way:

```
if "my key" in session:
    my_value = session["my key"]
```

For this example we will save a user's username after they login. Then the next time they visit the login page we will see if they are logged in by checking their session data. If they are there is no need for them to login again and we can redirect them to the user page.

```
@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        session.permanent = True
        user = request.form["nm"]
        session["user"] = user
        return redirect(url_for("user"))
    else:
        if "user" in session:
            return redirect(url_for("user"))

        return render_template("login.html")
```

Creating Session Data

Now from our /user page we can display the users name by simply grabbing the information from the session. If they have not signed in yet we will see that they have no username in their session and we can redirect them to the login page.

```
@app.route("/user")
def user():
    if "user" in session:
        user = session["user"]
        return f"<h1>{user}</h1>"
    else:
        return redirect(url_for("login"))
```

And finally time to code the logout page! When a user goes to /logout we need to clear their session data. To do this we can use a method called *session.pop("key", What to do if key doesn't exist)*. The pop method will try to remove and return the key from the session data and will return the second argument if that key doesn't exist. In our case we try to remove the key "user" and if it doesn't exist we will return None.

```
@app.route("/logout")
def logout():
    session.pop("user", None)
    return redirect(url_for("login"))
```

Session Duration

So now that we know how to create, add and remove data from sessions we should probably talk about how long they last. By default a session lasts as long as your browser is open. However, there is a way to change that from flask. We can set the duration of a session by creating a **permanent session**. Creating a permanent session allows us to define how long that session lasts. The default duration of a permanent session is 30 days.

We will start by defining the duration at the beginning of our program.

```
app.permanent_session_lifetime = timedelta(minutes=5)
```

In our example we've made our session last 5 minutes.

Next we will make the users session permanent as soon as they log in.

```
@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        session.permanent = True
        user = request.form["nm"]
        session["user"] = user
        return redirect(url_for("user"))
    else:
        if "user" in session:
            return redirect(url_for("user"))

        return render_template("login.html")
```

Session Duration

So now that we know how to create, add and remove data from sessions we should probably talk about how long they last. By default a session lasts as long as your browser is open. However, there is a way to change that from flask. We can set the duration of a session by creating a **permanent session**. Creating a permanent session allows us to define how long that session lasts. The default duration of a permanent session is 30 days.

We will start by defining the duration at the beginning of our program.

```
app.permanent_session_lifetime = timedelta(minutes=5)
```

In our example we've made our session last 5 minutes.

Next we will make the users session permanent as soon as they log in.

```
@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        session.permanent = True
        user = request.form["nm"]
        session["user"] = user
        return redirect(url_for("user"))
    else:
        if "user" in session:
            return redirect(url_for("user"))

        return render_template("login.html")
```

Full Python Code

tutorial5.py

```
from flask import Flask, redirect, url_for, render_template, request, session
from datetime import timedelta

app = Flask(__name__)
app.secret_key = "hello"
app.permanent_session_lifetime = timedelta(minutes=5)

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

@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        session.permanent = True
        user = request.form["nm"]
        session["user"] = user
        return redirect(url_for("user"))
    else:
        if "user" in session:
            return redirect(url_for("user"))

        return render_template("login.html")

@app.route("/user")
def user():
    if "user" in session:
        user = session["user"]
        return f"<h1>{user}</h1>"
    else:
        return redirect(url_for("login"))

@app.route("/logout")
def logout():
    session.pop("user", None)
    return redirect(url_for("login"))

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