

Welcome to lecture **2**:

Learning flask from

CS50 **web**

programming with

python and javascript

By [Amirhossein Safari](#)
Telegram: @AmirhosseinSFR

Following times are times in the video that the master shows his code or run it:

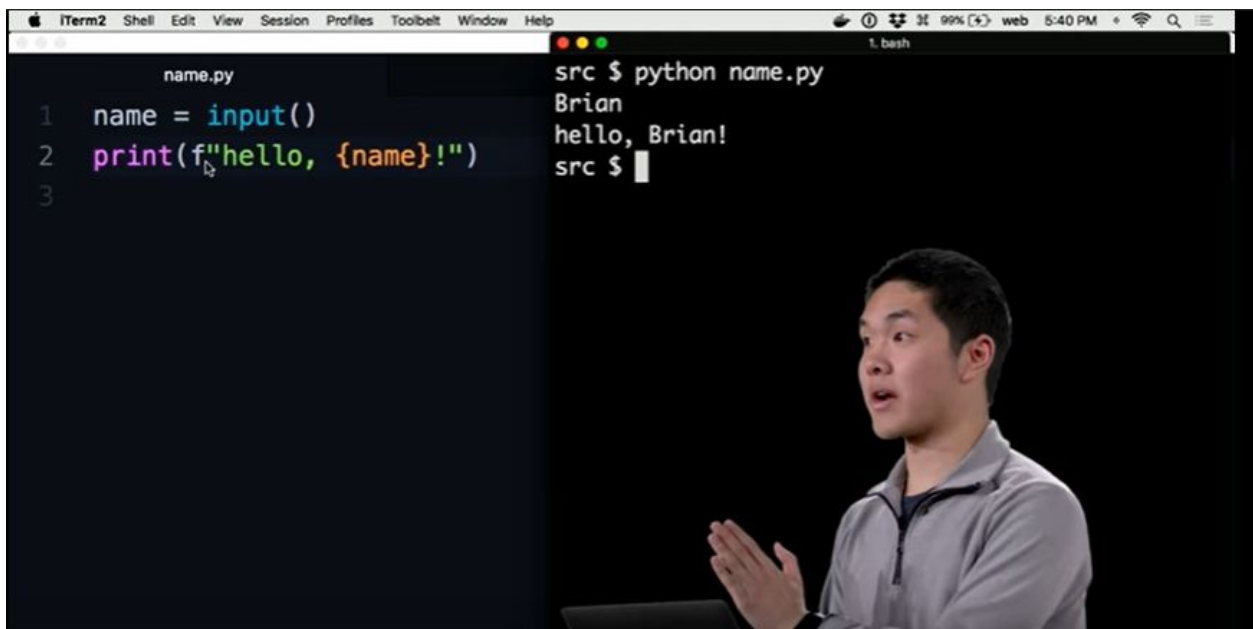
2:15	1:13:23
2:35	1:14:49
5:44 format string == f	1:15:12
9:19	1:16:28
12:09	1:16:34
12:20	1:18:32
12:26	1:19:50
14:50	1:20:58
15:34	1:23:36
16:05	1:23:42
17:36	1:30:29
17:59	1:35:26
19:32	1:36:07
20:02	1:40:56
21:16	1:44:47
23:51 showing data but not repeated data in a list	
24:40	
26:06 Mapping keys to values(data to another one)	
27:19	
29:35	
30:27 and we should define functions first in Python	
31:22	
32:44	
33:23	
34:38 fixed the bug in 33:23	
34:59	
36:12	
38:29	
38:22	
42:42	
45:30	
49:55	
50:59	
52:56	

54:04
54:57
57:39
58:51
1:05:15
1:06:15
1:06:27
1:12:14
1:12:29

Ok, and some summary:)

This is some basics of [python](#) language:

1. Getting inputs and set it in a format string by {}



```
name.py
1 name = input()
2 print(f"hello, {name}!")
3

src $ python name.py
Brian
hello, Brian!
src $
```

2.How to work with arrays and list items and error of indexing out of range(IndexError)

Sequences

- Strings:

```
name = "Alice"  
print(name[0])
```

- Strings are justs sequence of characters, and can be indexed as such.

- Tuples:

```
coordinates = (10.0, 20.0)  
print(coordinates[1])
```

- Tuples are immutable collections of values under a single name, which can be indexed positionally.

- Lists:

```
names = ["Alice", "Bob", "Charlie"]  
print(names[2])
```

- Lists are mutable collections of values under a single name, which can be indexed positionally.
- Indexing out of range raises a Python 'exception'. In this case, an `IndexError`, because there is no fourth value in `names` for Python to return.

Activate Windows

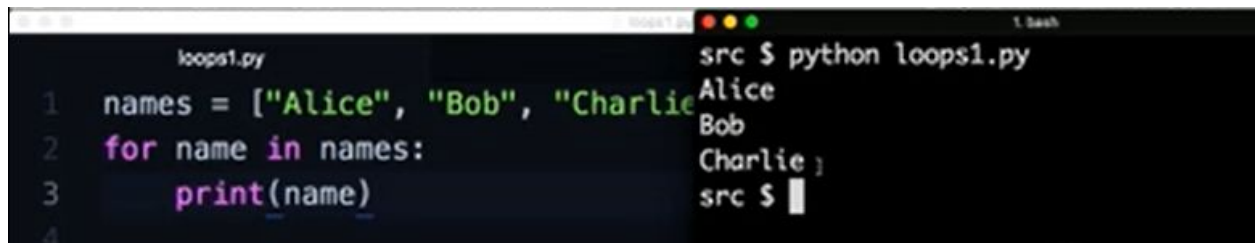
```
>>> names = ["Alice", "Bob", "Charlie"]
>>> names[0]
'Alice'
>>> names[1]
'Bob'
>>> names[2]
'Charlie'
>>> names[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> █
```

3.python interpreter

You can run codes in python interpreter by enter “python” command in terminal and write the code

```
>>> x = 28
>>> x
28
>>> if x > 0:
...     print("x is positive")
...
x is positive
>>> █
```

4. For loops in Arrays



The image shows a code editor window with a file named `loops1.py` and a terminal window. The code in the editor is a for loop that iterates over a list of names and prints each name. The terminal shows the command `python loops1.py` being executed, and the output is the names Alice, Bob, and Charlie, each on a new line.

```
1 names = ["Alice", "Bob", "Charlie"]
2 for name in names:
3     print(name)
```

```
src $ python loops1.py
Alice
Bob
Charlie
src $
```

5. set()

The way we can remove repeated items



The image shows a code editor window with a file named `sets.py`. The code creates a set `s` and adds several elements to it, including a repeated value. The code is as follows:

```
1 s = set()
2 s.add(1)
3 s.add(3)
4 s.add(5)
5 s.add(3)
6 print(s)
7
```

```
src $ python sets.py
{1, 3, 5}
src $
```

6.Mapping keys to values

```
dictionary.py
1 ages = {"Alice": 22, "Bob": 27}
2 ages["Charlie"] = 30
3 ages["Alice"] += 1
4
5 print(ages)
6
```

```
src $ python dictionaries.py
{'Alice': 23, 'Bob': 27, 'Charlie': 30}
src $
```

7.functions

```
functions.py
def square(x):
    return x * x

for i in range(10):
    print("{} squared is {}".format(i, square(i)))
```

```
src $ python functions.py
0 squared is 0
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
6 squared is 36
7 squared is 49
8 squared is 64
9 squared is 81
src $
```

Note: functions should define before where we call them in python and if we don't...

```
src $ python functions.py
Traceback (most recent call last):
  File "functions.py", line 2, in <module>
    print("{} squared is {}".format(i, square(i)))
NameError: name 'square' is not defined
src $
```

8.Importing functions from other file


```
modules.py      functions.py
1  from functions import square
2
3  print(square(10))
4
```

```
src $ python modules.py
0 squared is 0
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
6 squared is 36
7 squared is 49
8 squared is 64
9 squared is 81
100
src $
```

Wait...

There is a bug here, we just expect 100 here, what are the other numbers?

Well, we should put the for loop in a function to don't let it execute when we import the file in other file

So we have main() function!

```
modules.py      functions.py
1  def square(x):
2      return x * x
3
4  def main():
5      for i in range(10):
6          print("{} squared is {}".format(i, square(i)))
7
8  if __name__ == "__main__":
9      main()
10
```

Note: If statement just call the main() function, if we execute this file(functions.py here)

```
src $ python modules.py
100
src $
```

So bug fixed:))

9.Classes

```
classes.py
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6  p = Point(3, 5)
7  print(p.x)
8  print(p.y)
9
```

Note: `__init__` function execute when we make an instance of class in this way

And here you go...

Flask!

1.simple hello world

```
application.py
1  from flask import Flask
2  |
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      return "Hello, world!"
8
```

2.How to run flask app

```
first $ export FLASK_APP=application.py
first $
```

And then “flask run”

3.Dynamic hello!!

Using url parameters

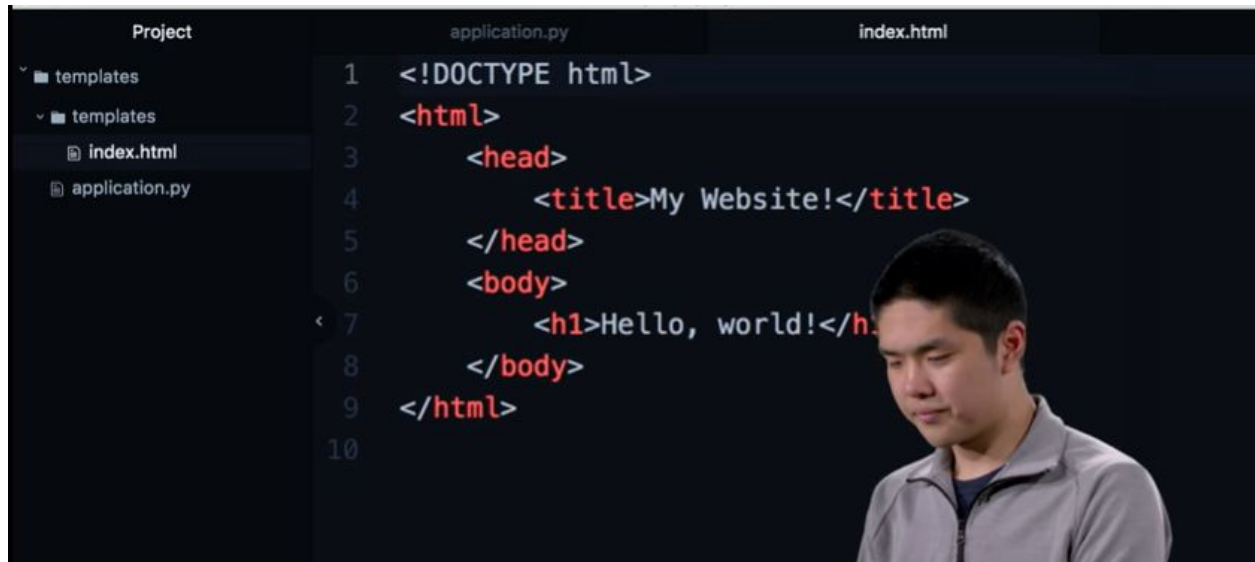
```
application.py
1 from flask import Flask
2     I
3 app = Flask(__name__)
4
5 @app.route("/")
6 def index():
7     return "Hello, world!"
8
9 @app.route("/<string:name>")
10 def hello(name):
11     return f"Hello, {name}!"
12
```

4.Flask can return html tags

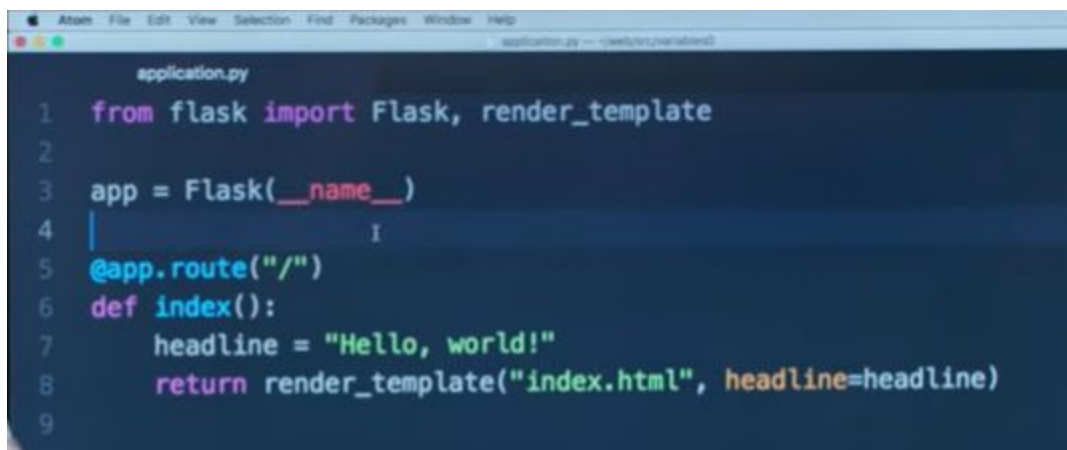
```
application.py
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      return "Hello, world!"
8
9  @app.route("/<string:name>")
10 def hello(name):
11     name = name.capitalize()
12     return f"<h1>Hello, {name}!</h1>"
13
```

5.Rendering html files

```
application.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
```



6. Sending data in html file from flask




```
application.py  index.html
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>My Website!</title>
5      </head>
6      <body>
7          <h1>{{ headline }}</h1>
8      </body>
9  </html>
10
```

Dynamic variable inside {{}}

7. Using date parameters in flask

```
application.py
1  import datetime
2
3  from flask import Flask, render_template
4
5  app = Flask(__name__)
6
7  @app.route("/")
8  def index():
9      now = datetime.datetime.now()
10     new_year = now.month == 1 and now.day == 1
11     return render_template("index.html", new_year=new_year)
12
```

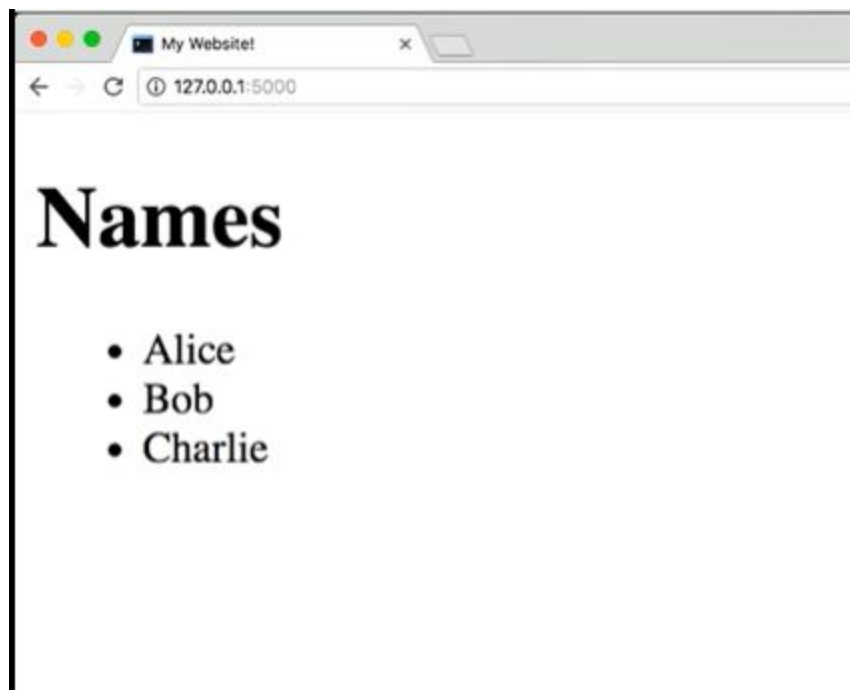
```
index.html -- ~/web/src/conditions
application.py  index.html
6         body {
7             text-align: center;
8         }
9     </style>
10 </head>
11 <body>
12     {% if new_year %}
13         <h1>Yes! Happy New Year!</h1>
14     {% else %}
15         <h1>NO</h1>
16     {% endif %}
17 </body>
18 </html>
```

Note: new_year is a Boolean

8. passing an array to html file

```
application.py -- ~/web/src/loops
application.py
1 from flask import Flask, render_template
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def index():
7     names = ["Alice", "Bob", "Charlie"]
8     return render_template("index.html", names=names)
```

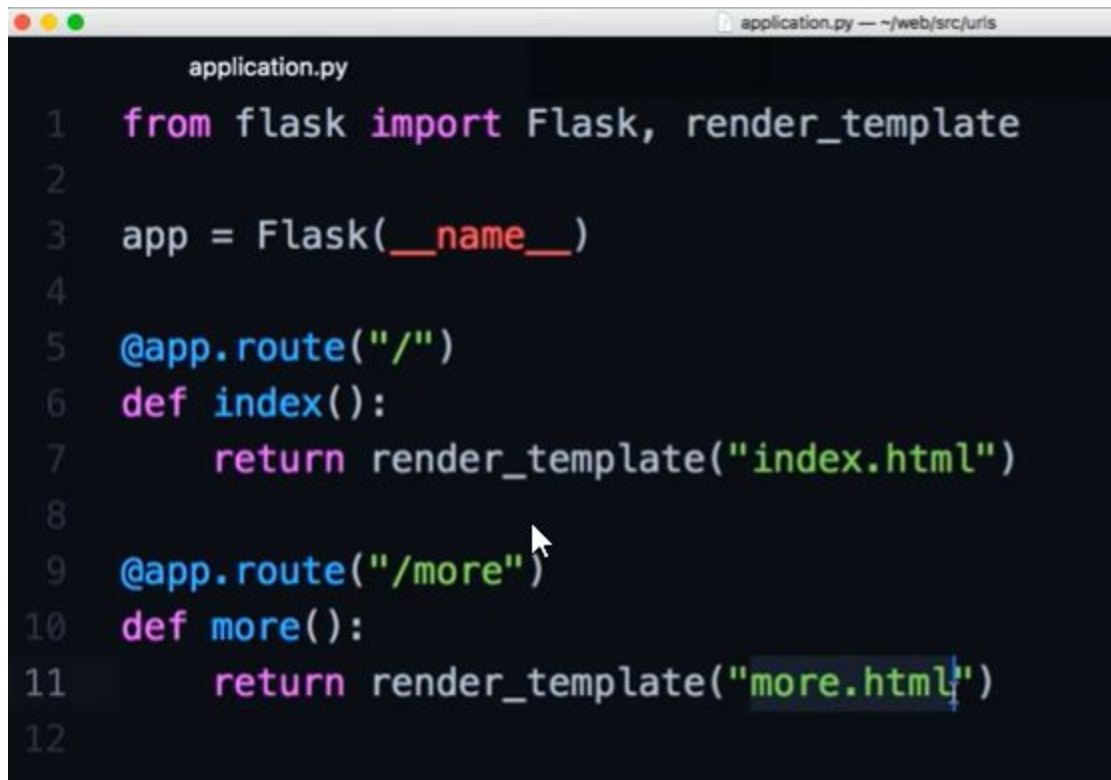
```
application.py    index.html
3      <head>
4          <title>My Website!</title>
5      </head>
6      <body>
7          <h1>Names</h1>
8
9          <ul>
10              {% for name in names %}
11                  <li>{{ name }}</li>
12              {% endfor %}
13          </ul>
14      </body>
15  </html>
16
```



9.url_for()

Well, this function get the name of function in flask file and route to the **url of that function**


And when page navigate to that url, the function automatically would be called

A screenshot of a code editor window titled 'application.py -- ~/web/src/uris'. The code is written in Python and defines a Flask application. It imports Flask and render_template, creates an app instance, and defines two routes: a root route '/' with an index function, and a '/more' route with a more function. The 'more.html' string in the second route is highlighted with a blue selection box.

```
application.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
9  @app.route("/more")
10 def more():
11     return render_template("more.html")
12
```

```
index.html -- ~/web/src/urls
application.py  index.html
3  <head>
4      <title>My Website!</title>
5  </head>
6  <body>
7      <h1>First Page</h1>
8
9      <p>
10         Lorem ipsum dolor sit amet, consectetur adipiscing elit.
11     </p>
12
13     <a href="{{ url_for('more') }}">See more...</a>
14 </body>
15 </html>
```

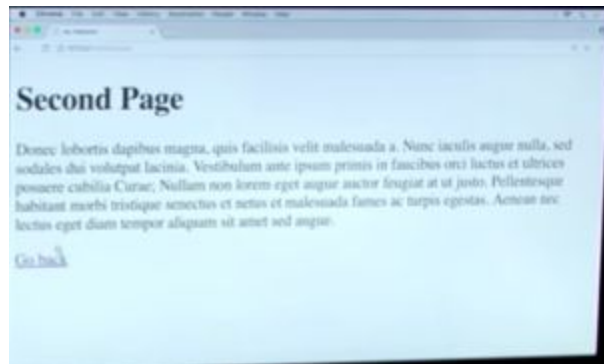
```
more.html -- ~/web/src/urls
application.py  index.html  more.html
3  <head>
4      <title>My Website!</title>
5  </head>
6  <body>
7      <h1>Second Page</h1>
8
9      <p>
10         Donec lobortis dapibus magna, quis facilisis velit malesuada
11     </p>
12
13     <a href="{{ url_for('index') }}">Go back</a>
14 </body>
15 </html>
```



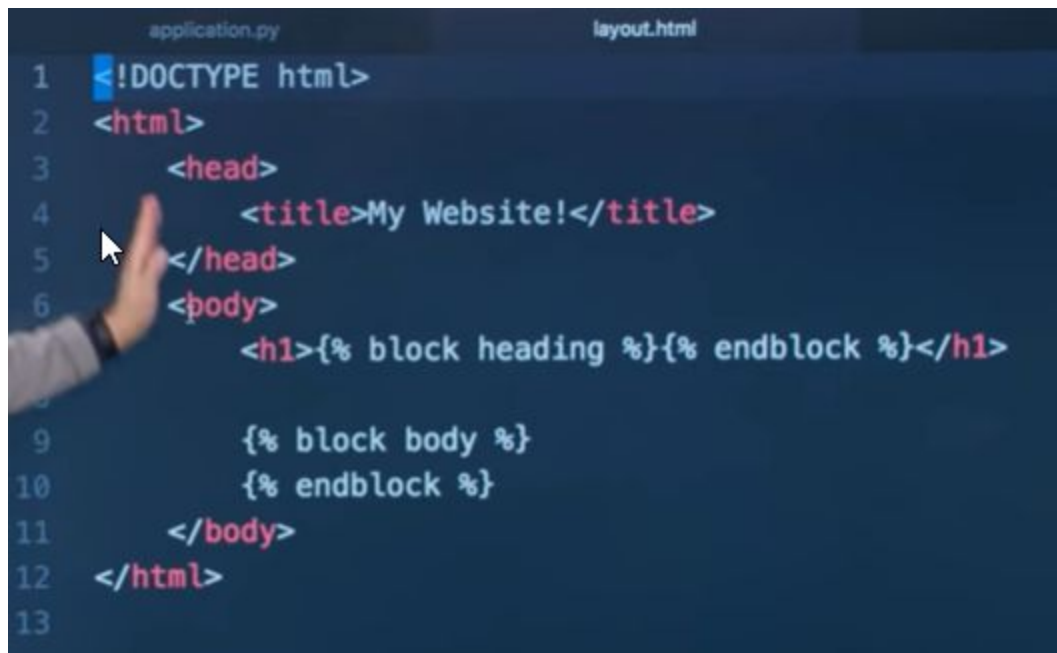
First Page

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce placerat rutrum nisi at pellentesque. Fusce orci magna, pulvinar ut ultricies id, feugiat eu elit. Nam molestie eu lacus eu fermentum. Fusce eleifend tempus sapien eu maximus. Donec lobortis dapibus magna, quis facilisis velit malesuada a. Nunc iaculis augue nulla, sed sodales dui volutpat lacinia.

[See more...](#)



10. Avoiding of repeating code and create base file with jinja



```
application.py  layout.html
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>My Website!</title>
5   </head>
6   <body>
7     <h1>{% block heading %}{% endblock %}</h1>
8
9     {% block body %}
10    {% endblock %}
11  </body>
12 </html>
13
```



```
application.py  layout.html  index.html
1 {% extends "layout.html" %}
2
3 {% block heading %}
4   First Page
5 {% endblock %}
6
7 {% block body %}
8   <p>
9     Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce
10   </p>
11
12   <a href="{{ url_for('more') }}">See more...</a>
13 {% endblock %}
```

```
more.html — ~/web/src/inheritance
application.py  layout.html  index.html  more.html
1 {% extends "layout.html" %}
2
3 {% block heading %}
4     Second Page
5 {% endblock %}
6
7 {% block body %}
8     <p>
9         Donec lobortis dapibus magna, quis facilisis velit malesuada a.
10    </p>
11
12    <a href="{{ url_for('index') }}">Go back</a>
13 {% endblock %}
14
```

11. Working with Post and Get methods

```
application.py — ~/web/src/forms
1 from flask import Flask, render_template, request
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def index():
7     return render_template("index.html")
8
9 @app.route("/hello", methods=["POST"])
10 def hello():
11     name = request.form.get("name")
12     return render_template("hello.html", name=name)
13
```



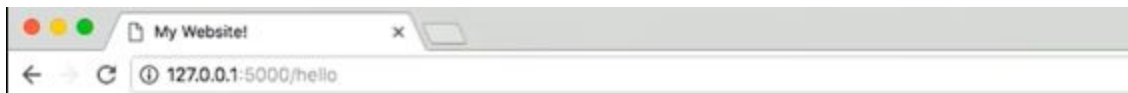
```
index.html — ~/web/src/forms
application.py  index.html
1  {% extends "layout.html" %}
2
3  {% block heading %}
4      First Page
5  {% endblock %}
6
7  {% block body %}
8      <form action="{% url_for('hello') %}" method="post">
9          <input type="text" name="name" placeholder="Enter Your Name">
10         <button>Submit</button>
11     </form>
12 {% endblock %}
13
```

```
application.py  hello.html  index.html
1  {% extends "layout.html" %}
2
3  {% block heading %}
4      Hello!
5  {% endblock %}
6
7  {% block body %}
8      Hello, {{ name }}!
9  {% endblock %}
10
```





First Page



Hello!

Hello, Brian!

12. Separating Get and Post request

```
application.py      hello.html      index.html
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      return render_template("index.html")
8
9  @app.route("/hello", methods=["GET", "POST"])
10 def hello():
11     if request.method == "GET":
12         return "Please submit the form instead."
13     else:
14         name = request.form.get("name")
15         return render_template("hello.html", name=name)
```

Note: and if you try to submit data by get method, the get data would be put into the url

0.1:5000/hello?name=Brian

nit the form instead.

And it's not good for security...

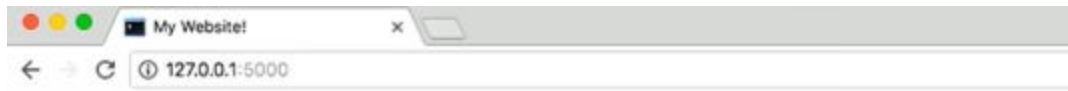
14. Storing data with arrays

```
application.py
6 app.config["SESSION_PERMANENT"] = False
7 app.config["SESSION_TYPE"] = "filesystem"
8 Session(app)
9
10 notes = []
11
12 @app.route("/", methods=["GET", "POST"])
13 def index():
14     if request.method == "POST":
15         note = request.form.get("note")
16         notes.append(note)
17
18     return render_template("index.html", notes=notes)
```

```
application.py  index.html
{% block body %}

    <ul>
        {% for note in notes %}
            <li>{{ note }}</li>
        {% endfor %}
    </ul>

    <form action="{{ url_for('index') }}" method="post">
        <input type="text" name="note" placeholder="Enter Note Here">
        <button>Add Note</button>
    </form>
{% endblock %}
```

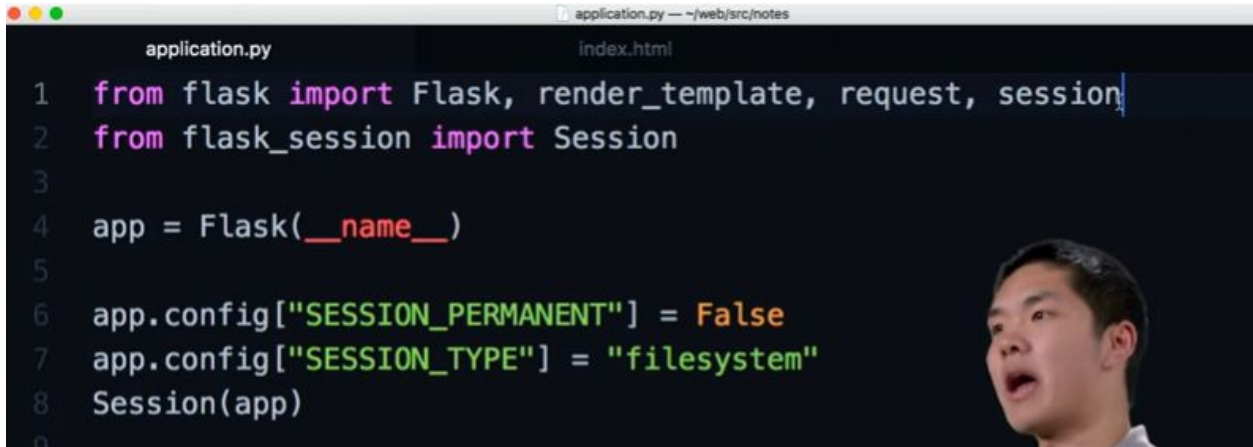


Notes

- hello
- hello again

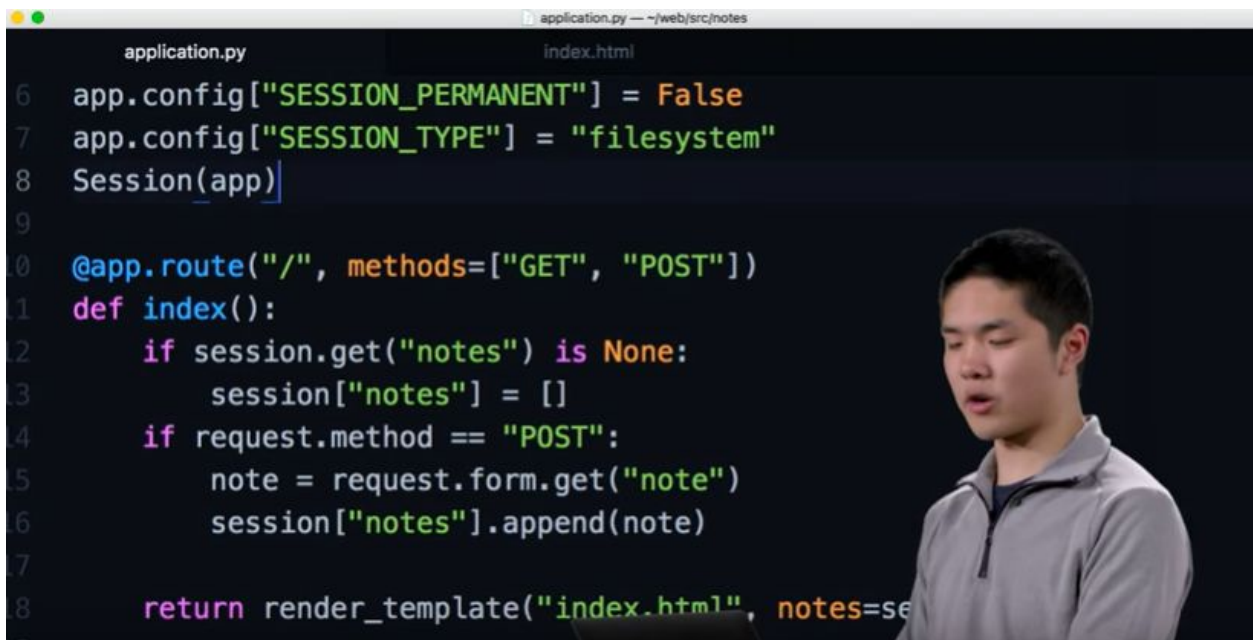
15. Storing data by sessions

This is necessary for separate different users' data



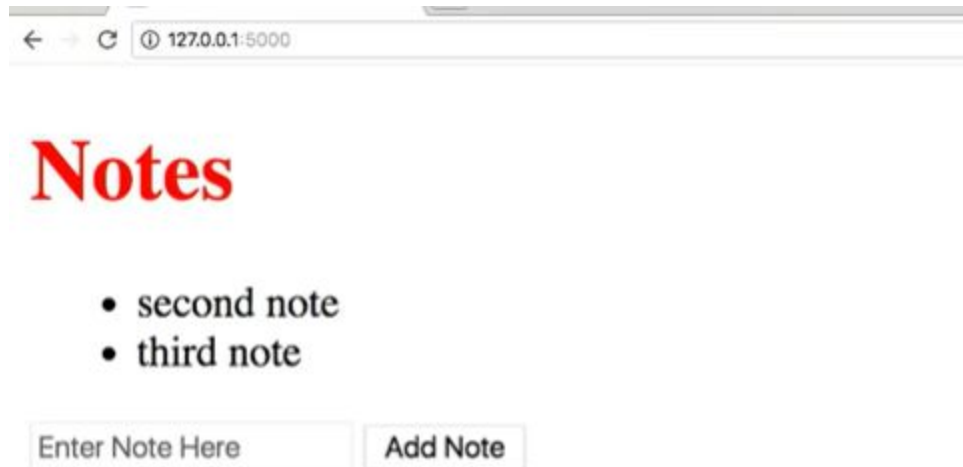
```
application.py | index.html
1 from flask import Flask, render_template, request, session
2 from flask_session import Session
3
4 app = Flask(__name__)
5
6 app.config["SESSION_PERMANENT"] = False
7 app.config["SESSION_TYPE"] = "filesystem"
8 Session(app)
```

A man is visible in the bottom right corner of the frame, looking towards the code.



```
application.py | index.html
6 app.config["SESSION_PERMANENT"] = False
7 app.config["SESSION_TYPE"] = "filesystem"
8 Session(app)
9
10 @app.route("/", methods=["GET", "POST"])
11 def index():
12     if session.get("notes") is None:
13         session["notes"] = []
14     if request.method == "POST":
15         note = request.form.get("note")
16         session["notes"].append(note)
17
18     return render_template("index.html", notes=session["notes"])
```

A man is visible in the bottom right corner of the frame, looking down at the code.



Ok now what is the difference? Well first if you close the tab your data wouldn't be removed, second every single user has his own data!

The End!!