

# آموزش مینی فریمورک فلسک

سید عماد رضوی

از ۱۳۹۶/۸/۲۶ (تولد ۶۶ سالگی پدرم) تا ۱۳۹۶/۹/۹ (تولد ۵ سالگی امیرعلی، کوچولوی فامیل)



این پروژه متن باز و بر پایه برنامه حروف چین Tex است. آخرین نسخه فایل Tex و PDF این آموزش در سایت محبوب GitLab قرار می گیرد. اگر مایل به بهبود آن هستید، به آموزش فلسک در GitLab مراجعه کنید.

## فهرست مطالب

۴	۱	پیش نیاز
۴	۲	بررسی اجمالی
۴	۱.۲	فریم ورک تحت وب چیست؟
۴	۲.۲	فلسک چیست؟
۴	۳.۲	WSGI
۴	۴.۲	Werkzeug
۴	۵.۲	jinja2
۵	۳	محیط کاری
۵	۱.۳	پیش نیازها
۵	۲.۳	نصب virtualenv برای محیط توسعه
۶	۴	برنامه
۷	۱.۴	حالت دیباگ
۷	۵	مسیریابی
۸	۶	rule های از نوع متغیر
۹	۷	ساخت آدرس اینترنتی
۱۰	۸	متدهای http
۱۰	۱.۸	متد GET
۱۰	۲.۸	متد HEAD
۱۰	۳.۸	متد POST
۱۰	۴.۸	متد PUT
۱۱	۵.۸	متد Delete
۱۳	۹	قالب ها
۱۶	۱۰	فایل های استاتیک
۱۷	۱۱	شی request
۱۸	۱۲	ارسال داده های فرم به قالب
۲۰	۱۳	کوکی ها
۲۲	۱۴	Session ها
۲۴	۱۵	ریدایرکت و ارورهایش
۲۶	۱۶	Flashing Message
۳۰	۱۷	آپلود فایل
۳۲	۱۸	افزونه ها
۳۳	۱۹	ایمیل
۳۳	۱.۱۹	کلاس Mail
۳۳	۲.۱۹	متد های کلاس Mail
۳۳	۳.۱۹	کلاس Message
۳۴	۴.۱۹	متدهای کلاس Message
۳۴	۵.۱۹	یک مثال کاربردی
۳۶	۲۰	WTF

۴۲	SQLite ۲۱
۴۷	SQLAlchemy ۲۲
۵۳	سایجکس ۲۳
۵۳	نصب ۱.۲۳
۵۳	تنظیمات ۲.۲۳
۵۴	برنامه Sijax ۳.۲۳
۵۵	توسعه ۲۴
۵۵	سرور خارجی قابل مشاهده ۱.۲۴
۵۵	گسترش ۲.۲۴
۵۵	نصب مود WSGI ۳.۲۴
۵۵	ساخت فایل wsgi ۴.۲۴
۵۶	تنظیمات Apache ۵.۲۴
۵۶	حامل های مستقل WSGI ۶.۲۴
۵۶	FastCGI ۲۵
۵۶	تنظیمات FastCGI ۱.۲۵
۵۷	تنظیمات Apache ۲.۲۵
۵۷	تنظیمات lighttpd ۳.۲۵

## ۱ پیش نیاز

برای شروع این آموزش، فرض ما بر این است که شما تجربه کدنویسی اچ تی ام ال و زبان پایتون را دارید. اگر از مفاهیم این دو آگاه نیستید، توصیه ما این است که دوره های کوتاهی درباره اچ تی ام ال و پایتون ببینید.

## ۲ بررسی اجمالی

### ۱.۲ فریم ورک تحت وب چیست؟

فریم ورک برنامه تحت وب یا به بیان ساده تر، فریم ورک وب شامل مجموعه ای از کتابخانه ها و ماژول هایی است که به یک توسعه دهنده برنامه تحت وب امکان می دهد که برنامه ها را بدون درگیری با جزئیات سطح پایین مانند پروتکل ها، مدیریت تردها (threads) بنویسد.

### ۲.۲ فلسک چیست؟

فلسک، فریمورک توسعه برنامه های تحت وب برای زبان برنامه نویسی پایتون است. Armin Ronacher توسعه دهنده آن است که گروه علاقه مندان به پایتون با نام Poccco را نیز رهبری می کند. فلسک بر پایه ابزار Werkzeug WSGI و موتور نشانه گذاری Jinja۲ کار می کند که هر دو از پروژه های Ronacher هستند.

### ۳.۲ WSGI

مخفف Web Server Gateway Interface یا دروازه واسط وب سرور، به عنوان یک استاندارد برای توسعه نرم افزار وب پایتون به کار گرفته شده است. WSGI یک مشخصه ی واسطه ی همگانی میان وب سرور و برنامه تحت وب است.

### ۴.۲ Werkzeug

یک ابزار WSGI است که پیاده سازی درخواست ها، پاسخ ها و سایر توابع ابزار را بر عهده دارد که ساخت یک وب فریم ورک را بر روی خود ممکن می کند. فریم ورک فلسک بر روی ابزار Werkzeug بنا نهاده شده است.

### ۵.۲ jinja2

جینجا دو، یک موتور نشانه گذاری محبوب در زبان پایتون است. یک سیستم نشانه گذاری وب، با ترکیب یک قالب (مثلا html) با منبع خاص داده، صفحات وب پویا (دینامیک) را تولید می کند. فلسک معمولاً یک میکرو فریم ورک نامیده می شود و هدف آن این است که هسته برنامه را ساده و در عین حال قابل گسترش کند. فلسک دارای یک لایه ی انتزاعی ساخته شده برای مدیریت دیتابیس نیست و پشتیبانی معتبری نیز بر روی آن صورت نمی گیرد. در عوض، فلسک از افزونه ها برای اضافه کردن این قابلیت ها به برنامه (در حال توسعه) پشتیبانی می کند. بعضی از افزونه های محبوب فلسک در ادامه این دوره مورد بحث قرار می گیرد.

## ۳ محیط کاری

### ۱.۳ پیش نیازها

برای نصب فلسک معمولاً به پایتون بالاتر از نسخه 2.6 نیاز است. هر چند که فلسک و وابستگی هایش (dependencies) به خوبی با پایتون نسخه ۳ (پایتون 3.3 به بعد) کار می کنند، معمولاً افزونه های فلسک به خوبی از این نسخه پشتیبانی نمی کند. پس توصیه می شود که بهتر است فلسک بر روی پایتون 2.7 نصب شود.

### ۲.۳ نصب virtualenv برای محیط توسعه

virtualenv یک سازنده محیط مجازی پایتون است که به کاربر در ساخت چندین محیط پایتون در کنار هم کمک می کند. به این ترتیب می توان از مشکلات سازگاری بین نسخه های مختلف کتابخانه ها جلوگیری کرد. دستور زیر virtualenv را در مسیر C:/pythonX/scripts نصب می کند که X در آن نام نسخه پایتون است (مثلاً ۲۷)

```
pip install virtualenv
```

خروجی باید چیزی شبیه به زیر باشد:

```
Collecting virtualenv
  Downloading virtualenv-15.0.1-py2.py3-none-any.whl (1.8MB)
    100% ##### 1.8MB 204kB/s
Installing collected packages: virtualenv
Successfully installed virtualenv-15.0.1
```

این دستور نیاز به دسترسی administrator دارد. در سیستم عامل های لینوکس و مک، قبل از pip از کلمه ی sudo استفاده کنید. برای ویندوز، به عنوان ادمین لاگین شوید. در سیستم عامل اوبونتو ممکن است virtualenv خود به همراه مدیریت بسته نصب شده باشد.

```
Sudo apt-get install virtualenv
```

پس از نصب، توسط کد زیر، محیط مجازی جدید در یک پوشه ایجاد می شود:

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

برای فعال کردن محیط مربوطه در لینوکس یا مکینتاش از کدهای زیر استفاده کنید:

```
venv/bin/activate
```

یا در محیط ویندوز به این شکل :

```
venv\scripts\activate
```

ما اکنون آماده نصب فلسک در این محیط مجازی هستیم.

```
pip install Flask
```

خروجی باید چیزی شبیه به زیر باشد:

```
Collecting Flask
  Downloading Flask-0.10.1.tar.gz (544kB)
    100% ##### 544kB 410kB/s
Collecting Werkzeug>=0.7 (from Flask)
```

```

Downloading Werkzeug-0.11.4-py2.py3-none-any.whl (305kB)
100% ##### 307kB 531kB/s
Collecting Jinja2>=2.4 (from Flask)
Downloading Jinja2-2.8-py2.py3-none-any.whl (263kB)
100% ##### 266kB 935kB/s
Collecting itsdangerous>=0.21 (from Flask)
Downloading itsdangerous-0.24.tar.gz (46kB)
100% ##### 49kB 1.6MB/s
Collecting MarkupSafe (from Jinja2>=2.4->Flask)
Downloading MarkupSafe-0.23.tar.gz
Installing collected packages: Werkzeug, MarkupSafe,
Jinja2, itsdangerous, Flask
Running setup.py install for MarkupSafe
Running setup.py install for itsdangerous
Running setup.py install for Flask
Successfully installed Flask-0.10.1 Jinja2-2.8 MarkupSafe-0.23
Werkzeug-0.11.4 itsdangerous-0.24

```

دستور بالایی می تواند فلسک را به صورت مستقیم و بدون نیاز به محیط مجازی، virtualenv، بر روی کل سیستم نصب کند.

## ۴ برنامه

برای تست نصب فلسک، کد پایتون زیر را در ادیتور خود با اسم Hello.py بنویسید.

```

from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World'

if __name__ == '__main__':
    app.run()

```

وارد کردن مازول فلاسک به پروژه اجباری است و یک شی از کلاس فلسک، برنامه WSGI ماست.

```
app.route(rule, options)
```

پارامتر rule نشان دهنده آدرس اینترنتی مرتبط با تابع است و options لیستی از پارامترهاست که باید به شی Rule اصلی اعمال شود. در مثال بالایی، آدرس اینترنتی "/" به تابع hello\_world() مقید است. از این رو وقتی صفحه اصلی وب سرور در مرورگر باز می شود، خروجی این تابع به نمایش در می آید. در نهایت متد run() کلاس فلسک برنامه را بر روی سرور توسعه مجازی اجرا می کند.

```
app.run(host, port, debug, options)
```

تمام پارامترها مشخص هستند.

host نام هاستی که به آن گوش می کنیم که به طور پیش فرض localhost یا همان 127.0.0.1 است. برای دسترسی خارجی سرور،

مقدار ۰.۰.۰.۰ را برای آن تنظیم کنید.

port مقدار پیش فرض آن ۵۰۰۰ است.

debug در حالت پیش فرض مقدار false دارد. اگر مقدار true داده شود، اطلاعات دیباگ برنامه نمایش داده می شود.

options برای ارسال اطلاعات به سرور Werkzeug اصلی برنامه استفاده می شود. اسکریپت پایتون بالا در python shell اجرا می شود.

```
Python Hello.py
```

یک پیام در python shell شما را (از موفقیت در اجرای برنامه) مطلع می کند.

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

آدرس اینترنتی بالا (localhost:5000) را درون مرورگر باز کنید. باید پیام 'Hello World' در صفحه نمایش داده شود.

## ۱.۴ حالت دیباگ

یک برنامه فلسک با فراخوانی متد `run()` شروع به کار می کند. با این حال اگرچه برنامه در حال توسعه است، باید پس از هر تغییری در کد برنامه، آن را به صورت دستی مجدداً راه اندازی کنیم. برای جلوگیری از این مشکل، حالت دیباگ را فعال کنید. در این حالت، با هر تغییری در برنامه، سرور مجدداً خودش را اجرا می کند. فایده دیگر آن نیز این است که از یک دیباگر مفید برای ردیابی خطاهای احتمالی برنامه استفاده می کند.

با تنظیم پارامتر دیباگ شی برنامه به مقدار `True` قبل از اجرای برنامه یا ارسال پارامتر دیباگ به متد `run()`، حالت دیباگ فعال می شود.

```
app.debug = True
app.run()
app.run(debug = True)
```

## ۵ مسیریابی

فریم ورک های مدرن وب برای کمک به کاربر جهت به خاطر سپاری آدرس های برنامه از تکنیک مسیریابی استفاده می کنند. دسترسی مستقیم به صفحه مورد نظر بدون نیاز به حرکت از سمت صفحه اصلی، مفید است. دکوراتور `route()` در فلسک برای مقید کردن یک آدرس اینترنتی به یک تابع مورد استفاده قرار می گیرد. برای مثال:

```
@app.route('/hello')
def hello_world():
    return 'hello world'
```

اینجا، دستور آدرس اینترنتی `'/hello'` به تابع `hello_world()` مقید شده است. در نتیجه، اگر کاربر از صفحه ی اینترنتی `http://localhost:5000/hello` بازدید کند، خروجی تابع `hello_world()` در مرورگر ظاهر خواهد شد.

تابع `add_url_rules()` یک شی برنامه هم برای مقید کردن آدرس اینترنتی به یک تابع می تواند مورد استفاده قرار گیرد. همان طور که در مثال بالا، تابع `route()` از آن استفاده می کند. هدف دکوراتور نیز در کدهای زیر قابل مشاهده است.

```
def hello_world():
    return 'hello world'
app.add_url_rule('/', 'hello', hello_world)
```

## ۶ rule های از نوع متغیر

با اضافه کردن قسمت های متغیر به پارامتر rule می توان آدرس های اینترنتی دینامیکی را ساخت. این بخش متغیر به صورت `<variable-name>` مشخص می شود و به صورت یک آرگومان کلیدی به تابعی که rule به آن مربوط است، فرستاده می شود. در مثال پیش رو، پارامتر rule دکوراتور `route()` شامل بخش متغیر `<name>` است که به آدرس اینترنتی `'/hello'` چسبیده است. از این رو اگر آدرس اینترنتی `http://localhost:5000/hello/TutorialsPoint` را مرورگر وارد کنید، بخش `'TutorialsPoint'` به عنوان یک آرگومان به تابع `hello()` ارسال می شود.

```
from flask import Flask
app = Flask(__name__)

@app.route('/hello/<name>')
def hello_name(name):
    return 'Hello %s!' % name

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

اسکرپت بالا را در `hello.py` ذخیره کنید و در شل پایتون آن را اجرا کنید. مرورگر را باز کنید و به آدرس اینترنتی زیر بروید:  
`http://localhost:5000/hello/TutorialsPoint`  
باید خروجی زیر در مرورگر به نمایش در بیاید.

```
Hello TutorialsPoint!
```

علاوه بر بخش متغیر رشته `string` پیش فرض، rule ها می توانند با انواع مبدل (سازنده) دیگری نیز ساخته شوند.  
`int` عدد صحیح قبول می کند.  
`float` متغیر اعشاری می پذیرد  
`path` کاراکتر اسلش را به عنوان کاراکتر جداکننده ی دایرکتوری قبول می کند.  
در کد زیر، هر سه سازنده مورد استفاده قرار گرفته اند.

```
from flask import Flask
app = Flask(__name__)

@app.route('/blog/<int:postID>')
def show_blog(postID):
    return 'Blog Number %d' % postID

@app.route('/rev/<float:revNo>')
def revision(revNo):
    return 'Revision Number %f' % revNo

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

کد بالا را در `python shell` اجرا کنید. آدرس اینترنتی `http://localhost:5000/blog/11` را در مرورگر باز کنید. عدد داده شده، به عنوان آرگومان در تابع `show_blog()` استفاده می شود. مرورگر خروجی زیر را نمایش می دهد.

```
Blog Number 11
```



حال این آدرس را امتحان کنید `http://localhost:5000/rev/1.1`

تابع `revision()` عدد اعشاری را به عنوان آرگومان می گیرد. نتیجه خروجی به شکل زیر در صفحه مرورگر به نمایش در می آید.

```
Revision Number 1.100000
```

قوانین آدرس های اینترنتی `rule` بر پایه ماژول مسیریابی Werkzeug است. این قوانین تضمین می کند که آدرس های اینترنتی ایجاد شده، همگی منحصر به فرد هستند و بر پایه نمونه های ساخت شده توسط آپاچی خواهند بود. اینک `rule` های تعریف شده در اسکریپت زیر را مورد توجه قرار بدهید.

```
from flask import Flask
app = Flask(__name__)

@app.route('/flask')
def hello_flask():
    return 'Hello Flask'

@app.route('/python/')
def hello_python():
    return 'Hello Python'

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

هر دو `rule` به نظر مشابه یکدیگر به نظر می آیند، ولی در دومی در پایان از کاراکتر اسلش (/) استفاده شده است. در نتیجه به یک آدرس اینترنتی کانونی تبدیل شده است. از این رو، استفاده از `/python/` یا `/python/` نتیجه یکسانی خواهد داشت. اما در مورد اول، آدرس اینترنتی `/flask/` به صفحه ۴۰۴ (صفحه یافت نشد) منتهی خواهد شد.

## ۷ ساخت آدرس اینترنتی

تابع `url_for()` برای ساخت یک آدرس اینترنتی پویا برای یک تابع مشخص بسیار مفید است. این تابع، اسم تابع را به عنوان پارامتر و یک یا چند آرگومان کلیدی می گیرد که هر یک برای قرارگیری یک بخش متغیر آدرس اینترنتی خواهد بود. اسکریپت زیر کاربرد تابع `url_for()` را نمایش می دهد.

```
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')
def hello_admin():
    return 'Hello Admin'

@app.route('/guest/<guest>')
def hello_guest(guest):
    return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
    if name == 'admin':
        return redirect(url_for('hello_admin'))
```

```

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

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

```

اسکرپت بالایی تابعی به نام `user(name)` دارد که مقدار آرگومانش را از آدرس اینترنتی می گیرد. تابع `User()` چک می کند که آیا آرگومان گرفته شده برابر 'admin' هست یا نه. اگر بود، برنامه با استفاده از متد `url_for()` به `hello_admin()` منتقل (ریدایرکت) می شود. در غیر این صورت آرگومان گرفته شده به عنوان پارامتر `guest` به تابع `hello_guest()` فرستاده می شود.

کد بالا را ذخیره و در `python shell` اجرا کنید.

مرورگر را باز کنید و آدرس اینترنتی را به این شکل بزنید `http://localhost:5000/user/admin`

پاسخ برنامه در مرورگر به این شکل خواهد بود:

```
Hello Admin
```

حال این آدرس اینترنتی را امتحان کنید `http://localhost:5000/user/mvl`

پاسخ برنامه به این شکل تغییر می کند:

```
Hello mvl as Guest
```

## ۸ متدهای http

پروتکل HTTP زیربنای ارتباطات داده ای در شبکه جهانی وب (WWW) است. در این پروتکل، متدهای مختلفی برای دریافت و بازایی اطلاعات از یک آدرس اینترنتی مشخص تعریف شده است. در ادامه خلاصه ای از متدهای مختلف HTTP مورد توجه قرار می گیرد.

### ۱.۸ متد GET

این متد داده ها را به شکل رمزنگاری نشده! به سرور ارسال می کند. این متد متداول ترین نوع تمام متدهاست.

### ۲.۸ متد HEAD

مشابه متد GET اما بدون پاسخ بخش Body است.

### ۳.۸ متد POST

برای ارسال اطلاعات یک فرم HTML به سرور مورد استفاده قرار می گیرد. اطلاعات گرفته شده توسط متد POST توسط سرور ذخیره (کش) نمی شود.

### ۴.۸ متد PUT

زمانی استفاده می شود که کاربر یک داکيومنت جایگزین می فرستد یا یک داکيومنت را در آدرس اینترنتی درخواست شده، درون وب سرور قرار می دهد.

## ۵.۸ متد Delete

برای حذف تمام داکيومنت های موجود در وب سرور موجود در آدرس اینترنتی مشخص شده مورد استفاده قرار می گیرد. به طور پیش فرض، مسیریابی فلسک به درخواست های GET پاسخ می دهد، هر چند این ترجیح پیش فرض با قرار دادن آرگومان `methods` در دکوراتور `route()` تغییر می کند. به منظور نمایش استفاده از متد POST در مسیریابی آدرس اینترنتی، اول اجازه بدهید یک فرم HTML بسازیم و از متد POST برای ارسال اطلاعات به آدرس اینترنتی استفاده کنیم. اسکریپت زیر را در `login.html` ذخیره کنید:

```
<html>
  <body>

    <form action = "http://localhost:5000/login" method = "post">
      <p>Enter Name:</p>
      <p><input type = "text" name = "nm" /></p>
      <p><input type = "submit" value = "submit" /></p>
    </form>

  </body>
</html>
```

حال اسکریپت زیر را در `python shell` اجرا کنید

```
from flask import Flask, redirect, url_for, request
app = Flask(__name__)

@app.route('/success/<name>')
def success(name):
    return 'welcome %s' % name

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        user = request.form['nm']
        return redirect(url_for('success', name = user))
    else:
        user = request.args.get('nm')
        return redirect(url_for('success', name = user))

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

پس از شروع کار سرور، `login.html` را درون مرورگر باز کنید، یک اسم درون فیلد متن وارد کنید و دکمه Submit را بزنید.

Enter Name:



داده های فرم در بخش اکشن از تگ فرم به آدرس اینترنتی POST (ارسال) می شود.  
 http://localhost/login به تابع login() مقید (نگاشته) شده است. زمانی که (چون!) سرور اطلاعات را با متد POST دریافت می کند، مقدار پارامتر 'nm' که از اطلاعات فرم گرفته شده است، توسط

```
user = request.form['nm']
```

گرفته می شود.  
 سپس به عنوان بخش متغیر به آدرس اینترنتی '/success' ارسال می شود. مرورگر یک پیام خوش آمدگویی در صفحه نمایش می دهد.

welcome mvl

پارامتر متد را در login.html به GET تغییر دهید و مرورگر را دوباره باز کنید. اطلاعات گرفته شده بر روی سرور توسط متد GET بوده است. اکنون مقدار پارامتر 'nm' با

```
User = request.args.get('nm')
```

گرفته می شود.

در اینجا، args یک شی دیکشنری و شامل لیستی از اطلاعات فرم به شکل جفت (پارامتر فرم و مقدارش) است. مشابه قبل، اطلاعات نظیر پارامتر 'nm' به آدرس اینترنتی '/success' فرستاده می شود.

## ۹ قالب ها

خروجی یک تابع را که به یک آدرس اینترنتی مقید شده است را می توان به فرم HTML بازگرداند. برای مثال، در اسکریپت زیر، تابع hello() عبارت 'Hello World' را با استفاده از تگ <h1> که همراهش است، نمایش می دهد.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return '<html><body><h1>Hello World</h1></body></html>'

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

با این حال تولید محتوای HTML در کدهای پایتون بسیار دشوار است. مخصوصا زمانی که نیاز به قرارگیری داده های متغیرها و عناصر زبان پایتون نظیر عبارات شرطی یا حلقه ها وجود دارد که این باعث خروج مکرر به HTML (از زبان پایتون) است. اینجاست که نیاز به استفاده از موتور نشانه گذاری (قالب) جینجا دو مطرح می شود که فلسف بر پایه آن بنا شده است. به جای بازگرداندن کد HTML هاردکد (اطلاعات بدون تغییر) از یک تابع پایتون، با استفاده از تابع render\_template() می توان یک فایل HTML را نمایش داد (رندر کرد)

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('hello.html')

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

فلسف تلاش می کند که فایل HTML را در پوشه ی templates پیدا کند. پوشه ای که خود درون همان پوشه ای است که این اسکریپت در آن قرار دارد.

- Application folder
  - Hello.py
  - templates
    - hello.html

عبارت سیستم قالب وب (web templating system) به اسکریپت HTML ای اشاره دارد که داده های متغیر آن می توانند به شکل پویا (دینامیک) وارد شوند. یک سیستم قالب وب شامل یک موتور قالب، نوعی از منابع داده و یک پردازشگر قالب است.

فلسک از موتور قالب جینجا دو استفاده می کند. یک قالب وب شامل جاهای خالی (placeholder) قرار گرفته میان کدهای HTML برای متغیرها و عبارات است (در این مورد عبارات پایتونی) که در هنگام رندر و نمایش قالب، با مقادیری جایگزین می شوند. کد زیر به نام hello.html در پوشه ی templates ذخیره شده است.

```
<!doctype html>
<html>
  <body>

    <h1>Hello {{ name }}!</h1>

  </body>
</html>
```

حالا این اسکریپت را در محیط شل پایتون اجرا کنید.

```
from flask import Flask , render_template
app = Flask(__name__)

@app.route('/hello/<user>')
def hello_name(user):
    return render_template('hello.html', name = user)

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

پس از شروع کار سرور، مرورگر را باز کنید و آدرس اینترنتی را به شکل http://localhost:5000/hello/mvl بزنید. بخش متغیر آدرس اینترنتی که دارای جای خالی در کد HTML است، {{ name }} خواهد بود.



موتور قالب جینجا دو از جداسازهای زیر برای خروج از کد HTML استفاده می کند.

{% ... %} برای عبارات

{{ ... }} برای عباراتی که می خواهند در خروجی چاپ شوند

{# ... #} برای درج توضیحات (کامنت) که در خروجی چاپ نمی شود

## ... # برای توضیحات خط

در مثال زیر، روش استفاده از جملات شرطی در قالب نمایش داده شده است. rule آدرس اینترنتی تابع hello() پارامترهای عدد صحیح integer می پذیرد. این پارامتر به قالب hello.html فرستاده می شود. درون قالب، مقدار عدد دریافت شده (با نام marks) با عدد پنجاه مقایسه می شود و بدین ترتیب HTML به صورت شرطی رندر می شود (به نمایش در می آید) اسکریپت پایتون به شکل زیر است:

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<int:score>')
def hello_name(score):
    return render_template('hello.html', marks = score)

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

اسکریپت قالب HTML با نام hello.html به این شکل است:

```
<!doctype html>
<html>
  <body>

    {% if marks>50 %}
    <h1> Your result is pass!</h1>
    {% else %}
    <h1>Your result is fail</h1>
    {% endif %}

  </body>
</html>
```

توجه داشته باشید که جملات شرطی if-else و endif درون جداساز {% ... %} محصور شده اند.

اسکریپت پایتون را اجرا کنید و آدرس اینترنتی http://localhost/hello/60 و http://localhost/hello/30 را بزیند تا

ببینید که خروجی HTML بر اساس شرط (بزرگ تر یا کوچک تر از ۵۰) تغییر می کند.

ساختار حلقه های پایتون نیز می توانند درون قالب استفاده شود. در اسکریپت بعدی، تابع result() وقتی آدرس اینترنتی http://localhost:5000/result

باز می شود، یک شی دیکشنری به قالب results.html می فرستد.

کد زیر را در شل پایتون اجرا کنید.

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/result')
def result():
    dict = {'phy':50, 'che':60, 'maths':70}
    return render_template('result.html', result = dict)
```

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

اسکرپت HTML زیر را در result.html و درون پوشه templates ذخیره کنید.

```
<!doctype html>
<html>
  <body>

    <table border = 1>
      {% for key, value in result.iteritems() %}

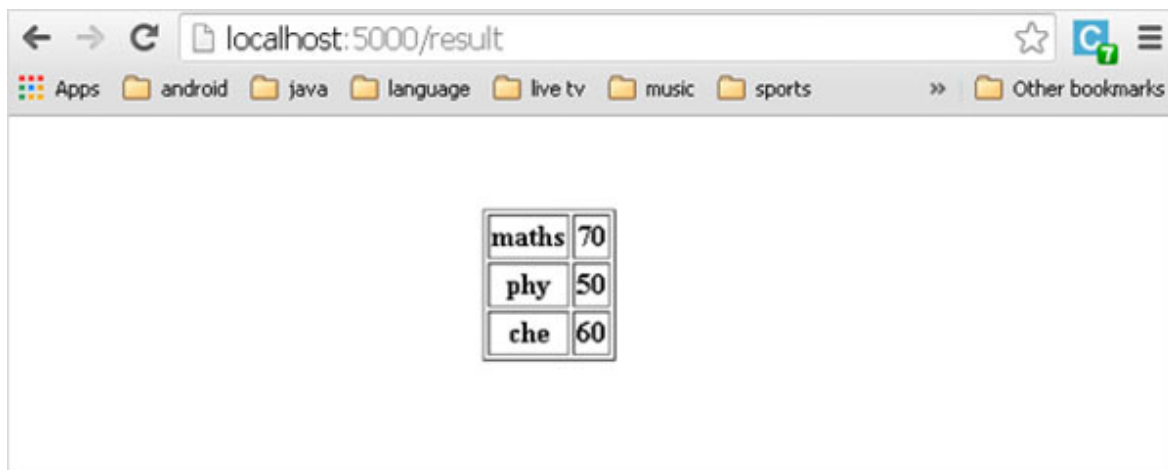
        <tr>
          <th> {{ key }} </th>
          <td> {{ value }} </td>
        </tr>

      {% endfor %}
    </table>

  </body>
</html>
```

در اینجا، عبارات پایتونی مربوط به حلقه for درون عبارت `{% ... %}` محصور شده است. در حالی که عبارات کلیدی و مقادیر درون عبارت `{{ ... }}` قرار گرفته اند.

پس از شروع برنامه، در مرورگر آدرس اینترنتی `http://localhost:5000/result` را باز کنید و خروجی مشابه زیر را ببینید.



## ۱۰ فایل های استاتیک

یک برنامه وب معمولاً به یک فایل استاتیک مانند یک فایل جاوا اسکرپت یا یک فایل CSS که به نمایش بهتر صفحات وب کمک می کند، نیاز دارد. عموماً، وب سرور برای ارائه این خدمت به شما تنظیم شده است، اما در طول توسعه، این فایل ها درون پوشه static درون پکیج برنامه یا در کنار ماژول (اسکرپت پایتون) شما قرار دارد و از آدرس `/static` درون برنامه قابل دسترسی است. یک endpoint با نام 'static' برای ساخت آدرس اینترنتی برای فایل های استاتیک مورد استفاده قرار می گیرد.



در مثال بالا، تابع جاوا اسکریپتی که درون hello.js تعریف شده است، روی رویداد onClick دکمه HTML در index.html تعریف شده است که در آدرس اینترنتی 'lr/' برنامه فلکس نمایش داده می شود.

```
from flask import Flask, render_template
app = Flask(__name__)

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

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

اسکریپت HTML مربوط به فایل index.html در زیر داده شده است:

```
<html>

<head>
  <script type = "text/javascript"
    src = "{{ url_for('static', filename = 'hello.js') }}" ></script>
</head>

<body>
  <input type = "button" onclick = "sayHello()" value = "Say Hello" />
</body>

</html>
```

ماژول hello.js شامل تابع sayHello() است.

```
function sayHello() {
  alert("Hello World")
}
```

## ۱۱ شی request

داده ها از یک صفحه وب کاربر به صورت یک شی request گلوبال به وب سرور ارسال می شود. برای پردازش داده ی شی request، بایستی از ماژول Flask آن را وارد کنیم (import) ویژگی های مهم شی request به ترتیب در زیر آورده شده است.

Form فرم، یک شی از نوع دیکشنری است که دربرگیرنده ی جفت کلید-مقدار پارامترهای فرم و مقادیرشان است.

arg آرگومان ها، محتویات رشته ی query را تجزیه می کند. رشته ی query همان بخش از آدرس اینترنتی است که بعد از علامت سوال (?) می آید.

cookies کوکی ها، شی از نوع دیکشنری که کوکی های مربوط به اسم ها و مقادیر را در خود نگه می دارد.

files فایل ها، داده های مربوط به فایل های آپلود شده است.

method نوع متد request فعلی.

## ۱۲ ارسال داده های فرم به قالب

ما پیش تر دیده ایم که متد HTTP می تواند در قسمت rule آدرس اینترنتی مشخص شود. داده های Form می توانند به شکل یک شی دیکشنری جمع آوری و به یک قالب فرستاده شود تا بر روی صفحه وب متناظر با آن نمایش داده شود. در مثال زیر، آدرس اینترنتی '/' یک صفحه وب را نمایش می دهد (student.html) که شامل یک فرم است. داده های پر شده در آن به آدرس اینترنتی '/result' ارسال می شود که باعث اجرای تابع result() می شود. تابع results() داده های فرم حاضر در request.form را در یک شی دیکشنری جمع می کند و برای نمایش به result.html می فرستد.

قالب به طور پویا (دینامیک) یک جدول HTML از داده های فرم رندر می کند(به نمایش در می آورد) در زیر، کد پایتون برنامه داده شده است:

```
from flask import Flask, render_template, request
app = Flask(__name__)

@app.route('/')
def student():
    return render_template('student.html')

@app.route('/result', methods = ['POST', 'GET'])
def result():
    if request.method == 'POST':
        result = request.form
        return render_template("result.html", result = result)

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

در زیر اسکریپت HTML فایل student.html را مشاهده می کنید.

```
<html>
<body>

<form action = "http://localhost:5000/result" method = "POST">
    <p>Name <input type = "text" name = "Name" /></p>
    <p>Physics <input type = "text" name = "Physics" /></p>
    <p>Chemistry <input type = "text" name = "chemistry" /></p>
    <p>Maths <input type = "text" name = "Mathematics" /></p>
    <p><input type = "submit" value = "submit" /></p>
</form>

</body>
</html>
```

کد قالب (result.html) در زیر داده شده است:

```
<!doctype html>
<html>
<body>
```

```

<table border = 1>
    {% for key, value in result.iteritems() %}

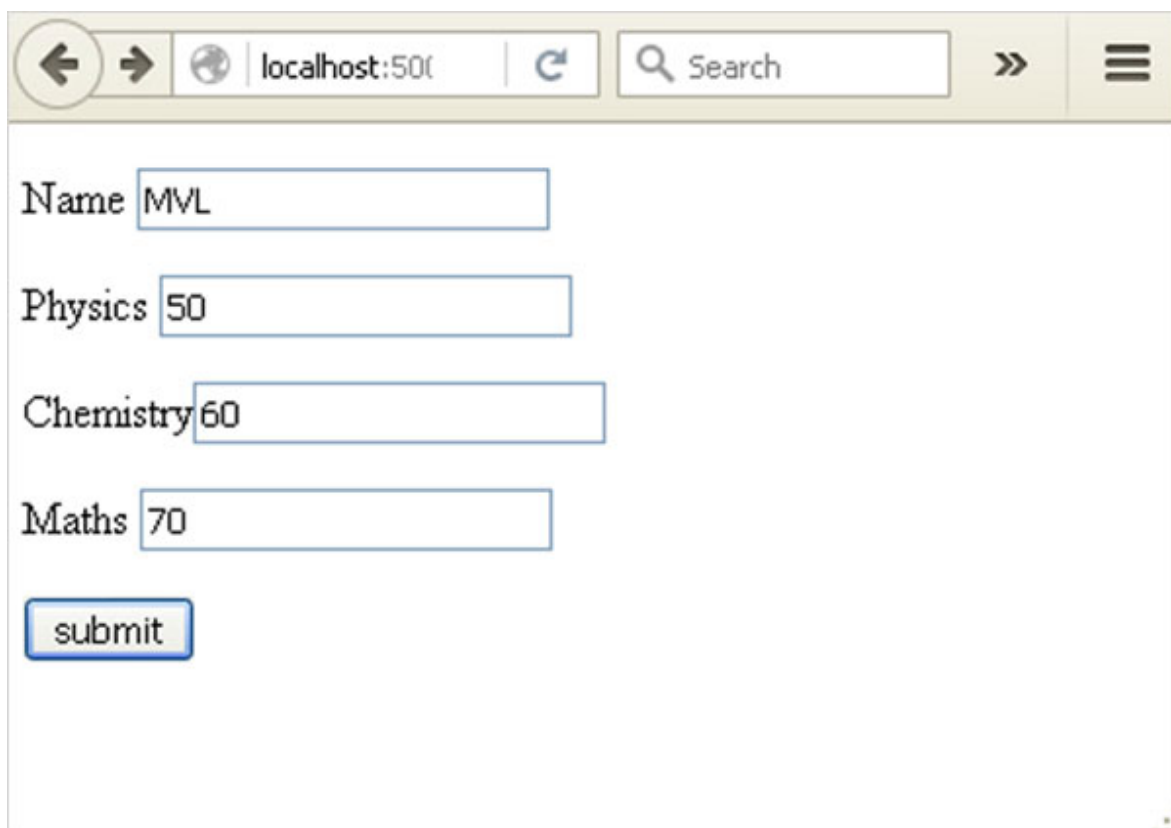
        <tr>
            <th> {{ key }} </th>
            <td> {{ value }} </td>
        </tr>

    {% endfor %}
</table>

</body>
</html>

```

اسکرپت پایتون را اجرا کنید و در مرورگر آدرس اینترنتی <http://localhost:5000/> را بزنید.



The screenshot shows a web browser window with the address bar set to `localhost:5000`. The page displays a form with the following elements:

- Name:** A text input field containing the value "MVL".
- Physics:** A text input field containing the value "50".
- Chemistry:** A text input field containing the value "60".
- Maths:** A text input field containing the value "70".
- submit:** A button labeled "submit" located below the input fields.

زمانی که دکمه Submit کلیک می شود، داده های فرم در `result.html` و به شکل یک جدول HTML به نمایش در می آید.

Mathematics	70
chemistry	60
Physics	50
Name	MVL

## ۱۳ کوکی ها

یک کوکی بر روی کامپیوتر کاربر به فرم فایل متنی ذخیره می شود و هدف آن ردیابی و به خاطر سپاری اطلاعات مربوط به فعالیت های کاربر جهت بهبود تجربه کاربر و آمار سایت است.

یک شی request شامل یک مشخصه ی cookie است. این مشخصه یک شی دیکشنری شامل تمام متغیرهای کوکی و مقادیر متناظر با آن است که کاربر آن را ایجاد می کند. به علاوه، یک کوکی زمان انقضا، مسیر و نام دامنه سایت را در خود ذخیره می کند. در فلسک، کوکی ها در شی response تنظیم می شوند. از تابع make\_response() برای دریافت شی response به منظور برگرداندن مقدار یک تابع view استفاده کنید. سپس از تابع set\_cookie() شی response برای ذخیره کوکی استفاده کنید. خواندن دوباره کوکی آسان است. متد get() مشخصه ی request.cookies برای خواندن کوکی مورد استفاده قرار می گیرد. در برنامه فلسک زیر، با مشاهده آدرس اینترنتی '/', یک فرم ساده باز می شود.

```
@app.route('/')
def index():
    return render_template('index.html')
```

این صفحه HTML شامل یک ورودی متن text input است.

```
<html>
<body>

<form action = "/setcookie" method = "POST">
    <p><h3>Enter userID</h3></p>
    <p><input type = 'text' name = 'nm'></p>
    <p><input type = 'submit' value = 'Login'></p>
</form>

</body>
</html>
```

فرم به آدرس اینترنتی '/setcookie' ارسال می شود. تابع مربوط به این آدرس، یک کوکی با نام userID تنظیم می کند و در صفحه دیگر آن را نمایش می دهد.

```
@app.route('/setcookie', methods = ['POST', 'GET'])
def setcookie():
    if request.method == 'POST':
        user = request.form['nm']

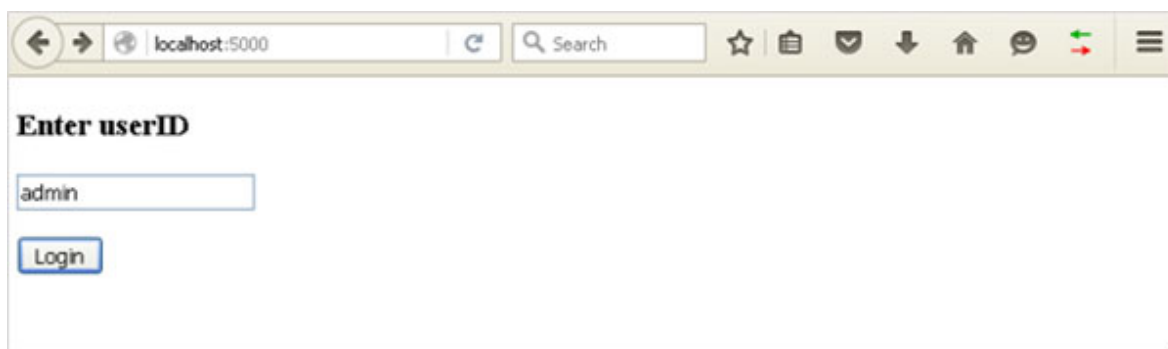
        resp = make_response(render_template('readcookie.html'))
        resp.set_cookie('userID', user)

    return resp
```

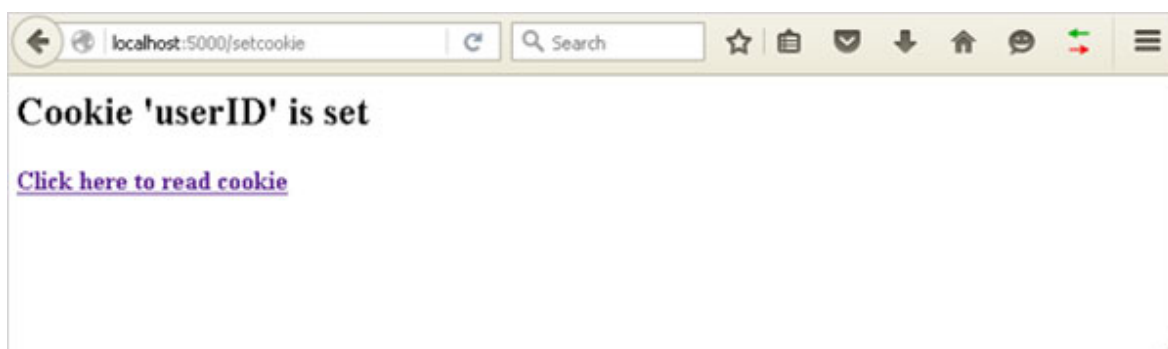
فایل HTML با نام 'readcookie.html' شامل یک لینک داخلی به تابع نمایش getcookie() است که مقدار کوکی را می خواند و در مرورگر به نمایش می گذارد.

```
@app.route('/getcookie')
def getcookie():
    name = request.cookies.get('userID')
    return '<h1>welcome '+name+'</h1>'
```

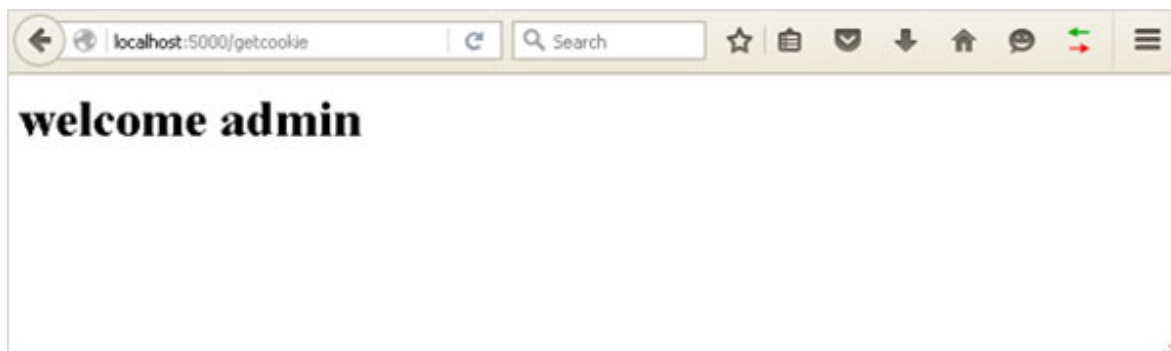
برنامه را اجرا کنید و آدرس http://localhost:5000/ را مشاهده کنید.



نتیجه ی تنظیم و ذخیره کردن کوکی به شکل زیر به نمایش در می آید.



خروجی کوکی خوانده شده در زیر به نمایش در می آید.



## ۱۴ Session ها

(مترجم) شاید ملموس ترین مثال برای Session در ذهن کاربران فارسی زبان، عبارت show all sessions در بخش تنظیمات برنامه تلگرام است که برای بررسی اینکه آیا اکانت تلگرام هک شده است یا نه، مورد استفاده قرار می گیرد. ورود یک کاربر در برنامه تلگرام با یک دستگاه جدید، یک Session جدید می سازد. (پایان مترجم)

بر خلاف یک کوکی، اطلاعات session سمت سرور ذخیره می شود. Session فاصله زمانی میان لاگین کاربر به سرور و خروج او از آن است. داده ها، که بایستی در طول این Session ذخیره شود، در یک دایرکتوری موقتی در سرور ذخیره می شود.

برای یک Session و هر کاربر (مربوط به آن) یک Session ID تعیین می شود. داده های Session بر روی کوکی ساخته می شود و سرور آن ها را به صورت رمزنگاری شده نشانه گذاری می کند. برای این رمزنگاری، برنامه فلسک نیاز به یک SECRET\_KEY تعریف شده دارد.

شی Session همچنین یک شی دیکشنری شامل جفت کلید-مقدار متغیرهای session و مقادیر متناظر با آن است. برای مثال، برای تنظیم session با نام 'username' این عبارت را استفاده کنید.

```
Session['username'] = 'admin'
```

برای آزاد کردن (حذف) یک session از متد pop() استفاده کنید.

```
session.pop('username', None)
```

کد زیر نمایش ساده ای از کارکرد session در فلسک است. آدرس اینترنتی '/' کاربر را برای لاگین کردن هدایت می کند، زیرا متغیر Session با نام 'username' هنوز تنظیم نشده است.

```
@app.route('/')
def index():
    if 'username' in session:
        username = session['username']
        return 'Logged in as ' + username + '<br>' + \
            "<b><a href = '/logout'>click here to log out</a></b>"
    return "You are not logged in <br><a href = '/login'></b>" + \
        "click here to log in</b></a>"
```

زمانی که کاربر صفحه ی '/login' را در مرورگر می زند، تابع login() به دلیل اینکه از طریق متد GET صدا زده شده است، یک فرم لاگین را باز می کند.

سپس یک فرم به صفحه '/login' برگردانده می شود و اینک متغیرهای session تنظیم می شود. برنامه به آدرس '/' منتقل می شود. در این لحظه متغیر session با نام 'username' پیدا شده است.

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return ''

<form action = "" method = "post">
    <p><input type = text name = username/></p>
    <p><input type = submit value = Login/></p>
</form>

'''
```

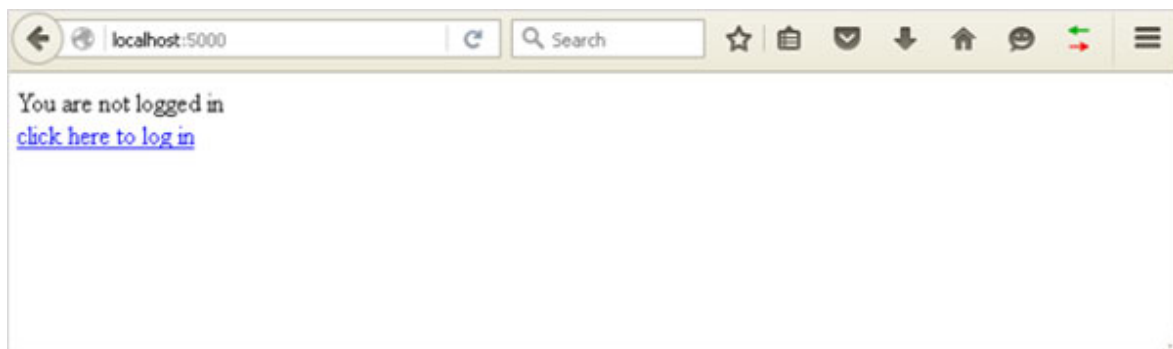
برنامه همچنین شامل یک تابع نمایش `logout()` است که متغیر `session` با نام `'username'` را حذف می‌کند. پس از آن آدرس `'/'` مجدداً صفحه اول را به نمایش در می‌آورد.

```
@app.route('/logout')
def logout():
    # remove the username from the session if it is there
    session.pop('username', None)
    return redirect(url_for('index'))
```

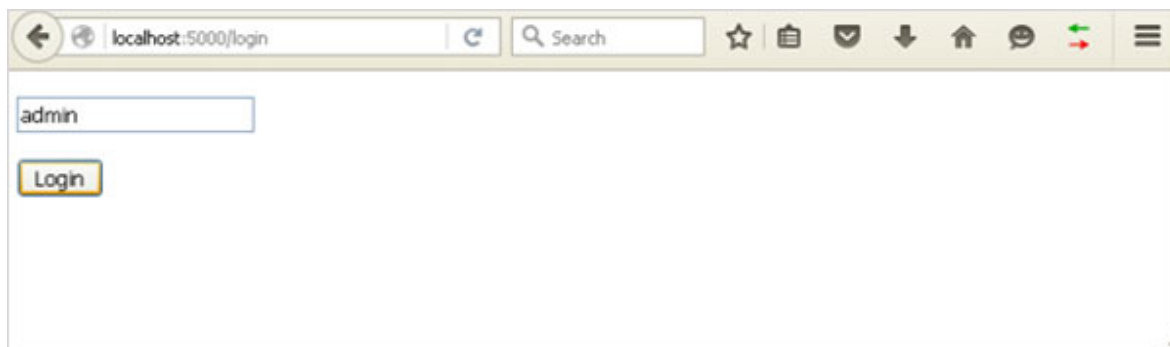
برنامه را اجرا کنید و صفحه اصلی را باز کنید. (از تنظیم کردن مقدار `secret_key` مطمئن شوید)

```
from flask import Flask, session, redirect, url_for, escape, request
app = Flask(__name__)
app.secret_key = 'any random string'
```

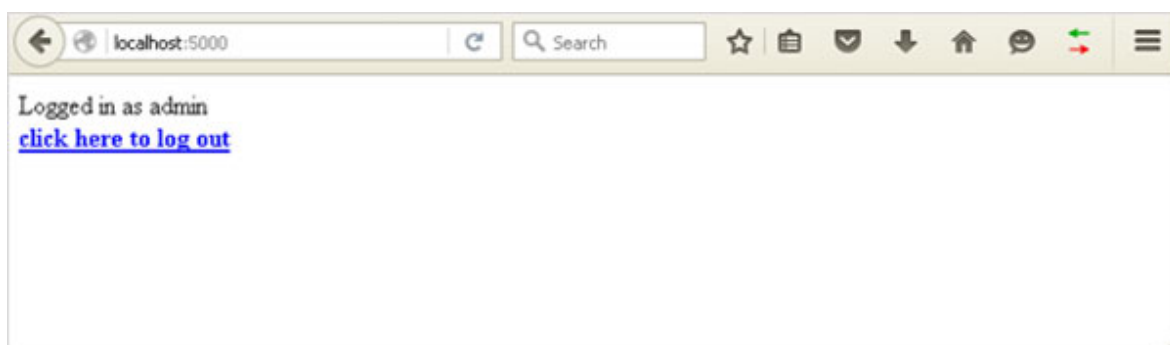
خروجی به صورت زیر به نمایش در می‌آید. بر روی لینک `click here to log in` کلیک کنید.



لینک شما را به صفحه دیگری منتقل می‌کند. عبارت `'admin'` را بنویسید.



صفحه نمایش پیام 'logged in as admin' را به نمایش در می آورد.



## ۱۵ ریدایرکت و ارورهایش

کلاس فلسک دارای یک تابع `redirect()` است که وقتی صدا زده شود، شی `response` را برگردانده و کاربر را به مکان دیگری با کد وضعیت مشخصی هدایت (ریدایرکت) می کند. نمونه اولین تابع `redirect()` در زیر مشاهده می شود.

```
Flask.redirect(location, statuscode, response)
```

در تابع بالا:

`location`

این پارامتر آدرسی را مشخص می کند که `response` به آن هدایت می شود.

`statuscode`

کد وضعیت ارسال شده به بخش هدر مرورگر را مشخص می کند، به طور پیش فرض عدد ۳۰۲ است.

`response`

پارامتری که برای ساخت شی `response` مورد استفاده قرار می گیرد.

کدهای وضعیت استاندارد در زیر آمده است.

`HTTP_300_MULTIPLE_CHOICES`

`HTTP_301_MOVED_PERMANENTLY`

`HTTP_302_FOUND`

`HTTP_303_SEE_OTHER`

`HTTP_304_NOT_MODIFIED`

`HTTP_305_USE_PROXY`



HTTP\_306\_RESERVED

HTTP\_307\_TEMPORARY\_REDIRECT

کد وضعیت پیش فرض ۳۰۲ است که برای حالت found می باشد.

در مثال زیر، از تابع redirect() برای نمایش مجدد صفحه لاگین، زمانی که تلاش برای لاگین به شکست منجر می شود، استفاده می

شود.

```
from flask import Flask, redirect, url_for, render_template, request
# Initialize the Flask application
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('log_in.html')

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST' and
    request.form['username'] == 'admin' :
        return redirect(url_for('success'))
        return redirect(url_for('index'))

@app.route('/success')
def success():
    return 'logged in successfully'

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

کلاس فلسک تابعی به نام abort() با یک کد ارور دارد:

Flask.abort(code)

پارامتر code یکی از مقادیر زیر را می گیرد.

Request Bad - 400

Unauthenticated - 401

Forbidden - 403

Found Not - 404

Acceptable Not - 406

Type Media Unsupported - 415

Requests Many Too - 429

اجازه بدهید یک تغییر جزئی در عملکرد تابع login() در کد بالایی داشته باشیم. به جای نمایش مجدد صفحه ی لاگین، اگر صفحه ی 'Unauthourized' می خواهد نمایش داده شود، آن را با صدا زدن تابع abort(401) جایگزین کنید.

```
from flask import Flask, redirect, url_for, render_template, request, abort
app = Flask(__name__)

@app.route('/')
def index():
```

```
def index():
    return render_template('log_in.html')

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        if request.form['username'] == 'admin':
            return redirect(url_for('success'))
        else:
            abort(401)
    else:
        return redirect(url_for('index'))

@app.route('/success')
def success():
    return 'logged in successfully'

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

## ۱۶ Flashing Message

یک برنامه با GUI (واسط گرافیکی کاربر) خوب، در مورد تعامل های کاربر (نظیر کلیک بر روی یک دکمه) بازخوردی را برای کاربر فراهم می کند. مثلا، برنامه های دسکتاپی از دیالوگ یا مسیج باکس و یا جاوا اسکریپت از هشدارها برای چنین منظوری بهره می برد. در برنامه ی وب فلسک ایجاد چنین پیام های دارای اطلاعات مفیدی ساده است. سیستم های فلشینگ فریم ورک فلسک امکان ساخت پیام در یک view و مشاهده و آن را در تابع نمایشی که بعدا فراخوانی می شود، نمایش می دهد. ماژول فلسک شامل متد flash() است. این متد یک پیام به request بعدی می فرستد که عموما یک قالب است.

```
flash(message, category)
```

اینجا،

پارامتر message پیامی ست که باید flash شود.

پارامتر category اختیاری ست. می تواند 'error' یا 'warning' باشد.

برای حذف پیام از session، قالب تابع get\_flashed\_messages() را صدا می زند.

```
get_flashed_messages(with_categories, category_filter)
```

بیایید یک مثال ساده را ببینیم که مکانیزم flashing را در پایتون به نمایش می گذارد. در کد زیر، یک آدرس اینترنتی '/' لینکی را به صفحه ی لاگین نشان می دهد که پیامی برای flash کردن ندارد.

```
@app.route('/')
def index():
    return render_template('index.html')
```

این لینک کاربر را به آدرس اینترنتی '/login' هدایت می کند که در آن فرم لاگین را به نمایش می گذارد. پس از پر کردن فرم و کلیک بر روی دکمه ی Submit، تابع نمایش login() رمز عبور و پسورد را تایید می کند و متعاقبا پیام 'success' را flash می کند یا اینکه متغیر 'error' را می سازد.

```

@app.route('/login', methods = ['GET', 'POST'])
def login():
    error = None

    if request.method == 'POST':
        if request.form['username'] != 'admin' or \
            request.form['password'] != 'admin':
            error = 'Invalid username or password. Please try again!'
        else:
            flash('You were successfully logged in')
            return redirect(url_for('index'))
    return render_template('login.html', error = error)

```

در صورت بروز خطا، قالب لاگین با یک پیام ارور، مجدداً به نمایش در می آید.

## login.html

```

<!doctype html>
<html>
  <body>

    <h1>Login</h1>

    {% if error %}
    <p><strong>Error:</strong> {{ error }}
    {% endif %}

    <form action = "" method = post>
      <dl>
        <dt>Username:</dt>

        <dd>
          <input type = text name = username
            value = "{{request.form.username }}">
        </dd>

        <dt>Password:</dt>
        <dd><input type = password name = password></dd>
      </dl>
      <p><input type = submit value = Login></p>
    </form>

  </body>
</html>

```

از سوی دیگر، اگر لاگین با موفقیت انجام شود، یک پیام success در قالب index فلش می شود.

## Index.html

```
<!doctype html>
<html>

  <head>
    <title>Flask Message flashing</title>
  </head>
  <body>

    {% with messages = get_flashed_messages() %}
      {% if messages %}
        <ul>
          {% for message in messages %}
            <li>{{ message }}</li>
          {% endfor %}
        </ul>
      {% endif %}
    {% endwith %}

    <h1>Flask Message Flashing Example</h1>
    <p>Do you want to <a href = "{ url_for('login') }">
      <b>log in?</b></a></p>

  </body>
</html>
```

کد کامل فلشک پیام فلش در زیر آمده است:

## Flash.py

```
from flask import Flask, flash, redirect, render_template, request, url_for
app = Flask(__name__)
app.secret_key = 'random string'

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

@app.route('/login', methods = ['GET', 'POST'])
def login():
    error = None
```

```

if request.method == 'POST':
    if request.form['username'] != 'admin' or \
       request.form['password'] != 'admin':
        error = 'Invalid username or password. Please try again!'
    else:
        flash('You were successfully logged in')
        return redirect(url_for('index'))

return render_template('login.html', error = error)

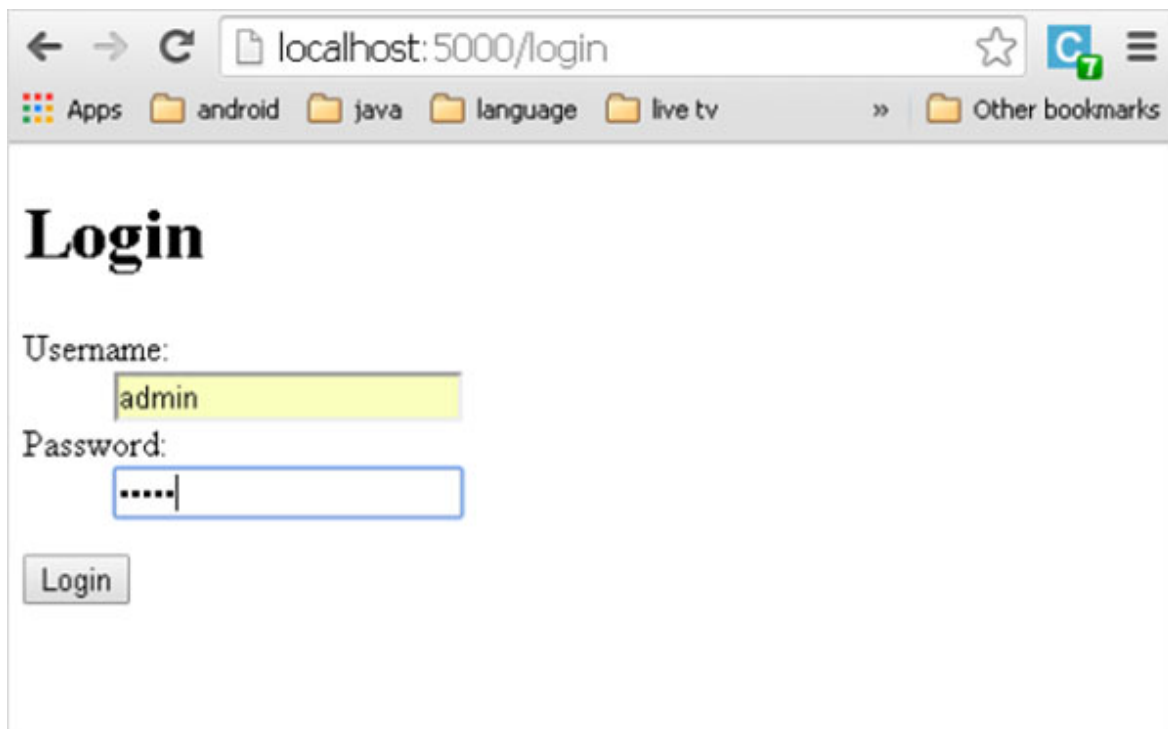
if __name__ == "__main__":
    app.run(debug = True)

```

پس از اجرای کدهای بالایی، در صفحه نمایش چیزی مشابه تصویر پایین خواهید دید:



هنگامی که بر روی لینک کلیک می شود، به صفحه ی لاگین منتقل خواهید شد. رمز عبور و پسورد خود را بنویسید:



← → ↻ localhost:5000/login ☆ C 7 ≡

Apps android java language live tv » Other bookmarks

# Login

Username:

Password:

Login

بر روی login کلیک کنید. یک پیام به نمایش در می آید. "You were successfully logged in"



← → ↻ localhost:5000 ☆ C 7 ≡

Apps android java language live tv » Other bookmarks

- You were successfully logged in

## Flask Message Flashing Example

Do you want to [log in?](#)

## ۱۷ آپلود فایل

مدیریت آپلود فایل در فلسک بسیار آسان است. برای این کار نیازمند یک فرم HTML است که enctype آن از نوع 'multipart/form-data' تنظیم شده است که فایل را به آدرس اینترنتی ارسال می کند. handler آدرس اینترنتی، فایل را از شی request.files می گیرد و در مکان دلخواه ذخیره می کند.

هر فایل آپلود شده ابتدا در یک مکان موقتی در سرور ذخیره می شود، پیش از این که واقعا در یک مکان نهایی ذخیره شود. اسم فایل نهایی می تواند هاردکود شده باشد یا از متغیر filename شی request.files[file] گرفته شود. هر چند که توصیه می شود که از ورژن امن آن با استفاده از تابع secure\_filename() این نام گرفته شود.

امکان تعیین متغیرهای پیش فرض مسیر پوشه ی آپلود و حداکثر اندازه فایل آپلود شده در تنظیمات configuration شی فلسک وجود دارد.

```
app.config['UPLOAD_FOLDER']
```

مسیر پوشه ی آپلود را تعیین می کند.

```
app.config['MAX_CONTENT_PATH']
```

ماکزیمم اندازه فایلی قابل آپلود را بر مبنای بایت مشخص می کند.

کد زیر دارای rule آدرس '/upload' است که 'upload.html' را که در پوشه ی templates است به نمایش در می آورد و نیز rule آدرس اینترنتی '/upload-file' که تابع uploader() را صدا می زند که خود پروسه ی آپلود را handle می کند.

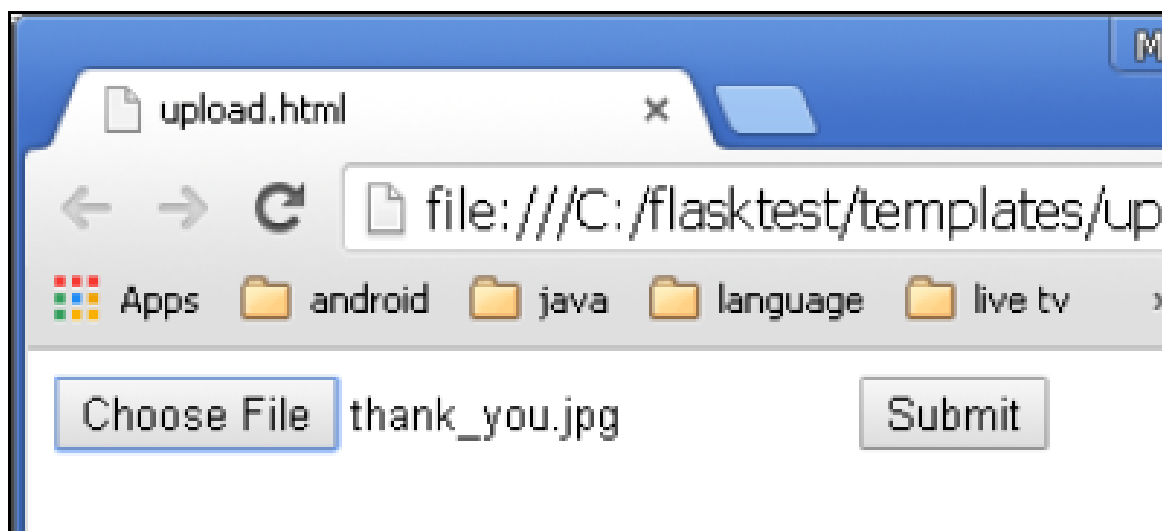
فایل 'upload.html' یک دکمه ی انتخاب فایل و یک دکمه ی submit دارد.

```
<html>
<body>

<form action = "http://localhost:5000/uploader" method = "POST"
  enctype = "multipart/form-data">
  <input type = "file" name = "file" />
  <input type = "submit"/>
</form>

</body>
</html>
```

احتمالا صفحه ای مشابه با تصویر زیر می بینید.



پس از انتخاب فایل بر روی submit کلیک کنید. متد POST فرم، آدرس اینترنتی '/upload\_file' را فراخوانی می کند. تابع uploader() عملیات ذخیره سازی را انجام می دهد.

کد پایتون برنامه فلسک در زیر مشاهده می شود.

```
from flask import Flask, render_template, request
from werkzeug import secure_filename

app = Flask(__name__)

@app.route('/upload')
def upload_file():
    return render_template('upload.html')

@app.route('/uploader', methods = ['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['file']
        f.save(secure_filename(f.filename))
        return 'file uploaded successfully'

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

## ۱۸ افزونه ها

اغلب فلسک به عنوان یک میکرو فریم ورک معرفی می شود چون امکانات و قابلیت های اصلی و هسته ای آن شامل WSGI و مسیریابی بر اساس Werkzeug و موتور قالب بر اساس Jinja2 است. در ضمن فلسک از کوکی و session و همچنین ابزارهای کمکی وب مانند JSON، فایل های استاتیک و غیره بهره می گیرد. واضح است که امکانات آن برای توسعه یک برنامه وب کامل نیست. اینجاست که افزونه های فلسک رخ می نمایند که به فرم ورک فلسک امکان گسترش می دهد.

تعداد بسیار زیادی افزونه فلسک موجود است. یک افزونه فلسک، نوعی ماژول پایتون است که امکانات خاصی به برنامه فلسک می دهد. Flask Extension Registry یک دایرکتوری شامل افزونه های موجود است. افزونه های مورد نیاز می توانند توسط ابزار pip دانلود شوند.

در این آموزش ما در مورد افزونه های مهم فلسک که در زیر آمده بحث خواهیم کرد.

Flask Mail رابط کاربری SMTP را برای برنامه Flask فراهم می کند.

Flask WTF نمایش و اعتبار سنجی فرم های WTForms ممکن می کند.

Flask SQLAlchemy قابلیت پشتیبانی از SQLAlchemy را به برنامه اضافه می کند.

Flask Sijax رابطی برای Sijax است. Sijax یک کتابخانه ی Python/jQuery است که استفاده از AJAX در برنامه های وب را

راحت می کند.

هر کدام از افزونه ها مستنداتی درباره نحوه استفاده اش آماده کرده است. از آنجا که افزونه یک ماژول پایتونی است، برای استفاده نیاز

دارد تا در برنامه import شود. افزونه های فلسک معمولاً به عنوان fooflask- نامیده می شوند. برای وارد کردن و import :

```
from flask_foo import [class, function]
```

برای نسخه های فلسک پایین تر از 0.7 ، می توانید از کد زیر استفاده کنید.

```
from flask.ext import foo
```

برای این کار، نیاز به فعال سازی یک ماژول سازگاری است که با اجرای کد زیر این اتفاق می افتد.



```
import flaskext_compat
flaskext_compat.activate()
from flask.ext import foo
```

## ۱۹ ایمیل

یک برنامه مبتنی بر وب معمولاً به امکان ارسال ایمیل به کاربرانش دارد. افزونه ی Flask-Mail امکان ساخت ساده ی یک رابط کاربری ساده برای هر سرور ایمیلی را فراهم می کند. در ابتدا باید با کمک ابزار pip افزونه Flask-Mail را نصب کنید.

```
pip install Flask-Mail
```

سپس افزونه Flask-Mail نیاز دارد تا مقادیر پارامترهای برنامه را که در زیر آمده است، تنظیم کند

- MAIL\_SERVER نام یا آی پی آدرس سرور ایمیل
- MAIL\_PORT شماره پورت سرور استفاده شده
- MAIL\_USE\_TLS رمزنگاری لایه امنیتی تبادل اطلاعات را فعال یا غیر فعال می کند
- MAIL\_USE\_SSL رمزنگاری لایه ی سوکت های امنیتی را فعال یا غیر فعال می کند
- MAIL\_DEBUG پشتیبانی از حالت دیباگ. برنامه های فلسک به طور پیش فرض در حالت دیباگ قرار دارند
- MAIL\_USERNAME نام کاربری فرستنده
- MAIL\_PASSWORD کلمه عبور فرستنده
- MAIL\_DEFAULT\_SENDER فرستنده پیش فرض را تنظیم می کند
- MAIL\_MAX\_EMAILS حداکثر تعداد نامه ای که باید فرستاده شود
- MAIL\_SUPPRESS\_SEND ارسال به صورت suppressed در صورتی که app.testing به عنوان true تنظیم شده باشد
- MAIL\_ASCII\_ATTACHMENTS اگر true تنظیم شده باشد، نام فایل ها به فرمت ASCII تبدیل می شود

در مائول Flask-Mail کلاس های مهم زیر تعریف شده اند.

### ۱.۱۹ کلاس Mail

این کلاس نیازمندی های ارسال ایمیل را مدیریت می کند. سازنده کلاس به شکل زیر است:

```
flask_mail.Mail(app = None)
```

سازنده کلاس، شی برنامه فلسک را به عنوان پارامتر می گیرد.

### ۲.۱۹ متد های کلاس Mail

- send() محتوای شی کلاس Message را می فرستد
- connect() ارتباط با هاست ایمیل را برقرار می کند
- send\_message() شی message را می فرستد.

### ۳.۱۹ کلاس Message

این کلاس، پیام ایمیل را کیسوله می کند. سازنده کلاس Message دارای چندین پارامتر است.

```
flask-mail.Message(subject, recipients, body, html, sender, cc, bcc,
    reply-to, date, charset, extra_headers, mail_options, rcpt_options)
```

## ۴.۱۹ متدهای کلاس Message

attach() یک پیوست به پیام اضافه می‌کند. این متد پارامترهای زیر را می‌گیرد:

filename نام فایلی که پیوست می‌شود

content\_type نوع فایل یا همان MIME type

data داده‌ی خام فایل

diposition وضعیت محتوا (در صورت وجود)

add\_recipient() یک گیرنده دیگر برای پیام اضافه می‌کند

## ۵.۱۹ یک مثال کاربردی

در مثال زیر، از سرور SMTP سرویس جیمیل گوگل به نام MAIL\_SERVER برای تنظیمات افزونه Flask-Mail استفاده می‌شود. قدم اول: مشابه کد زیر، کلاس های Mail و Message را از ماژول Flask-Mail وارد کنید

```
from flask_mail import Mail, Message
```

قدم دوم: سپس Flask-Mail بر اساس کد زیر تنظیم شود

```
app.config['MAIL_SERVER']='smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = 'yourId@gmail.com'
app.config['MAIL_PASSWORD'] = '*****'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
```

قدم سوم: یک شی از کلاس Mail بسازید

```
mail = Mail(app)
```

قدم چهارم: شی Message را درون تابع پایتونی که مربوط به آدرس اینترنتی ('/') است، تنظیم کنید.

```
@app.route("/")
def index():
    msg = Message('Hello', sender = 'yourId@gmail.com', recipients = ['id1@gmail.com'])
    msg.body = "This is the email body"
    mail.send(msg)
    return "Sent"
```

قدم پنجم: کل کد در زیر آمده است. اسکریپت زیر را در محیط شل پایتون اجرا کنید و آدرس اینترنتی http://localhost:5000/ را مشاهده کنید.

```
from flask import Flask
from flask_mail import Mail, Message

app =Flask(__name__)
mail=Mail(app)
```

```

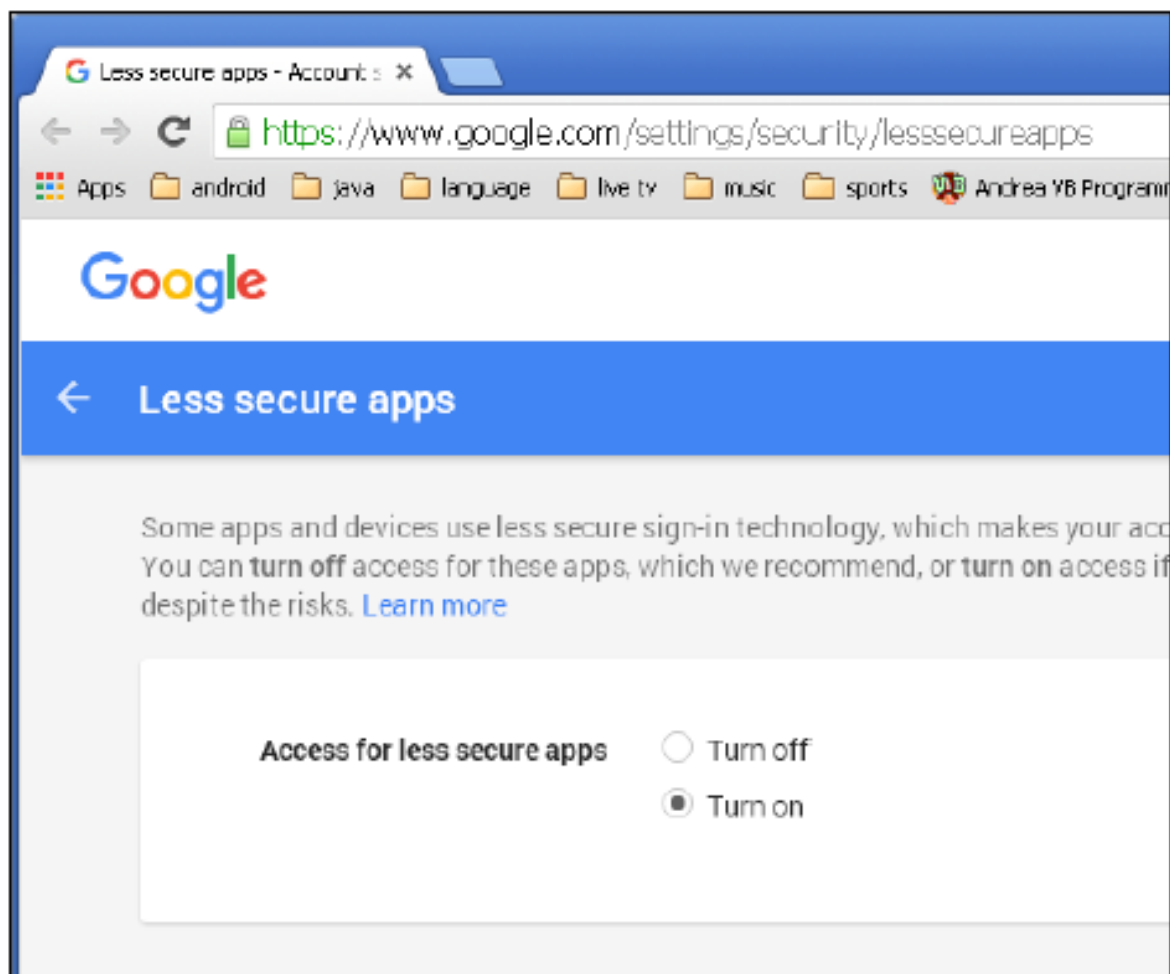
app.config['MAIL_SERVER']='smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = 'yourId@gmail.com'
app.config['MAIL_PASSWORD'] = '*****'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)

@app.route("/")
def index():
    msg = Message('Hello', sender = 'yourId@gmail.com', recipients = ['id1@gmail.com'])
    msg.body = "Hello Flask message sent from Flask-Mail"
    mail.send(msg)
    return "Sent"

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

```

توجه کنید که قابلیت built-insecurity جیمیل ممکن است اجازه لاگین شما را بگیرد. برای جلوگیری از این اتفاق در اکانت جیمیل خود لاگین شوید و با ورود به اینجا ، امنیت جیمیل خود را پایین بیاورید.



یکی از جنبه های مهم برنامه وب، ارائه یک رابط کاربری برای کاربر است. HTML تگ <FORM> را برای طراحی یک رابط کاربری فراهم می کند. عناصر یک فرم نظیر ورودی متن، دکمه رادیویی، انتخاب و غیره باید به خوبی استفاده شود. داده های وارد شده توسط کاربر به فرم یک درخواست Http و توسط متدهای GET و POST به اسکریپت سمت سرور ارسال می شود. اسکریپت سمت سرور باید عناصر فرم را از داده درخواست Http دریافت کند. در نتیجه عناصر فرم بایستی دو بار تعریف شوند، یک بار در HTML و بار دیگر در اسکریپت سمت سرور. یکی دیگر از ضعف های استفاده از فرم های HTML سختی (یا گاهی اوقات عدم امکان) نمایش عناصر فرم به شکل پویا و دینامیک است. HTML به خودی خود، راهی برای اعتبارسنجی ورودی کاربر فراهم نمی کند. اینجاست که WTForms که یک کتابخانه ی انعطاف پذیر نمایش و اعتبارسنجی فرم هاست، به کار می آید. افزونه ی Flask-WTF یک رابط کاربری ساده با استفاده از کتابخانه ی WTForms ایجاد می کند. با استفاده از Flask-WTF، ما می توانیم فیلدهای فرم را در اسکریپت پایتون تعریف کنیم و با استفاده از قالب های HTML نمایش دهیم. همچنین امکان اعتبارسنجی فیلدهای WTF امکان پذیر خواهد بود. بیا ببینیم این نسل پویای HTML چگونه کار می کند. ابتدا باید افزونه Flask-WTF را نصب کنید

```
pip install flask-WTF
```

بسته نصب شده شامل کلاس Form است که باید به عنوان کلاس والد فرم های کاربر استفاده شود. بسته WTforms شامل تعریف انواع فیلدهای فرم است. چند فرم استاندارد فیلد های فرم در زیر لیست شده است:

- TextField : نمایش عنصر <input type = 'text'> در فرم HTML
- BooleanField : نمایش عنصر <input type = 'checkbox'> در فرم HTML
- DecimalField نوعی TextField برای نمایش اعداد با مقادیر اعشاری
- IntegerField نوعی TextField برای نمایش اعداد صحیح
- RadioField : نمایش عنصر <input type = 'radio'> در فرم HTML
- SelectField نمایش عنصر select یا انتخاب در فرم
- TextAreaField : نمایش عنصر <testarea> در فرم HTML
- PasswordField : نمایش عنصر <input type = 'password'> در فرم HTML
- SubmitField : نمایش عنصر <input type = 'submit'> در فرم HTML

به عنوان مثال، یک فرم شامل یک فیلد متن می تواند به شکل زیر طراحی شود:

```
from flask_wtf import Form
from wtforms import TextField

class ContactForm(Form):
    name = TextField("Name Of Student")
```

علاوه بر فیلد "نام"، یک فیلد مخفی برای توکن CSRF به طور خودکار ایجاد می شود تا از حملات CSRF جلوگیری شود. نوعی از حملات CSRF با فریب کاربر وی را به سمت یک درخواست مخرب می برد و کنترل اکانت را از وی می گیرد. هنگام رندر و نمایش، این فیلد به شکل اسکریپت HTML متناسب با خود که در زیر آمده است، تبدیل می شود.

```
from flask import Flask, render_template
from forms import ContactForm

app = Flask(__name__)
```

```
app.secret_key = 'development key'

@app.route('/contact')
def contact():
    form = ContactForm()
    return render_template('contact.html', form = form)

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

بسته WTForms نیز شامل یک کلاس اعتبارسنج است. پیاده سازی سنجش اعتبار فیلدهای فرم مفید است. لیست زیر اعتبارسنج های معمولی که مورد استفاده قرار می گیرند را نشان می دهد.

- DataRequired چک می کند که آیا فیلد پر شده است یا نه
- Email چک می کند که آیا متن درون فیلد فرمت نرمال ایمیل را داراست یا نه
- IPAddress اعتبارسنجی آدرس آی پی در فیلد ورودی
- Length سنجش اینکه آیا طول رشته درون فیلد ورودی در محدوده تعریف شده است یا نه
- NumberRange سنجش اینکه آیا عدد وارد شده در محدوده تعریف شده قبلی هست یا نه
- URL اعتبارسنجی آدرس اینترنتی وارد شده در فیلد ورودی

اینک ما اعتبارسنجی 'DataRequired' را برای فیلد نام در فرم تماس اعمال خواهیم کرد.

```
name = TextField("Name Of Student",[validators.Required("Please enter your name.")])
```

تابع validate() از شی فرم، داده های فرم را اعتبارسنجی می کند و در صورتی که اعتبارسنجی به مشکل بخورد، ارورهای اعتبارسنجی پرتاب می کند. پیام های ارور به قالب HTML ارسال می شود. درون قالب، پیام ارور به طور خودکار به نمایش در می آید:

```
{% for message in form.name.errors %}
    {{ message }}
{% endfor %}
```

مثال کد زیر نمایش مفاهیم ارائه شده قبلی است. طراحی فرم Contact در زیر آمده است (فایل forms.py)

```
from flask_wtf import Form
from wtforms import TextField, IntegerField, TextAreaField, SubmitField, RadioField,
    SelectField

from wtforms import validators, ValidationError

class ContactForm(Form):
    name = TextField("Name Of Student",[validators.Required("Please enter
        your name.")])
    Gender = RadioField('Gender', choices = [('M', 'Male'), ('F', 'Female')])
    Address = TextAreaField("Address")

    email = TextField("Email",[validators.Required("Please enter your email address."),
        validators.Email("Please enter your email address.")])

    Age = IntegerField("age")
    language = SelectField('Languages', choices = [('cpp', 'C++'),
```

```
( 'py', 'Python' )])
submit = SubmitField("Send")
```

اعتبارسنج ها فیلد نام و ایمیل ارسال می شوند. در زیر اسکریپت برنامه فلسک آمده است (فایل formexample.py)

```
from flask import Flask, render_template, request, flash
from forms import ContactForm
app = Flask(__name__)
app.secret_key = 'development key'

@app.route('/contact', methods = ['GET', 'POST'])
def contact():
    form = ContactForm()

    if request.method == 'POST':
        if form.validate() == False:
            flash('All fields are required.')
            return render_template('contact.html', form = form)
        else:
            return render_template('success.html')
        elif request.method == 'GET':
            return render_template('contact.html', form = form)

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

کد قالب در زیر آمده است (فایل contact.html)

```
<!doctype html>
<html>
<body>

    <h2 style = "text-align: center;">Contact Form</h2>

    {% for message in form.name.errors %}
        <div>{{ message }}</div>
    {% endfor %}

    {% for message in form.email.errors %}
        <div>{{ message }}</div>
    {% endfor %}

    <form action = "http://localhost:5000/contact" method = post>
        <fieldset>
            <legend>Contact Form</legend>
            {{ form.hidden_tag() }}

            <div style = font-size:20px; font-weight:bold; margin-left:150px;>
                {{ form.name.label }}<br>
                {{ form.name }}
```

```

        <br>

        {{ form.Gender.label }} {{ form.Gender }}
        {{ form.Address.label }}<br>
        {{ form.Address }}
        <br>

        {{ form.email.label }}<br>
        {{ form.email }}
        <br>

        {{ form.Age.label }}<br>
        {{ form.Age }}
        <br>

        {{ form.language.label }}<br>
        {{ form.language }}
        <br>
        {{ form.submit }}
    </div>

</fieldset>
</form>

</body>
</html>

```

فایل formexample.py را اجرا کنید و آدرس اینترنتی <http://localhost:5000/contact> را باز کنید. فرم Contact مشابه زیر به نمایش در می آید.

← → ↻ localhost:5000/contact ☆ C ≡

Apps android java language live tv music sports >> Other bookmarks

## Contact Form

Contact Form

**Name Of Student**

**Gender**

- ☒ Male
- ☐ Female

**Address**

**Email**

**age**

**Languages**

C++ ▼

Send

در صورت وجود ارور، صفحه ای مشابه زیر خواهید دید.



← → ↻ localhost:5000/contact ☆ C ≡

Apps android java language live tv music sports >> Other bookmarks

## Contact Form

Please enter your name.  
Please enter your email address.

Contact Form

**Name Of Student**

**Gender**

- ☒ Male
- ☐ Female

**Address**

**Email**

**age**

**Languages**

C++ ▼

Send

اگر خطایی نباشد، صفحه ی 'success.html' به نمایش در خواهد آمد.

← → ↻ localhost:5000/contact ☆ C ≡

Apps android java language live tv music sports >> Other bookmarks

## Form posted successfully

## SQLite ۲۱

پایتون دارای پشتیبانی داخلی برای SQLite است. ماژول ۳ SQLite به همراه توزیع پایتون عرضه می شود. برای مشاهده آموزش همراه با جزئیات دیتابیس SQLite در پایتون، به اینجا مراجعه کنید. در این بخش می بینیم که چگونه برنامه Flask با SQLite در ارتباط قرار می گیرد.

یک دیتابیس SQLite با نام database.db بسازید و یک جدول students در آن بسازید

```
import sqlite3

conn = sqlite3.connect('database.db')
print "Opened database successfully";

conn.execute('CREATE TABLE students (name TEXT, addr TEXT, city TEXT, pin TEXT)')
print "Table created successfully";
conn.close()
```

برنامه فلسک ما سه تابع View دارد.

اولی تابع new\_student() به آدرس اینترنتی ('/addnew') مربوط شده است که یک فایل HTML شامل فرم اطلاعات دانشجو را نمایش می دهد.

```
@app.route('/enternew')
def new_student():
    return render_template('student.html')
```

کد HTML برای student.html در زیر آمده است:

```
<html>
<body>

<form action = "{{ url_for('addrec') }}" method = "POST">
    <h3>Student Information</h3>
    Name<br>
    <input type = "text" name = "nm" /><br>

    Address<br>
    <textarea name = "add" /></textarea><br>

    City<br>
    <input type = "text" name = "city" /><br>

    PINCODE<br>
    <input type = "text" name = "pin" /><br>
    <input type = "submit" value = "submit" /><br>
</form>

</body>
</html>
```

همانطور که مشاهده می شود، داده ی فرم به آدرس اینترنتی '/addrec' فرستاده می شود که با تابع addrec() ارتباط دارد.

تابع `addrec()` داده های فرم با متد POST را بازیابی می کند و در جدول `students` قرار می دهد. پیام مربوط به موفقیت یا ارور در عملیات قرار دادن اطلاعات در `result.html` به نمایش در می آید.

```
@app.route('/addrec', methods = ['POST', 'GET'])
def addrec():
    if request.method == 'POST':
        try:
            nm = request.form['nm']
            addr = request.form['add']
            city = request.form['city']
            pin = request.form['pin']

            with sql.connect("database.db") as con:
                cur = con.cursor()
                cur.execute("INSERT INTO students (name,addr,city,pin)
                             VALUES (?,?,?,?)",(nm,addr,city,pin) )

                con.commit()
                msg = "Record successfully added"
            except:
                con.rollback()
                msg = "error in insert operation"

            finally:
                return render_template("result.html",msg = msg)
                con.close()
```

کد HTML فایل `result.html` شامل عبارت `{{msg}}` است که نتیجه عملیات قرار دادن اطلاعات در دیتابیس را نمایش می دهد.

```
<!doctype html>
<html>
  <body>

    result of addition : {{ msg }}
    <h2><a href = "\>go back to home page</a></h2>

  </body>
</html>
```

این برنامه شامل یک تابع دیگر به نام `list()` است که به آدرس اینترنتی `/list` مربوط می شود که در این آدرس، سطرهای جدول (rows) با یک شی `MultiDict` که شامل تمامی رکوردهای جدول `students` است، پر می شود. این شی به قالب `list.html` ارسال می شود.

```
@app.route('/list')
def list():
    con = sql.connect("database.db")
    con.row_factory = sql.Row

    cur = con.cursor()
    cur.execute("select * from students")
```

```
rows = cur.fetchall();
return render_template("list.html", rows = rows)
```

فایل list.html یک قالب است که در مجموعه سطرها یکی یکی جلو می رود و داده ها را در یک جدول HTML به نمایش در می آورد.

```
<!doctype html>
<html>
  <body>

    <table border = 1>
      <thead>
        <td>Name</td>
        <td>Address</td>
        <td>city</td>
        <td>Pincode</td>
      </thead>

      {% for row in rows %}
        <tr>
          <td>{{row["name"]}}</td>
          <td>{{row["addr"]}}</td>
          <td> {{ row["city"]}}</td>
          <td>{{row['pin']}}</td>
        </tr>
      {% endfor %}
    </table>

    <a href = "/">Go back to home page</a>

  </body>
</html>
```

در نهایت، آدرس اینترنتی '/' صفحه ی home.html را نمایش می دهد که به عنوان نقطه ورود به برنامه عمل می کند.

```
@app.route('/')
def home():
    return render_template('home.html')
```

کد کامل برنامه ی Flask-SQLite در زیر آمده است:

```
from flask import Flask, render_template, request
import sqlite3 as sql
app = Flask(__name__)

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

@app.route('/enternew')
def new_student():
```

```

        return render_template('student.html')

@app.route('/addrec', methods = ['POST', 'GET'])
def addrec():
    if request.method == 'POST':
        try:
            nm = request.form['nm']
            addr = request.form['add']
            city = request.form['city']
            pin = request.form['pin']

            with sql.connect("database.db") as con:
                cur = con.cursor()

                cur.execute("INSERT INTO students (name,addr,city,pin)
                    VALUES (?,?,?,?)",(nm,addr,city,pin) )

                con.commit()
                msg = "Record successfully added"
            except:
                con.rollback()
                msg = "error in insert operation"

            finally:
                return render_template("result.html",msg = msg)
                con.close()

@app.route('/list')
def list():
    con = sql.connect("database.db")
    con.row_factory = sql.Row

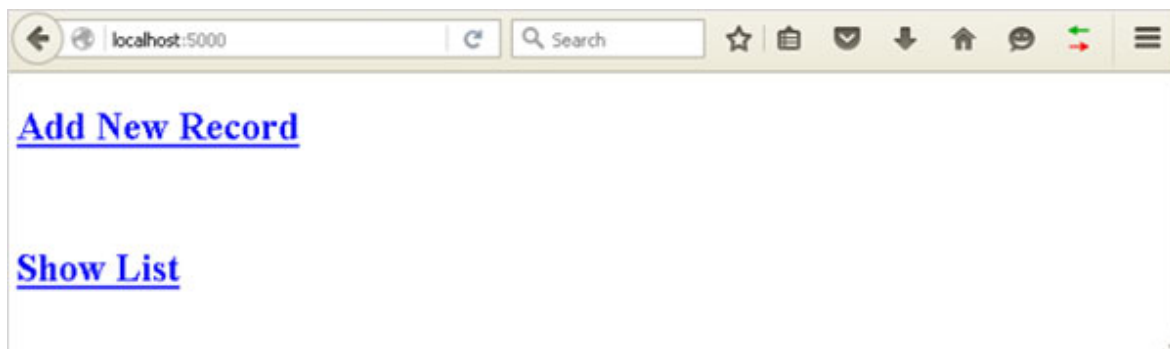
    cur = con.cursor()
    cur.execute("select * from students")

    rows = cur.fetchall();
    return render_template("list.html",rows = rows)

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

```

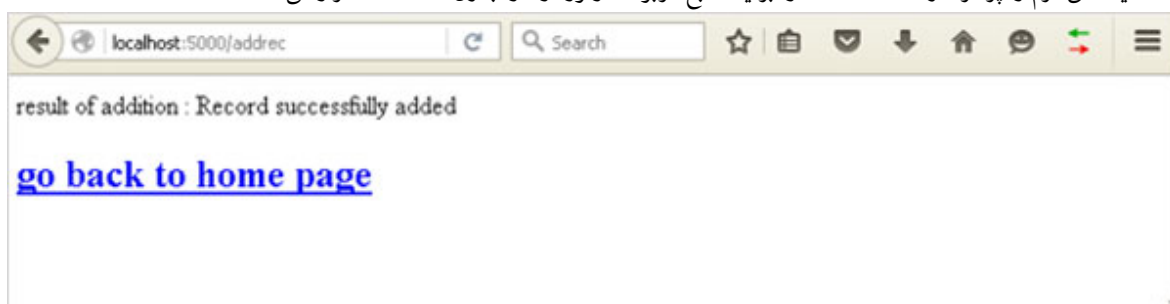
این اسکریپت را در شل پایتون اجرا کنید و زمانی که سرور توسعه اجرا شد، آدرس <http://localhost:5000/> را در مرورگر مشاهده کنید که باید یک منوی ساده مانند زیر نمایش داده شود:



برای باز کردن فرم اطلاعات دانشجویی، روی لینک Add New Record کلیک کنید.

A screenshot of a web browser window. The address bar shows 'localhost:5000/enternew'. The page has a heading 'Student Information'. Below it are four text input fields labeled 'Name', 'Address', 'City', and 'PINCODE'. At the bottom left is a 'submit' button.

فیلدهای فرم را پر کرده و دکمه submit را بزنید. تابع مربوطه، رکورد را در جدول students قرار می دهد.



به صفحه اصلی برگردید و بر روی لینک Show List کلیک کنید. جدولی که داده ساده ای را نمایش می دهد، قابل مشاهده است:

Name	Address	city	Pincode
Sunil	Nariman Point	Mumbai	400021
Rahul	Jubilee Hills	Hyderabad	500033
Ravi	Shivaji Nagar	Nanded	431602
Meenakshi	Hinjewadi	Pune	411057
Kiran	Ravinagar	Nagpur	440001
Kishore	Tilaknagar	Aurangabad	431001

[Go back to home page](#)

## ۲۲ SQLAlchemy

استفاده از SQL خام در برنامه های وب فلسک برای انجام عملیات CRUD می تواند خسته کننده باشد. SQLAlchemy که یک ابزار پایتونی ست، یک ORM قدرتمند است که به برنامه نویسان برنامه و انعطاف کامل استفاده از SQL را می دهد. Flask-SQLAlchemy یک افزونه فلسک است که پشتیبانی از SQLAlchemy را در برنامه های فلسک به شما می دهد.

ORM یا Object Relation Mapping چیست؟

بیشتر پلتفرم های زبان برنامه نویسی شی گرا هستند. از طرف دیگر داده در سرورهای RDBMS (مثلا یک مرکز پایگاه داده) به صورت جدولی ذخیره می شوند. ORM یک تکنیک نگاشت پارامترهای یک شی به ساختار جدول RDBMS اصلی است. یک ORM API متدهایی را برای عملیات CRUD بدون نیاز به نوشتن عبارات SQL فراهم می کند.

در این بخش، می خواهیم مطالعه ای بر روی تکنیک های ORM افزونه Flask-SQLAlchemy داشته باشیم و یک برنامه وب ساده بسازیم.

قدم اول: افزونه ی Flask-SQLAlchemy را نصب کنید

```
pip install flask-sqlalchemy
```

از این ماژول، کلاس SQLAlchemy را import یا وارد کنید

```
from flask_sqlalchemy import SQLAlchemy
```

قدم سوم: یک شی برنامه فلسک بسازید و یک آدرس اینترنتی برای جهت استفاده تنظیم کنید:

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///students.sqlite3'
```

قدم چهارم: سپس یک شی از کلاس SQLAlchemy بسازید و شی کلاس را به عنوان پارامتر به آن بدهید. این شی شامل توابع کمکی برای عملیات ORM است. همچنین یک کلاس Model با استفاده از مدل تعریف شده توسط کاربر ارائه می شود. در قطعه کد زیر، یک مدل students ایجاد شده است.

```
db = SQLAlchemy(app)
class students(db.Model):
    id = db.Column('student_id', db.Integer, primary_key = True)
    name = db.Column(db.String(100))
```

```

city = db.Column(db.String(50))
addr = db.Column(db.String(200))
pin = db.Column(db.String(10))

def __init__(self, name, city, addr, pin):
    self.name = name
    self.city = city
    self.addr = addr
    self.pin = pin

```

قدم پنجم: برای ساخت یا استفاده از دیتابیس اشاره شده در آدرس اینترنتی، تابع `create_all()` اجرا می شود:

```
db.create_all()
```

شی `Session` در `SQLAlchemy` تمامی عملیات های ماندگار شی `ORM` را مدیریت می کند. متدهای `session` زیر عملیات `CRUD` را اجرا می کنند.

```
db.session.add(model object)
```

یک رکورد در جدول نگاشته شده درج می کند.

```
db.session.delete(model object)
```

رکورد جدول را پاک می کند

```
model.query.all()
```

تمامی رکوردها را مطابق با کوئری `SELECT` بر می گرداند.

شما می توانید با استفاده از صفت `filter`، یک فیلتر برای رکوردهای بازگردانده شده ایجاد کنید. برای مثال، برای گرفتن رکوردهایی از جدول `students` که در آن `city = 'Hyderabad'` است، از عبارت زیر استفاده کنید:

```
Students.query.filter_by(city = "Hyderabad").all()
```

با این پیش زمینه، اکنون ما توابع `view` برنامه خود را جهت اضافه کردن داده `student` آماده می کنیم. نقطه شروع برنامه تابع `show_all()` است که به آدرس اینترنتی `'/'` مرتبط شده است. مجموعه رکورد جدول `students` به صورت پارامتر به قالب `HTML` فرستاده می شود. کد سمت سرور در قالب، رکورد را درون فرم جدول `HTML` نمایش می دهد.

```

@app.route('/')
def show_all():
    return render_template('show_all.html', students = students.query.all())

```

اسکرپت `HTML` قالب در فایل `show_all.html` به صورت زیر است:

```

<!DOCTYPE html>
<html lang = "en">
  <head></head>
  <body>

    <h3>
      <a href = "{ url_for('show_all') }">Comments - Flask
      SQLAlchemy example</a>
    </h3>

    <hr/>
    {% for message in get_flashed_messages() %}

```



```

    {{ message }}
{%- endfor %}

<h3>Students (<a href = "{ { url_for('new') } }">Add Student
    </a></h3>

<table>
    <thead>
        <tr>
            <th>Name</th>
            <th>City</th>
            <th>Address</th>
            <th>Pin</th>
        </tr>
    </thead>

    <tbody>
        {% for student in students %}
            <tr>
                <td>{{ student.name }}</td>
                <td>{{ student.city }}</td>
                <td>{{ student.addr }}</td>
                <td>{{ student.pin }}</td>
            </tr>
        {% endfor %}
    </tbody>
</table>

</body>
</html>

```

صفحه بالا شامل یک لینک به آدرس اینترنتی /new است که تابع new() به آن مرتبط است. با کلیک بر روی آن، فرم اطلاعات Student به نمایش در می آید. اطلاعات با استفاده از متد POST به همان آدرس اینترنتی ارسال می شود:

فایل new.html را در زیر می بینید:

```

<!DOCTYPE html>
<html>
    <body>

        <h3>Students – Flask SQLAlchemy example</h3>
        <hr/>

        {%- for category, message in get_flashed_messages(with_categories = true) %}
            <div class = "alert alert-danger">
                {{ message }}
            </div>
        {%- endfor %}

        <form action = "{ { request.path } }" method = "post">

```

```

<label for = "name">Name</label><br>
<input type = "text" name = "name" placeholder = "Name" /><br>
<label for = "email">City</label><br>
<input type = "text" name = "city" placeholder = "city" /><br>
<label for = "addr">addr</label><br>
<textarea name = "addr" placeholder = "addr"></textarea><br>
<label for = "PIN">City</label><br>
<input type = "text" name = "pin" placeholder = "pin" /><br>
<input type = "submit" value = "Submit" />
</form>

</body>
</html>

```

زمانی که متد http به صورت POST شناسایی می شود، اطلاعات فرم در جدول students ذخیره می شود و برنامه ضمن برگشتن به صفحه اصلی خود، اطلاعات اضافه شده را نمایش می دهد.

```

@app.route('/new', methods = ['GET', 'POST'])
def new():
    if request.method == 'POST':
        if not request.form['name'] or not request.form['city'] or not request.form['addr']:
            flash('Please enter all the fields', 'error')
        else:
            student = students(request.form['name'], request.form['city'],
                                request.form['addr'], request.form['pin'])

            db.session.add(student)
            db.session.commit()

            flash('Record was successfully added')
            return redirect(url_for('show_all'))
    return render_template('new.html')

```

در زیر کد کامل برنامه که در فایل app.py قرار دارد، مشاهده می شود.

```

from flask import Flask, request, flash, url_for, redirect, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///students.sqlite3'
app.config['SECRET_KEY'] = "random string"

db = SQLAlchemy(app)

class students(db.Model):
    id = db.Column('student_id', db.Integer, primary_key = True)
    name = db.Column(db.String(100))
    city = db.Column(db.String(50))
    addr = db.Column(db.String(200))
    pin = db.Column(db.String(10))

```

```

def __init__(self, name, city, addr, pin):
    self.name = name
    self.city = city
    self.addr = addr
    self.pin = pin

@app.route('/')
def show_all():
    return render_template('show_all.html', students = students.query.all() )

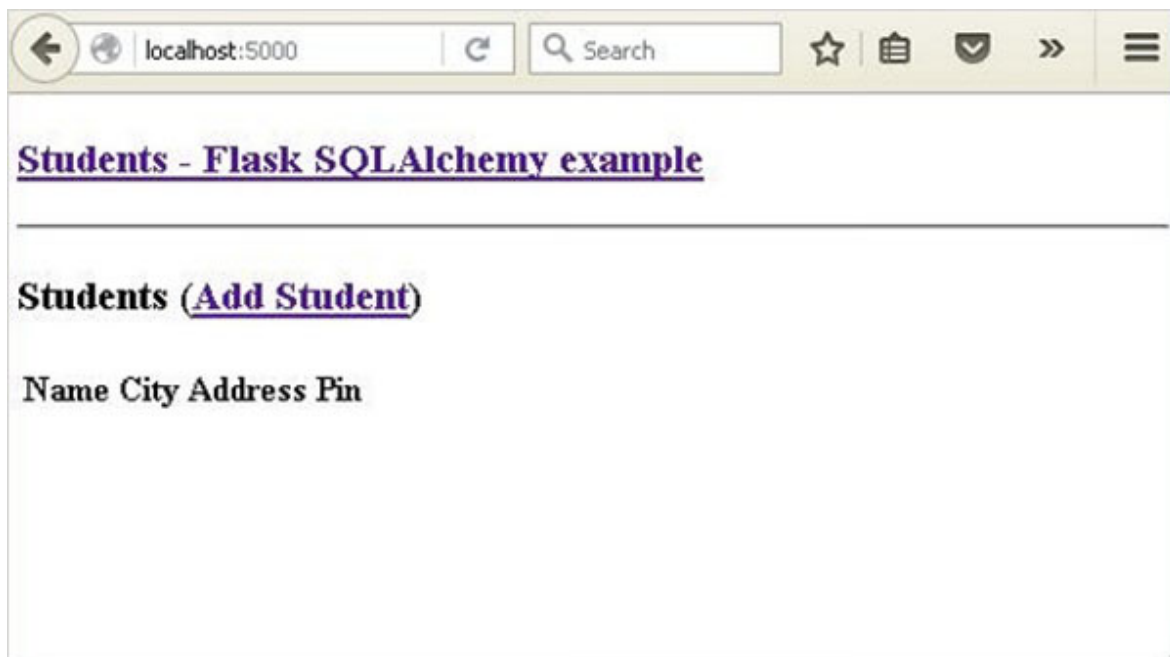
@app.route('/new', methods = ['GET', 'POST'])
def new():
    if request.method == 'POST':
        if not request.form['name'] or not request.form['city'] or not request.form['addr']:
            flash('Please enter all the fields', 'error')
        else:
            student = students(request.form['name'], request.form['city'],
                                request.form['addr'], request.form['pin'])

            db.session.add(student)
            db.session.commit()
            flash('Record was successfully added')
            return redirect(url_for('show_all'))
    return render_template('new.html')

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

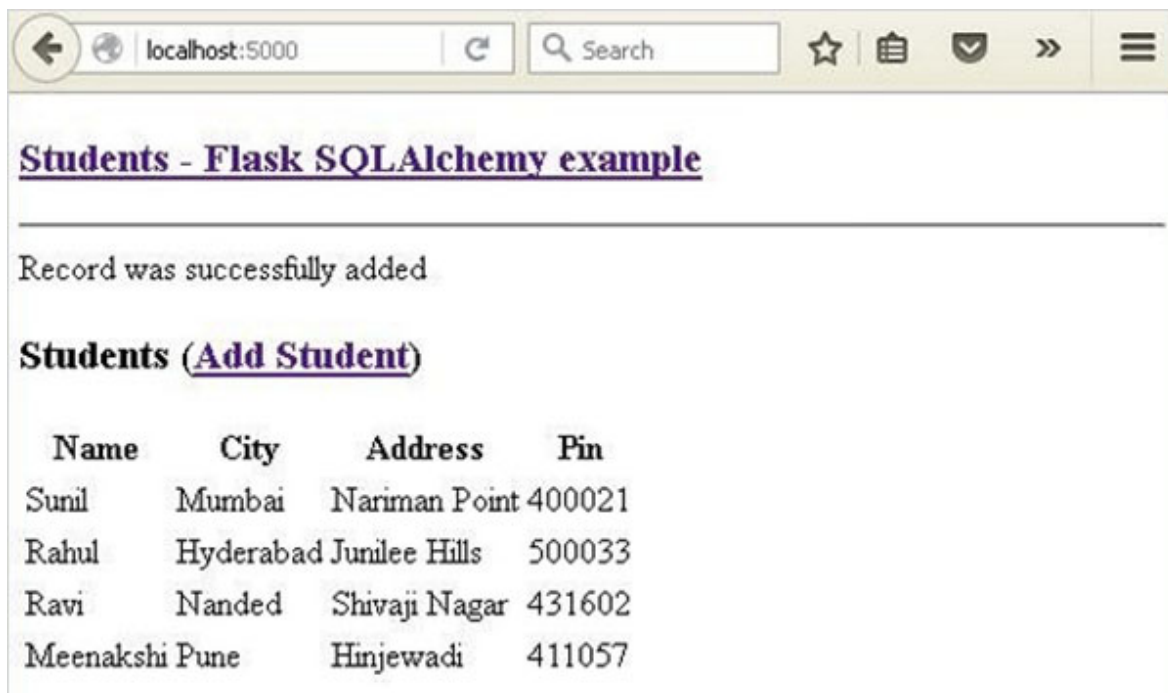
```

کد برنامه را در شل پایتون اجرا کنید و در مرورگر به آدرس اینترنتی <http://localhost:5000/> وارد شوید.



بر روی لینک Add Student کلیک کنید تا فرم اطلاعات Student به نمایش در بیاید

فرم را پر کنید و دکمه submit را بزنید. صفحه اصلی با اطلاعات وارد شده مجدداً نمایان می‌شود. می‌توانیم نتیجه خروجی را به شکل زیر ببینیم.



## ۲۳ سایجکس

سایجکس مخفف Simple Ajax و یک کتابخانه Python/jQuery است که برای کمک به شما در استفاده ساده از ای جکس در برنامه هایتان طراحی شده است و از jQuery.ajax برای ساخت درخواست های ای جکس استفاده می کند.

### ۱.۲۳ نصب

نصب Flask-Sijax ساده است:

```
pip install flask-sijax
```

### ۲.۲۳ تنظیمات

SIJAX\_STATIC\_PATH : آدرس استاتیکی که می خواهید فایل های جاوا اسکریپت سایجکس در آن قرار گیرند. آدرس پیش فرض در lstatic/js/sijax است. در این پوشه، فایل های sijix.js و json2.js قرار دارند.

SIJAX\_JSON\_URI : آدرسی که فایل استاتیک json2.js در آن بارگذاری می شود

سایجکس از جیسون برای ارسال اطلاعات میان مرورگر و سرور استفاده می کند به این معنی که یا مرورگر خودش از جیسون پشتیبانی می کند یا اینکه از فایل پشتیبانی json2.js استفاده می شود.

تابع هایی که پیش تر در آموزش ثبت شده اند، نمی توانند قابلیت های سایجکس را برای خود فراهم کنند چرا که نمی توانند به صورت پیش فرض با متد POST دسترسی پیدا کنند در حالی که سایجکس از درخواست های متد POST استفاده می کند.

برای آنکه یک تابع View (تابعی که به یک آدرس اینترنتی مرتبط می شود) را به گونه ای تبدیل که بتواند درخواست های سایجکس را پردازش کند، امکان آن را برای متد POST با کد زیر بدهید (@app.route('/url', methods = ['GET', 'POST']) یا اینکه از دکوراتور کمکی @flask\_sijax.route به شکل زیر بهره ببرید.

```
@flask_sijax.route(app, '/hello')
```

هر تابع پردازشگر سایجکس (مانند همین یکی) حداقل به طور خودکار یک پارامتر می‌گیرد. همانگونه که در متدهای شی 'self' را پاس می‌دهد. پارامتر `obj_response` روش تابع برای صحبت کردن با مرورگر است.

```
def say_hi(obj_response):  
    obj_response.alert('Hi there!')
```

زمانی که درخواست سایجکس شناسایی می‌شود، سایجکس آن را به شکل زیر پردازش می‌کند.

```
g.sijax.register_callback('say_hi', say_hi)  
return g.sijax.process_request()
```

## ۳.۲۳ برنامه Sijax

حداقل کد یک برنامه سایجکس چیزی شبیه زیر است:

```
import os  
from flask import Flask, g  
from flask_sijax import sijax  
  
path = os.path.join('.', os.path.dirname(__file__), 'static/js/sijax/')  
app = Flask(__name__)  
  
app.config['SIJAX_STATIC_PATH'] = path  
app.config['SIJAX_JSON_URI'] = '/static/js/sijax/json2.js'  
flask_sijax.Sijax(app)  
  
@app.route('/')  
def index():  
    return 'Index'  
  
@flask_sijax.route(app, '/hello')  
def hello():  
    def say_hi(obj_response):  
        obj_response.alert('Hi there!')  
    if g.sijax.is_sijax_request:  
        # Sijax request detected — let Sijax handle it  
        g.sijax.register_callback('say_hi', say_hi)  
        return g.sijax.process_request()  
        return _render_template('sijaxexample.html')  
  
if __name__ == '__main__':  
    app.run(debug = True)
```

زمانی که سایجکس یک درخواست (نوعی خاص از درخواست `jQuery.ajax()`) به سرور ارسال می‌کند. این درخواست با `g.sijax.is_sijax_request` در سمت سرور شناسایی می‌شود. در این حالت شما به سایجکس اجازه پردازش این درخواست را می‌دهید.

تمام تابع‌های ثبت شده با استفاده از `g.sijax.register_callback()` برای فراخوانی از سمت مرورگر به نمایش در می‌آیند. فراخوانی `g.sijax.process_request()` به سایجکس می‌گوید که تابع (پیش‌تر ثبت شده) مناسبی را اجرا کند و پاسخ را به مرورگر ارسال کند.

## ۲۴ توسعه

### ۱.۲۴ سرور خارجی قابل مشاهده

یک برنامه فلسک در سرور توسعه تنها روی کامپیوتری که محیط توسعه بر روی آن تنظیم شده است، قابل دسترسی است. این یک رفتار پیش فرض است، زیرا در حالت اشکال زدایی، یک کاربر می تواند کد دلخواهش را در کامپیوتر اجرا کند. اگر حالت دیباگ غیرفعال شود، سرور توسعه روی کامپیوتر محلی می تواند با تنظیم شماره هاست به صورت 0.0.0.0 برای کاربران قابل دسترسی باشد.

```
app.run(host = "0.0.0.0")
```

به این ترتیب، سیستم عامل شما به تمام IP های عمومی گوش می دهد.

### ۲.۲۴ گسترش

برای سوییچ کردن از یک محیط توسعه به یک محیط تولید کامل، یک برنامه نیاز به یک سرور وب واقعی دارد. بسته به چیزی که دارید، گزینه های مختلفی برای قرار دادن یک برنامه وب فلسک در سرور وب وجود دارد. برای برنامه های کوچک، قرار دادن آن در هر یک از پلتفرم های زیر را در نظر بگیرید که برای همگی آن ها اجرای مجانی را برای برنامه های کوچک ممکن است.

webfaction dotcloud Heroku

برنامه فلسک می تواند در این پلتفرم های ابری نیز اجرا شوند. به علاوه، امکان نصب و اجرای برنامه فلسک روی پلتفرم ابری گوگل نیز وجود دارد. سرویس Localtunnel امکان اشتراک گذاری برنامه تان روی هاست لوکال بدون درگیری با DNS و تنظیمات فایروال را فراهم می کند.

اگر شما مایل به استفاده از یک سرور وب اختصاصی به جای پلتفرم های اشتراکی بالا هستید، گزینه های زیر را مورد بررسی قرار دهید. `mod_wsgi` یک ماژول Apache است که یک رابط سازگار با WSGI برای میزبانی از برنامه های وب مبتنی بر پایتون را در سرورهای آپاچی فراهم می کند.

### ۳.۲۴ نصب مود WSGI

برای نصب مستقیم نسخه رسمی از PyPi، می توانید کد زیر را اجرا کنید:

```
pip install mod_wsgi
```

برای تایید اینکه نصب موفق آمیز بوده است، اسکریپت `mod_wsgi-express start-server` را با دستور اجرا کنید.

```
mod_wsgi-express start-server
```

این کار، `Apache/mod_wsgi` را روی پورت ۸۰۰۰ راه اندازی می کند. سپس می توانید با مراجعه به آدرس اینترنتی زیر از صحت نصب خود مطمئن شوید.

```
http://localhost:8000/
```

### ۴.۲۴ ساخت فایل wsgi.

اینجا باید یک فایل `yourapplication.wsgi` وجود داشته باشد. این فایل شامل کد `mod_wsgi` است که در هنگام راه اندازی اجرا می شود تا یک شی برنامه را بگیرد. برای بیشتر برنامه ها، این فایل همین فایل کفایت می کند.

```
from yourapplication import app as application
```

اطمینان حاصل کنید که yourapplication و تمام کتابخانه هایی که مورد استفاده هستند، در مسیر اجرا و بارگذاری پایتون قرار دارند.

## ۵.۲۴ تنظیمات Apache

برای انجام این کار، باید مسیر برنامه خود را به mod\_wsgi بدهید.

```
<VirtualHost *>
    ServerName example.com
    WSGIScriptAlias / C:\yourdir\yourapp.wsgi

    <Directory C:\yourdir>
        Order deny,allow
        Allow from all
    </Directory>

</VirtualHost>
```

## ۶.۲۴ حامل های مستقل WSGI

تعداد زیادی سرور محبوب به زبان پایتون وجود دارد که شامل برنامه های WSGI است. از جمله :

Gunicorn

Tornado

Gevent

Web Twisted

## ۲۵ FastCGI

FastCGI یک گزینه دیگر برای برنامه ی فلسک در سرورهای وب نظیر nginx ، lighttpd و Cherokee است.

## ۱.۲۵ تنظیمات FastCGI

اول شما نیاز به ساخت یک فایل سرور FastCGI دارید. بیایید اسمش را yourapplication.fcgi بگذاریم.

```
from flup.server.fcgi import WSGIServer
from yourapplication import app

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

nginx و نسخه های قدیمی تر lighttpd برای برقراری ارتباط با سرور FastCGI نیازمند یک سوکت هستند که به صورت آشکارا منتقل شود. برای این کار، نیاز به ارسال مسیر سوکت به WSGIServer دارید.

```
WSGIServer(application, bindAddress = '/path/to/fcgi.sock').run()
```



## ۲.۲۵ تنظیمات Apache

برای استقرار یک Apache پایه، فایل fcgi. شما در آدرس اینترنتی برنامه تان ظاهر می شود. مثلاً

```
example.com/yourapplication.fcgi/hello/
```

چندین راه برای تنظیم برنامه تان به گونه ای که فایل yourapplication.fcgi در مرورگر ظاهر نشود، وجود دارد.

```
<VirtualHost *>
    ServerName example.com
    ScriptAlias / /path/to/yourapplication.fcgi/
</VirtualHost>
```

## ۳.۲۵ تنظیمات lighttpd

تنظیمات پایه ای lighttpd چیزی شبیه زیر است:

```
fastcgi.server = ( "/yourapplication.fcgi" => ( (
    "socket" => "/tmp/yourapplication-fcgi.sock",
    "bin-path" => "/var/www/yourapplication/yourapplication.fcgi",
    "check-local" => "disable",
    "max-procs" => 1
) ) )

alias.url = (
    "/static/" => "/path/to/your/static"
)

url.rewrite-once = (
    "^(/static($|/.*))$" => "$1",
    "^(/.*)$" => "/yourapplication.fcgi$1"
)
```

به یاد داشته باشید که باید ماثول های alias و rewrite را از FastCGI فعال کنید. این تنظیمات برنامه را به آدرس /yourapplication متصل و مقید می کند.