

Qt  **GTK**

Graphical User Interface


 **Flask**
web development,
one drop at a time

django

Web Development

System Administration

LSB

 **THE LINUX FOUNDATION**


python




powered


Productivity

PyUNO


 The Document Foundation


Data Science

 **pandas**

 **matplotlib**

Machine Learning

 **PyTorch**

 **TensorFlow**

Framework

- **Веб-фреймворк** — это каркас для написания веб-приложений. Он определяет структуру, задаёт правила и предоставляет необходимый набор инструментов для разработки.

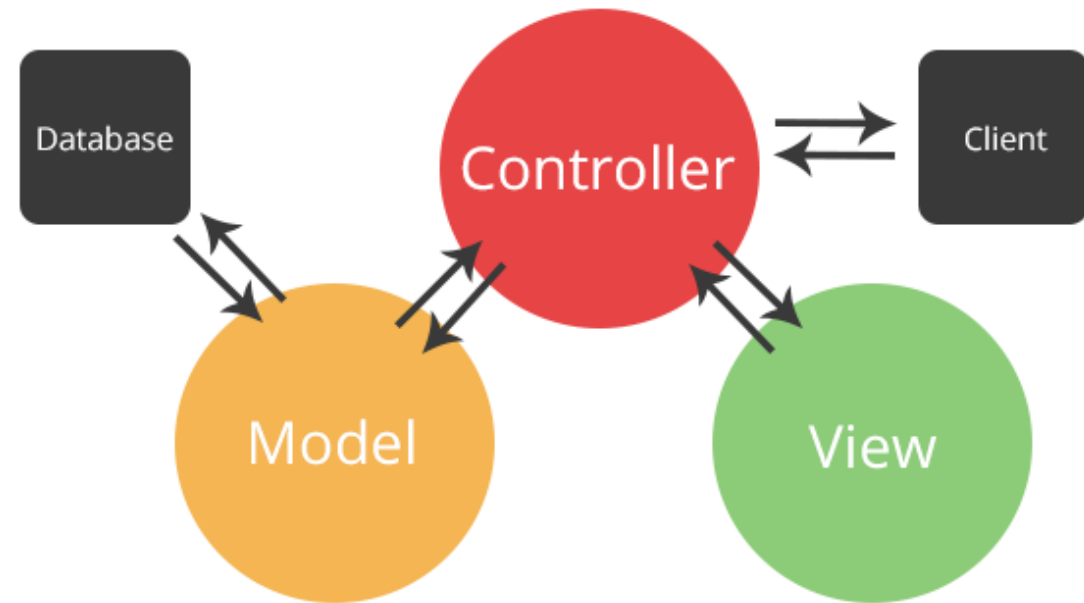
Архитектура веб-фреймворков

- **MVC** — **Модель**, **Представление** и **Контроллер** (**Model-View-Controller**) — три составляющих каждого веб-фреймворка.

Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

Покупку бутерброда можно описать через **MVC**:

- **Модель**: кухня, на которой повар делает сэндвич.
- **Представление**: готовый бутерброд, который вы с удовольствием едите.
- **Контроллер**: продавец или бармен, который принимает заказ и передаёт его на кухню.



FLASK



- Относится к категории микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

- **Flask** — фреймворк для создания веб-приложений на языке программирования *Python*.

Метрика	<u>Django</u> (Full-stack)	<u>Web2py</u>	<u>Flask</u>	<u>Bottle</u> (Micro-framework)	<u>CherryPy</u>
Звезды Github	46 528	1 832	48 385	6 594	1 130
Релизы Github	272	72	30	75	127
Вопросы Stack-overflow	217 030	2 094	32 621	1 371	1 300
Вакансии	42	0	18	4	0

Flask documentation

<https://flask.palletsprojects.com/en/2.0.x/> - original

<https://flask-russian-docs.readthedocs.io/ru/latest/quickstart.html> - russian

Installation

Create a project folder and a **venv** folder within

MacOS/Linux

```
$ mkdir myproject  
$ cd myproject  
$ python3 -m venv venv
```

Windows

```
> mkdir myproject  
> cd myproject  
> py -3 -m venv venv
```

Activate the environment

```
$ source .venv/bin/activate
```

```
> .venv\Scripts\activate
```

Install Flask

```
$ pip install Flask
```

A Minimal Application

```
from flask import Flask

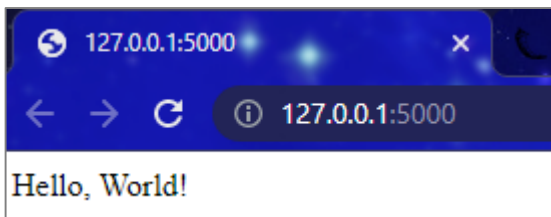
app = Flask(__name__)

@app.route("/")
def hello_world():
    return "Hello, World!"

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

Save as **hello.py** and run.

```
$ python hello.py
* Running on http://127.0.0.1:5000/
```



(browser)

1. Импортируем класс **Flask**
2. Создаём экземпляр класса. Первый аргумент - имя модуля или пакета приложения.
3. Используем декоратор **route()**, чтобы сказать **Flask**, какой из **URL** должен запускать нашу функцию.
4. Функция возвращает сообщение, которое мы хотим отобразить в браузере пользователя.
5. Для запуска локального сервера используем функцию **run()**.

Debug mode

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

В режиме отладки сервер перезагрузит сам себя при изменении кода

Опциональные атрибуты `app.run(..)`

`host, port, debug, options`

Routing

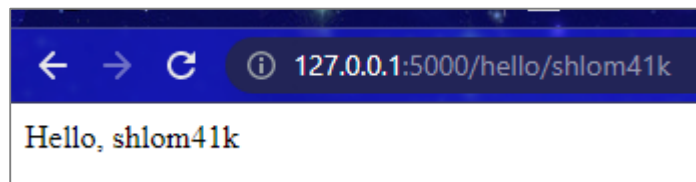
```
@app.route("/")  
def index():  
    return "Index Page"
```

```
@app.route("/hello")  
def hello():  
    return "Hello, World!"
```

- Современные веб-приложения используют «красивые» URL.
- Декоратор **route()** используется для привязки функции к URL.

Variable Rules

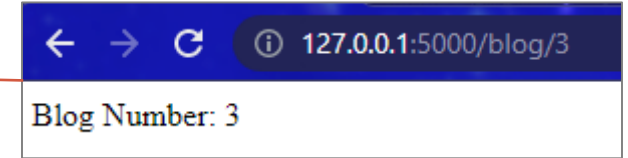
```
@app.route('/hello/<name>')  
def hello_name(name):  
    return "Hello, {}".format(name)
```



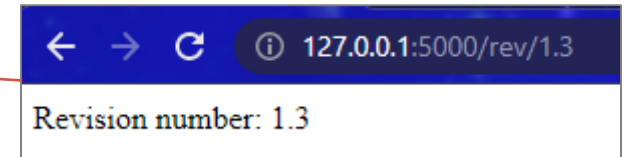
- Чтобы добавлять к адресу URL переменные части, можно эти части выделить как **<variable_name>**.
- Подобные части передаются в вашу функцию в качестве аргумента.

Variable Rules

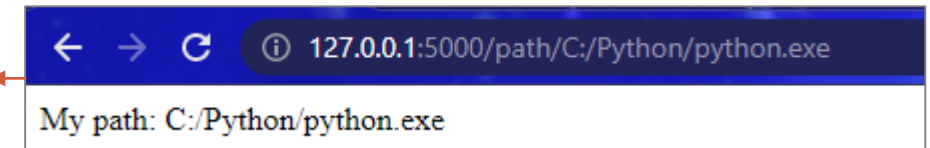
```
@app.route('/blog/<int:post_id>')
def show_blog(post_id):
    # show the blog with the given id, the id is an integer
    return "Blog Number: {}".format(post_id)
```



```
@app.route('/rev/<float:rev_number>')
def revision(rev_number):
    # show the revision version, the rev is an float
    return "Revision number: {}".format(rev_number)
```



```
@app.route('/path/<path:my_path>')
def my_path_to(my_path):
    # show the subpath after /path/
    return "My path: {}".format(my_path)
```



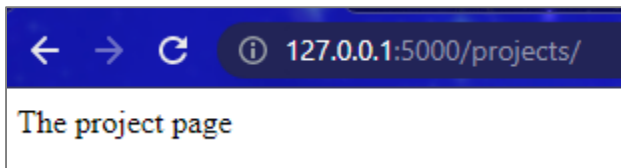
Converter types: string, int, float, path, uuid



string	(default) accepts any text without a slash
int	accepts positive integers
float	accepts positive floating point values
path	like string but also accepts slashes
uuid	accepts UUID strings

Unique URLs / Redirection Behavior

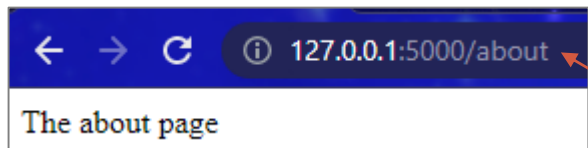
```
@app.route('/projects/')  
def projects():  
    return 'The project page'
```



! Есть разница в использовании “/” в определении URL !

- В первом случае URL имеет завершающую часть **projects/** со слэшем в конце.
- В данном случае, при доступе к URL без слэша, **Flask** перенаправит к каноническому URL с завершающим слэшем.

```
@app.route('/about')  
def about():  
    return 'The about page'
```



- Во втором случае, URL определен без косой черты.
- Доступ к URL с завершающей косой чертой будет приводить к появлению ошибки **404 «Not Found»**.



URL Building / Redirects

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

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

@app.route('/guest/<guest>')
def hello_guest(guest):
    return "Hello {} as Guest".format(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))
```

Генерация URL

- Для построения URL используется функция **url_for()**.
- Первый аргумента - имя функции.
- Именованные аргументы – переменная часть для URL.

Перенаправления

- Чтобы перенаправить пользователя в иную конечную точку, используйте функцию **redirect()**.

HTTP Methods

HTTP-метод сообщает серверу, что хочет сделать клиент с запрашиваемой страницей.

GET

- Браузер говорит серверу, чтобы он просто получил информацию, хранимую на этой странице, и отослал её. Возможно, это самый распространённый метод.

POST

- Браузер говорит серверу, что он хочет сообщить этому URL некоторую новую информацию, и что сервер должен убедиться, что данные сохранены и сохранены в единожды.

PUT

- Похоже на POST, только сервер может вызвать процедуру сохранения несколько раз, перезаписывая старые значения более одного раза.

HEAD

- Браузер просит сервер получить информацию, но его интересует только заголовки, а не содержимое страницы.

DELETE

- Удалить информацию, расположенную в указанном месте.

OPTIONS

- Обеспечивает быстрый способ выяснения клиентом поддерживаемых для данного URL методов.

The Request Object

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

```
@app.route('/success/<name>')
def success(name):
    return "Welcome, {}".format(name)

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

<http://127.0.0.1:5000/login?name=shlom41k>

FORM

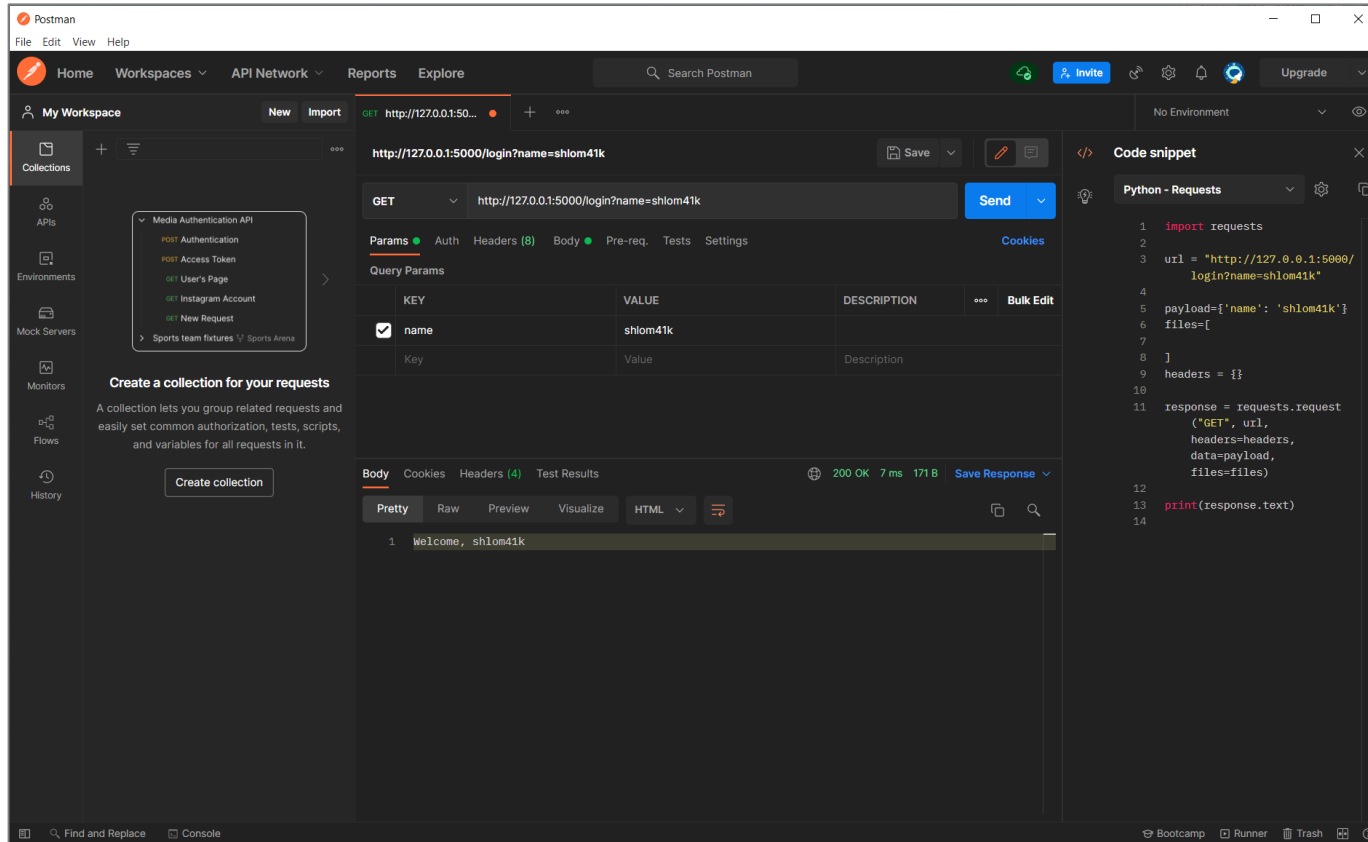
Для доступа к данным формы (данным, которые передаются в запросах типа **POST** или **PUT**), используется атрибут **form**.

ARGS

Для доступа к параметрам, представленным в URL (**?ключ=значение**), используется атрибут **args**.

Объект запроса: <https://flask-russian-docs.readthedocs.io/ru/latest/api.html#flask.request>

Postman



Postman — это HTTP-клиент для тестирования API. HTTP-клиенты тестируют отправку запросов с клиента на сервер и получение ответа от сервера.

С помощью **Postman** можно:

- составлять и отправлять HTTP-запросы к API;
- менять параметры запросов (например ключи авторизации и URL);
- добавлять при вызове API контрольные точки (фиксацию момента передачи данных) и т.д.

Download: <https://www.postman.com/downloads/>

About Postman: <https://blog.skillfactory.ru/glossary/postman/>

Postman - GET

The screenshot displays the Postman application interface. The main workspace shows a GET request to `http://127.0.0.1:5000/login?name=shlom41k`. The request is configured with the following parameters:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> name	shlom41k	
Key	Value	Description

The response is displayed in the 'Body' tab, showing a 'Welcome, shlom41k' message. The status is 200 OK, Time: 10 ms, Size: 171 B.

On the right side, the 'Code snippet' panel shows a Python script using the `requests` library to send the GET request and print the response text.

```
1 import requests
2
3 url = "http://127.0.0.1:5000/login?name=shlom41k"
4
5 payload={'name': 'shlom41k'}
6 files=[]
7
8 ]
9 headers = {}
10
11 response = requests.request
12     ("GET", url,
13     headers=headers,
14     data=payload,
15     files=files)
16
17 print(response.text)
```

Postman - POST

The screenshot displays the Postman application interface. The main workspace shows a POST request configured for the URL `http://127.0.0.1:5000/login`. The request body is set to `form-data` and contains a single key-value pair: `name` with the value `shlom41k`. The response status is `200 OK` with a time of `7 ms` and a size of `171 B`. The response body is displayed in the `Body` tab, showing `Welcome, shlom41k`.

Yellow arrows highlight the following elements:

- The `POST` method dropdown.
- The request URL `http://127.0.0.1:5000/login`.
- The `Body` tab selection.
- The `form-data` radio button.
- The `name` key and `shlom41k` value in the body table.
- The `Send` button.
- The `Code snippet` panel on the right, which contains a Python script for making a POST request.

The `Code snippet` panel shows the following Python code:

```
1 import requests
2
3 url = "http://127.0.0.1:5000/login"
4
5 payload={'name': 'shlom41k'}
6 files=[]
7
8 ]
9 headers = {}
10
11 response = requests.request
12 ("POST", url,
13 headers=headers,
14 data=payload,
15 files=files)
16
17 print(response.text)
```

Rendering Templates

```
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)
```



(browser)



Rendering Templates

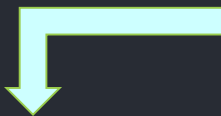
application.py

```
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)
```



index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>FlaskApp</title>
</head>
<body>
  <h1>Hello World!</h1>
  <h2>Welcome to FlaskApp!</h2>
</body>
</html>
```

- Для визуализации шаблона используется метод **render_template()**.
- **Flask** будет искать шаблоны в папке **templates**

```
app = Flask(__name__, template_folder="templates")
```

Module

```
/application.py
/templates
/index.html
```

Package

```
/application
/ __init__.py
/templates
/index.html
```


Rendering Templates. Передача параметров на шаблон

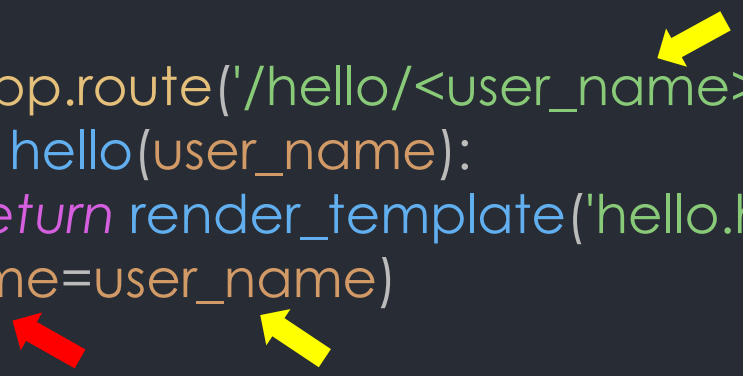
hello.py

```
from flask import Flask, render_template

app = Flask(__name__,
            template_folder="templates")

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

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




hello.html

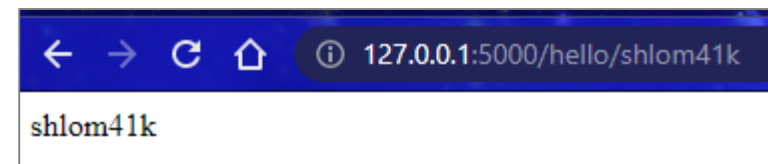
```
<!DOCTYPE html>
<html lang="en">

  <meta charset="UTF-8">
  {{name}}
<body>

</body>
</html>
```



(browser)



- Необходимо указать имя шаблона, а также переменные в виде именованных аргументов, которые вы хотите передать движку обработки шаблонов

Rendering Templates. Конструкции в шаблонах

marks.html

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

← → ↻ 🏠 ⓘ 127.0.0.1:5000/marks/40

Your result is fail

← → ↻ 🏠 ⓘ 127.0.0.1:5000/marks/80

Your result is pass!

subj.html

```
<!DOCTYPE html>
<html>
  <body>
    <table border = 1>
      {% for key, value in result.items() %}
        <tr>
          <th> {{ key }} </th>
          <td> {{ value }} </td>
        </tr>
      {% endfor %}
    </table>
  </body>
</html>
```

← → ↻ 🏠 ⓘ 127.0.0.1:5000/vals?math=33&phys=36&eng=24

math	33
phys	36
eng	24

Rendering Templates. Конструкции в шаблонах

constr.py

```
from flask import Flask, render_template, request

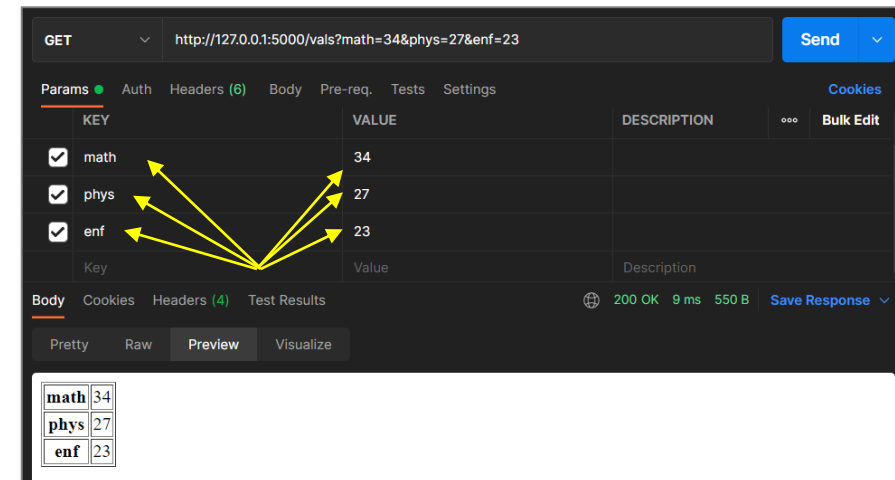
app = Flask(__name__)

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

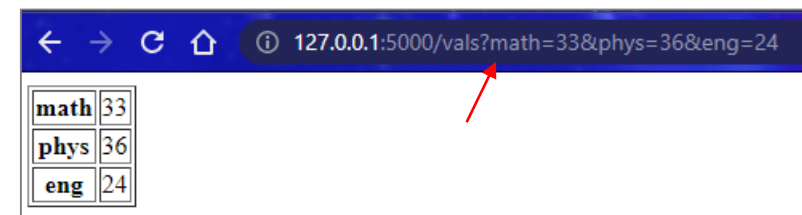
@app.route('/vals')
def vals():
    vals = request.args
    return render_template('subj.html', result=vals)

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

(postman)



(browser)



Rendering Templates. Отправка данных на шаблон

marks.html

```
<!DOCTYPE html>
<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>
```

Rendering Templates. Отправка данных на шаблон

to_template.py

```
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("subj.html", result=result)

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

(browser)

The image shows two browser screenshots. The top screenshot is at `localhost:5000` and displays a form with the following fields: Name (shlom41k), Physics (52), Chemistry (48), and Maths (76). A 'submit' button is at the bottom. A red arrow points to the 'submit' button. A green arrow points down to the second screenshot. The second screenshot is at `localhost:5000/result` and displays the rendered HTML table:

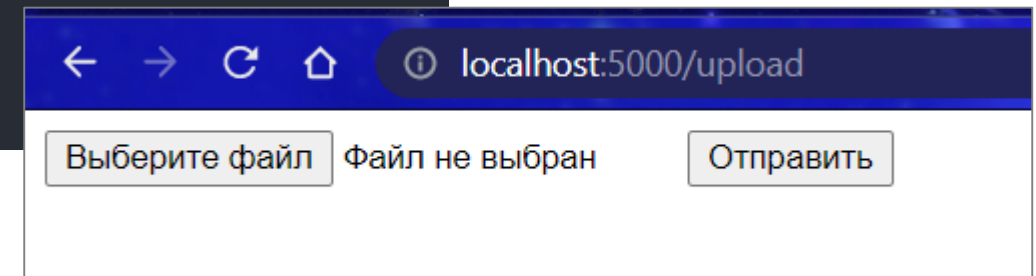
Name	shlom41k
Physics	52
Chemistry	48
Mathematics	76

File Uploads

upload.html

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <form action = "http://localhost:5000/uploader" method = "POST"
      enctype = "multipart/form-data">
      <input type = «File" name = "file" />
      <input type = "submit"/>
    </form>
  </body>
</html>
```

(browser)



- В HTML-форме необходимо установить атрибут **enctype="multipart/form-data"**, в противном случае браузер не передаст файл.

File Uploads

file_to_server.py

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

app = Flask(__name__)

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

@app.route('/uploader', methods=['GET', 'POST'])
def uploader_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)
```

- Загруженные на сервер файлы сохраняются в памяти в словаре **files**.
- Они ведут себя так же, как стандартный объект Python **file**, однако имеют метод **save()**, который позволяет сохранить файл внутри файловой системы сервера.
- Чтобы использовать имя файла на клиентской стороне для сохранения файла на сервере, используется функция **secure_filename()** из модуля **werkzeug.utils**