



# Python Practice

Ручное обновление курсов



# Python Practice

## Автор курса



Крементарь Ксения

Ведущий Python разработчик

Системный архитектор

в компании K-Solutions

# Python Practice

После урока обязательно



Повторите этот урок в видео формате на  
[ITVDN.com](http://ITVDN.com)



Проверьте как Вы усвоили данный материал на  
[TestProvider.com](http://TestProvider.com)

## Ручное обновление курсов

# Python Practice

## На этом уроке

1. Познакомимся с обработкой POST запросов во Flask приложениях.
2. Добавим ручное обновление курсов на отдельной странице web приложения Golden Eye.
3. Вспомним, что такое декораторы в Python.
4. Добавим простейшую защиту в проект с помощью декораторов.

# Python Practice

## Что же именно нужно сделать?

Очевидно, что для добавления возможности изменять курсы вручную необходима новая html страница, на которой можно будет вводить новые значения для выбранного курса валют. Следовательно, нам понадобится новый шаблон и новый url как минимум для рендеринга этого шаблона.

На новой html странице будут поля ввода и кнопка для сохранения изменений. И при нажатии этой кнопки будет происходить отправка запроса и сохранение нового значения курса валют. Следовательно, нужно будет добавить еще url для обработки этого запроса, в рамках которого будет происходить обновление в базе данных.

Как правило в Web разработке подобное поведение реализуется с помощью html форм, причем запрос на сохранение данных требует HTTP запроса типа POST.

# Python Practice

## Типы HTTP запросов

Тип HTTP-запроса (также называемый HTTP-метод) указывает серверу на то, какое действие мы хотим произвести — получить какие-то данные с сервера, добавить какие-то данные, удалить или обновить.

Существует несколько типов HTTP запросов:

- GET — получение ресурса
- POST — создание ресурса
- PUT — обновление ресурса
- DELETE — удаление ресурса и другие

Изначально (в начале 90-х) предполагалось, что клиент может хотеть от ресурса только одно — получить его, однако сейчас по протоколу HTTP можно создавать посты, редактировать профиль, удалять сообщения и многое другое.

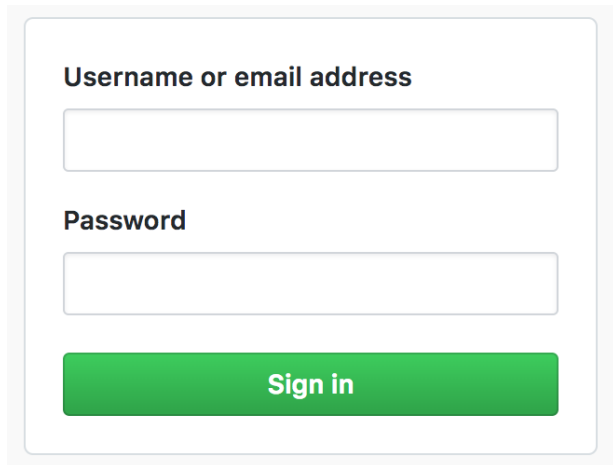
# Python Practice

## GET vs POST

Все запросы, которые мы до этого обрабатывали в нашем приложении — это GET запросы, так как это были запросы на получение информации.

Основное отличие от GET запросов это то, что в теле POST или PUT запроса могут быть переданы собственно новые данные.

В случае GET некоторые данные могут быть переданы в параметрах url (как мы указывали для фильтрации нужных курсов), но такая передача небезопасна и объем этих данных ограничен длиной url.



A login form with two input fields and a sign-in button. The first input field is labeled "Username or email address" and the second is labeled "Password". Below the input fields is a green button labeled "Sign in".

`https://site.com/path/resource?param1=value1&param2=value2`



# Python Practice

## Различные HTTP методы во Flask

Flask поддерживает обработку различных HTTP методов. Для указания ожидаемого метода необходимо указать параметр `methods` (список) при вызове метода `route` при указании `url`. По умолчанию все запросы ожидаются как GET.

Обращаю внимание, что `methods` — это именно список, и мы можем указать несколько методов, через запятую, что будет означать, что указанный `url` может обрабатывать запросы с разными HTTP методами.

```
@app.route("/", methods=["GET", "POST"])
def hello():
    return "Hello everybody!"
```

# Python Practice

## Добавление нового url и view функции

Для обработки запроса по сохранению курсов валют нужно будет использовать метод POST. Для рендеринга шаблона с формой ввода данных о курсе — метод GET.

Можно "повесить" эти методы один и тот же url: `/edit/<int:from_currency>/<int:to_currency>`, указав в параметре `methods = ["GET", "POST"]` два метода.

А при обработке запроса, в зависимости от метода производить разные действия — либо просто отображать html страницу с формой (GET запрос), либо производить сохранение данных о курсе в БД (POST запрос).

## Полезные атрибуты объекта request

Помимо уже известного нам параметра `request.args` в глобальном объекте `request` Flask запроса содержится много интересной информации:

- `request.method`, HTTP метод запроса
- `request.headers`, заголовки запроса
- `request.remote_addr`, ip адрес клиента
- `request.referrer`, реферер — url, с которого пользователь был перенаправлен на текущую страницу
- `request.form`, словарь `ImmutableMultiDict` с данными POST запроса

И много других параметров, полное описание доступно во Flask документации, <http://flask.pocoo.org/docs/1.0/api/#flask.Request>

# Python Practice

## HTML формы

Основные элементы HTML формы

- элемент `<form>` с указанием атрибутов — `action` и `method`.
- элемент `<input>` с указанием атрибута `type` и `name` — для полей ввода
- элемент `<input type="submit">` для кнопки подтверждения

```
<form action="/login" method="POST">  
  <input type="text" name="login">  
  <input type="password" name="password">  
  <input type="submit" name="commit" value="Sign in">  
</form>
```

# Python Practice

## HTML форма для обновления курса

Давайте создадим новый шаблон rate\_edit.html и добавим url для рендеринга этого шаблона.

```
<!DOCTYPE html>
<html>
<head>
  <title>Rate edit</title>
</head>
<body>
  <p>Rate: {{from_currency}} => {{to_currency}}</p>
  <form action="{{url_for('edit_xrate', from_currency=from_currency, to_currency=to_currency)}}"
        method="POST">

    <input type="text" name="new_rate">

    <input type="submit" value="Save">

  </form>
</body>
</html>
```

# Python Practice

## Защищаем наше приложение

Логично, что страница с ручным обновлением курсов валют, должна быть доступна только владельцу проекта Golden Eye. Но при текущей реализации, каждый пользователь интернет, который узнает о существовании url для обновления валют вручную, `/edit/<int:from_currency>/<int:to_currency>`, сможет изменить курсы как ему вздумается.

Также нам хотелось сделать недоступными для всеобщего просмотра логи общения с удаленными api, но при этом самим иметь возможность просматривать их.

Есть много вариантов реализации такой "выборочной" доступности отдельных url, мы рассмотрим один из них.

# Python Practice

## Декораторы в Python

**Декоратор** — это паттерн проектирования, который позволяет динамически добавлять новую функциональность к объекту, но не путем наследования.

В Python декораторы интересны тем, что реализованы на уровне синтаксиса языка и представляют собой очень полезный инструмент.

Благодаря тому, что в Python все является объектом, в том числе и функция, и функции могут являться аргументами других функций, можно декорировать функции дополнительным функционалом других функций. Подробнее рассмотрим на простейших примерах.

# Python Practice

## Декораторы в Python

```
def makebold(fn):
```

```
    def wrapped():
```

```
        return "<b>" + fn() + "</b>"
```

```
    return wrapped
```

```
def makeitalic(fn):
```

```
    def wrapped():
```

```
        return "<i>" + fn() + "</i>"
```

```
    return wrapped
```

```
@makebold
```

```
@makeitalic
```

```
def hello():
```

```
    return "hello habr"
```

```
print hello() ## выведет <b><i>hello habr</i></b>
```



# Python Practice

## Добавление фильтрации по ip

Для организации простейшей системы защиты некоторых страниц, давайте добавим проверку по ip — то есть страница с отображением html формы с вводом нового курса валют и обновление курса в БД будет доступно только для ip владельцев сайта.

Новый функционал, который мы хотим добавить к нашим некоторым view-функциям — это проверка, что ip адрес запроса находится в списке разрешенных ip адресов. Список с разрешенными ip адресами добавим в конфигурацию приложения и напомним новую функцию, которая будет осуществлять проверку. Если ip адрес не из списка разрешенных, возвращать ошибку 403 с помощью Flask функции abort. А затем просто декорируем нужные нам view-функции и проверим в браузере.

# Python Practice

## Декоратор check\_ip

```
from functools import wraps
from flask import request, abort
from config import IP_LIST
from app import app
import controllers

def check_ip(func):
    @wraps(func)
    def checker(*args, **kwargs):
        if request.remote_addr not in IP_LIST:
            return abort(403)

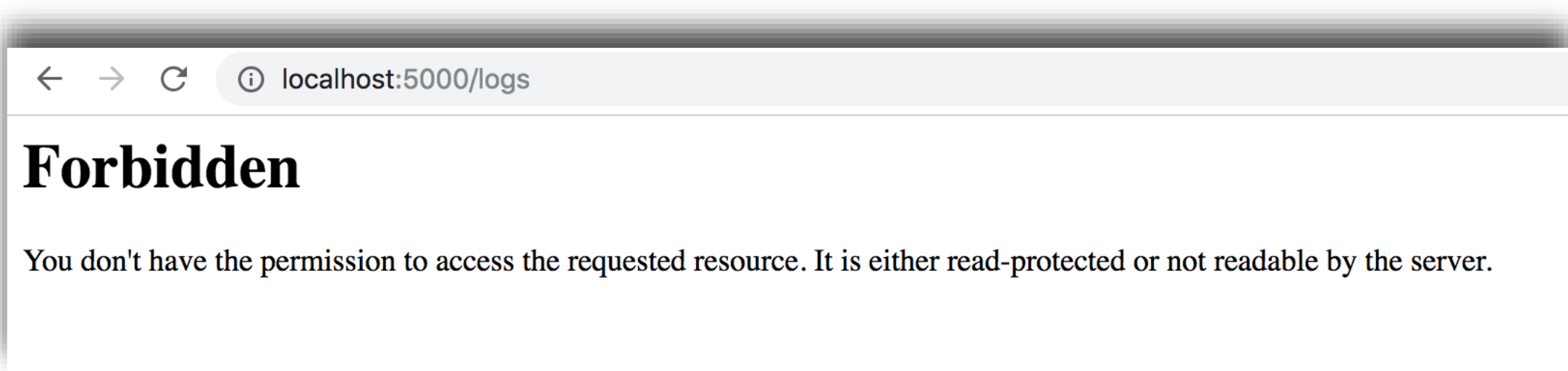
        return func(*args, **kwargs)

    return checker

@app.route("/logs")
@check_ip
def view_logs():
    return controllers.ViewLogs().call()
```

# Python Practice

## Ошибка 403 Forbidden



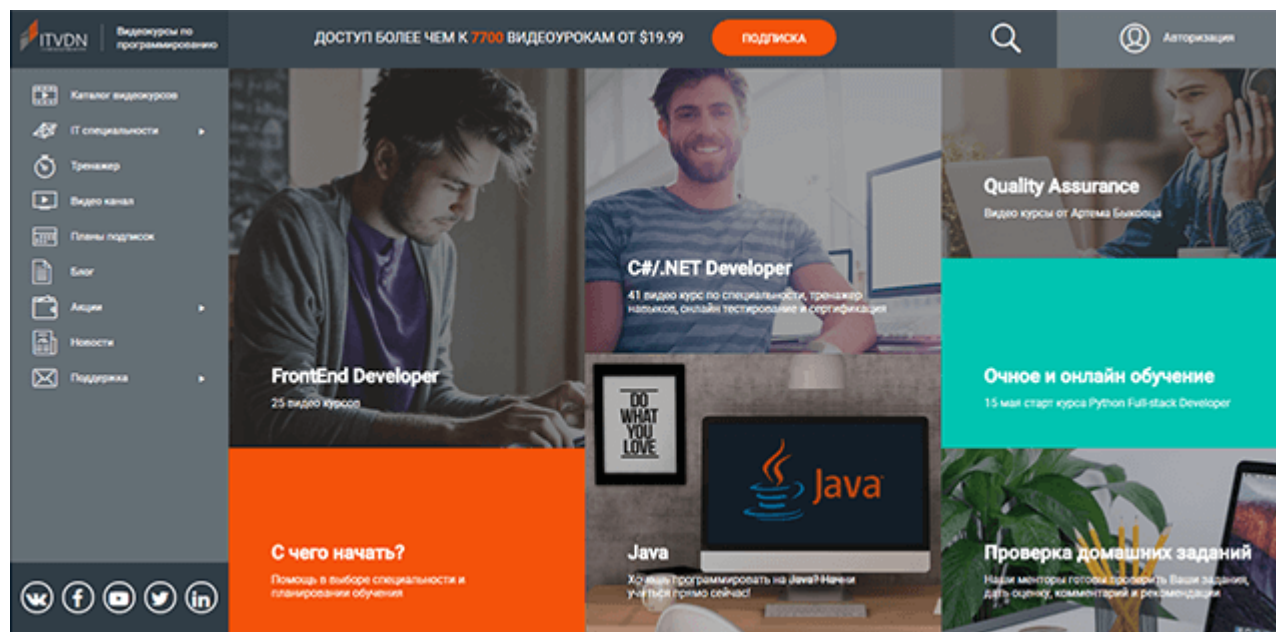
# Python Practice

## Что дальше?

На следующем уроке мы рассмотрим различные варианты организации периодически запускаемых задач в Python, выберем подходящий и добавим автоматическое обновление курсов в проект Golden Eye, проверим его работу.

# Смотрите наши уроки в видео формате

ITVDN.com



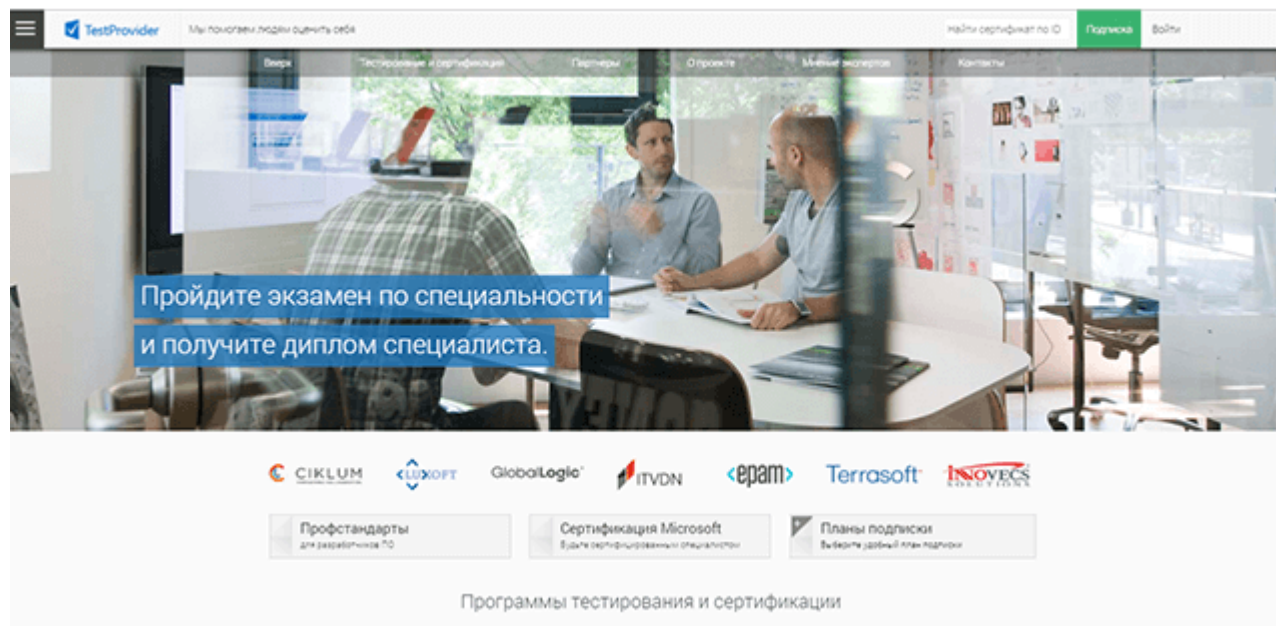
Посмотрите этот урок в видео формате на образовательном портале [ITVDN.com](http://ITVDN.com) для закрепления пройденного материала.

Курсы записаны сертифицированными тренерами, которые работают в учебном центре CyberBionic Systematics и другими высококвалифицированными разработчиками.



# Проверка знаний

TestProvider.com



TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

После каждого урока проходите тестирование для проверки знаний на [TestProvider.com](https://testprovider.com)

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.



# Python Practice

Q&A

# Информационный видеосервис для разработчиков программного обеспечения

