

 **ТЕХНОСФЕРА**

Реализация API

Антон Кухтичев





Напоминание отметиться на портале

Иначе плохо всё будет.



Содержание занятия

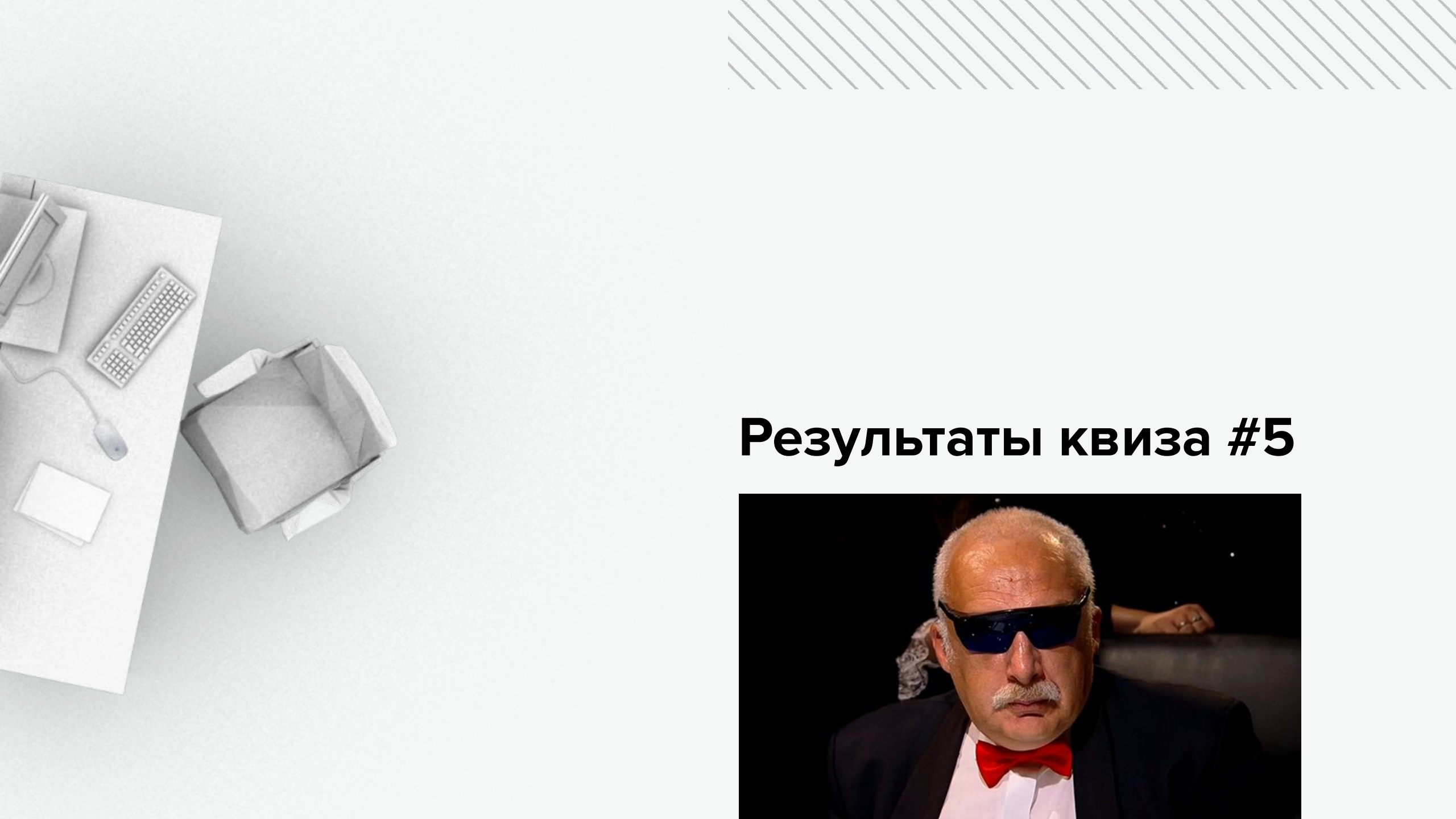
1. Квиз #6
2. Результаты квиза #5
3. API;
4. Текстовые протоколы;
5. Двоичные протоколы;
6. REST и RPC;
7. Формы;
8. Валидация форм;
9. Django Rest Framework;
10. Домашнее задание



Квиз #6

Квиз N°6

???



Результаты квиза #5





Немного статистики

Абсолютные чемпионы: **Османов А. С. (7/7), Селезнев Павел (7/7), Солодухов Артемий (7/7)**

- Оценка отлично (6-7 баллов): 25.00%
- Оценка хорошо (5 баллов): 25.00%
- Оценка удовлетворительно (4 баллов): 32.14%
- Оценка неудовлетворительно (<4 баллов): 17.86%

Вопрос №1

С помощью какой команды изменить БД в консольной утилите psql?

- 1) `\l <db name>;`
- 2) `change <db name>;`
- 3) `\dt <db name>;`
- 4) `\c <db name>;`

Вопрос N°2

Как выбрать первую строку (кортеж, tuple) из таблицы movie?

1. **SELECT * FROM movie LIMIT 1;**
2. SELECT * FROM movie SIZE 1;
3. SELECT * FROM movie WHERE id = 1;
4. SELECT * FROM movie GET 1;

Вопрос N°3

Какого типа данных нет в PostgreSQL?

1. serial
2. json
3. biginteger
- 4. tinyinteger**
5. money
6. integer

Вопрос №4

Есть уже база данных, теперь хотим подключить её к Django, в какую переменную её нужно прописать в `settings.py` вместе с пользователем, паролем, хостом и портом?

1. POSTGRESQL
- 2. DATABASES**
3. VOLUMES
4. STORAGES

Вопрос N°5

От какого класса нужно наследовать модели в Django?

1. **django.db.models.Model**
2. django.db.models.User
3. django.db.models.ORM
4. object

Вопрос N°6

Какое поле автоматически создаётся при наследовании от специальной модели (из вопроса 5)?

1. __id
2. **id**
3. _id
4. fk

Вопрос №7

Есть класс-модель User и Chat, описывающий соответственно пользователей и чаты. У User может быть несколько чатов. Какое из нижеперечисленных полей в классе User позволит для одного чата иметь несколько участников?


1. ForeignKey
2. OneToOneField
3. **ManyToManyField**



API



Application programming interface (API)



Описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.



Виды совместимости приложений

- Обратная совместимость;
- Прямая совместимость;



Виды совместимости приложений

- **Обратная совместимость** — более новый код способен читать данные, записанные более старым;
- **Прямая совместимость** — более старый код способен читать данные, записанные более новым.



REST (REpresentational State Transfer)

REST API подразумевает под собой простые правила:


- Каждый URL является ресурсом;
- При обращении к ресурсу методом GET возвращается описание этого ресурса;
- Метод POST добавляет новый ресурс;
- Метод PUT изменяет ресурс;
- Метод DELETE удаляет ресурс.

RESTful

- Конечные точки в URL – имя существительное, не глагол;
 - + /posts/
 - /getPosts/
- Используйте множественное число для названия своих REST сервисов;
- Документирование программного обеспечения является общей практикой для всех разработчиков;
- Версионность
 - URI версии.
 - Мультимедиа версии.



JSON-RPC



JSON-RPC (JavaScript Object Notation Remote Procedure Call — JSON-вызов удалённых процедур) — протокол удалённого вызова процедур, использующий JSON для кодирования сообщений.

JSON-RPC

Формат входного запроса:

- `method` — строка с именем вызываемого метода;
- `params` — массив объектов, которые должны быть переданы методу, как параметры;
- `id` — значение любого типа, которое используется для установки соответствия между запросом и ответом.

JSON-RPC

Формат ответа:

- `result` — данные, которые вернул метод. Если произошла ошибка во время выполнения метода, это свойство должно быть установлено в `null`;
- `error` — код ошибки, если произошла ошибка во время выполнения метода, иначе `null`;
- `id` — то же значение, что и в запросе, к которому относится данный ответ.

JSON-RPC

Пример запроса:

```
{ "method": "echo", "params": ["Hello JSON-RPC"], "id": 1 }
```

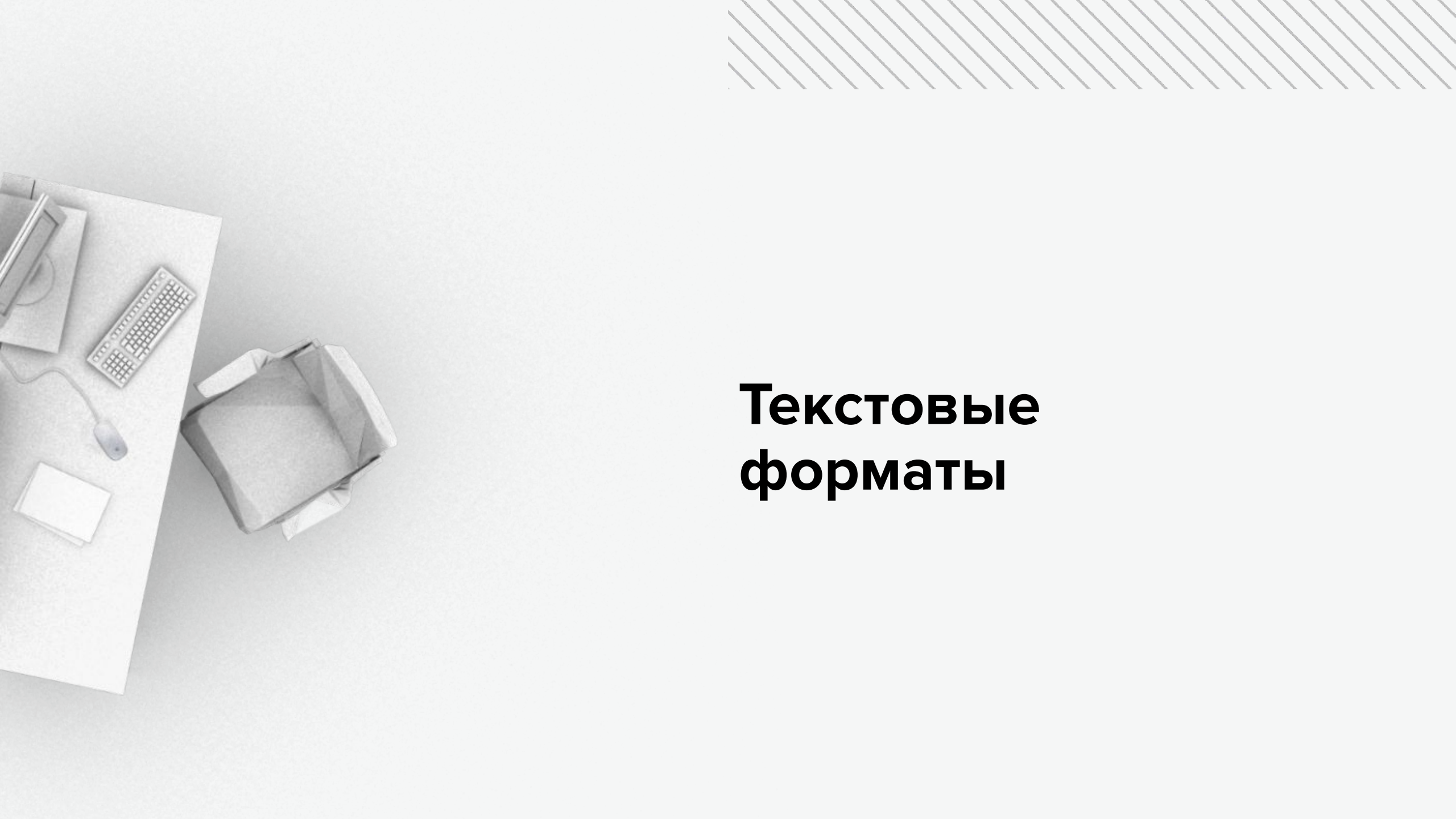
Пример ответа:

```
{ "result": "Hello JSON-RPC", "error": null, "id": 1 }
```




Форматы передачи данных

- Текстовые форматы (JSON, XML, CSV);
- Бинарный формат (Apache Thrift, Protocol Buffers);



Текстовые форматы

Формат CSV

- Каждая строка файла — это одна строка таблицы.
- Разделителем значений колонок является символ запятой (,)
- Однако на практике часто используются другие разделители.

1997,Ford,E350,"ac, abs, moon",3000.00

1999,Chevy,"Venture «Extended Edition»","",4900.00

1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00

Формат XML

XML (eXtensible Markup Language) – язык разметки, позволяющий стандартизировать вид файлов-данных, используемых компьютерными программами, в виде текста, понятного человеку.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Формат XML

- Синтаксис XML избыточен;
- XML не содержит встроенной в язык поддержки типов данных;
- + Есть схема;
- + Человекочитаемый.

Формат JSON

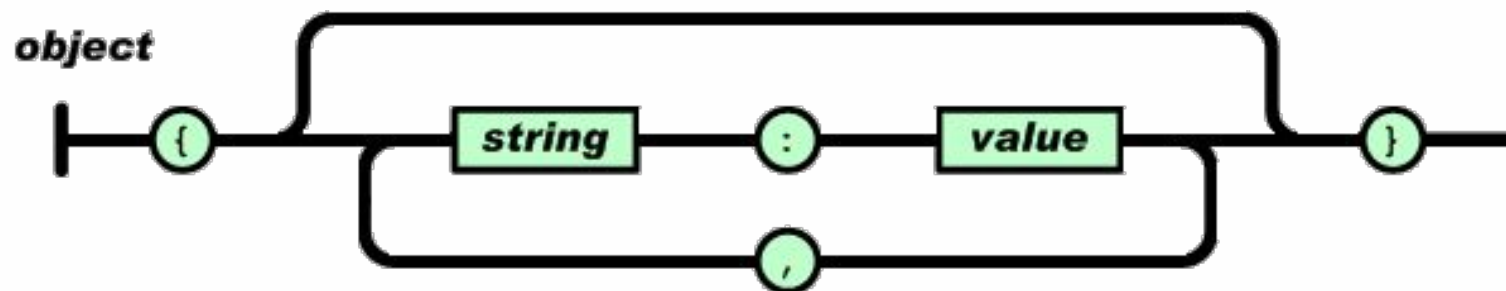
JSON (JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript.

```
{  
    "first_name": "Иван",  
    "last_name": "Иванов",  
    "phone_numbers": [  
        "812 123-1234",  
        "916 123-4567"  
    ]  
}
```

Формат JSON

JSON основан на двух структурах данных:

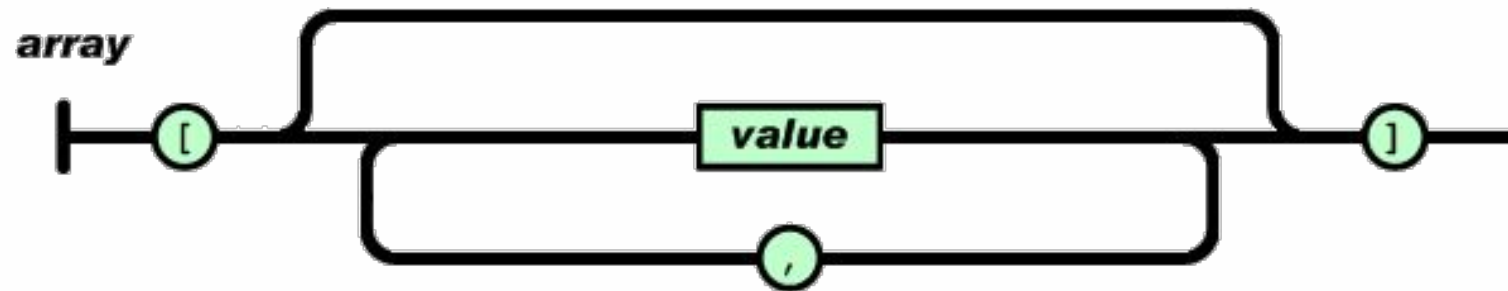
1. Коллекция пар ключ/значение. В разных языках, эта концепция реализована как объект, запись, структура, словарь, хэш, именованный список или ассоциативный массив;



Формат JSON

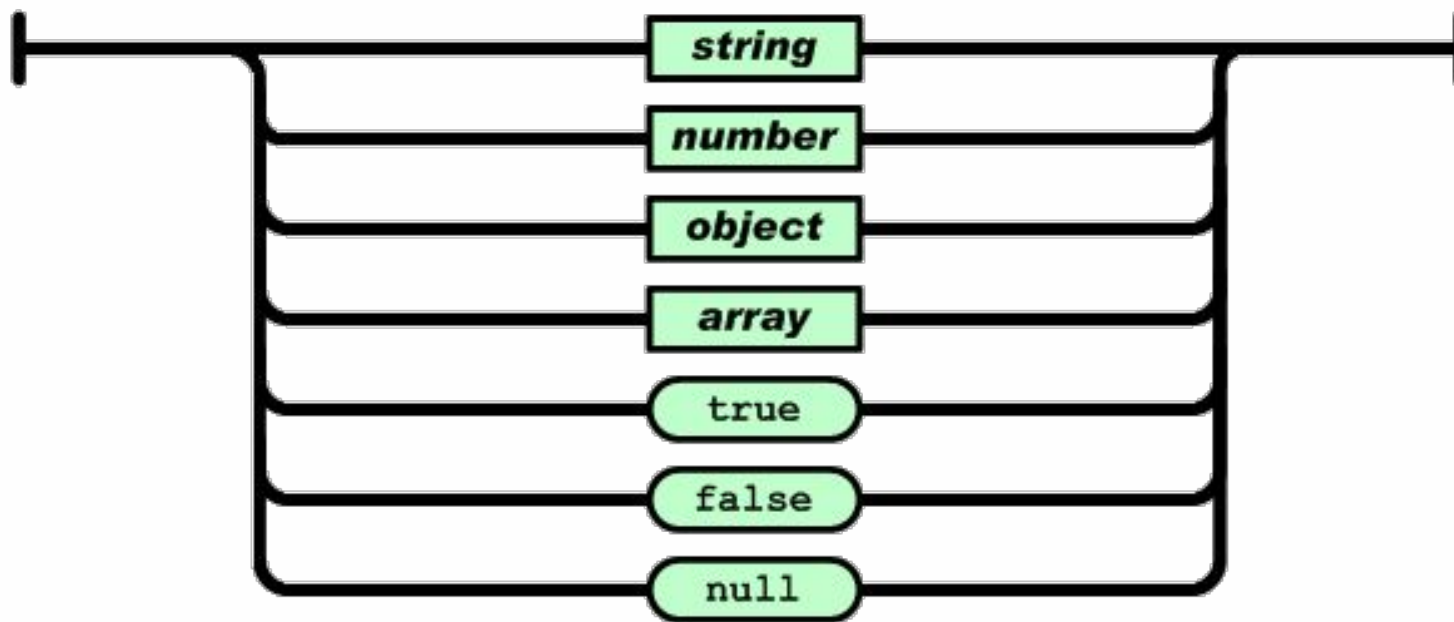
JSON основан на двух структурах данных:

2. Упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.



Формат JSON

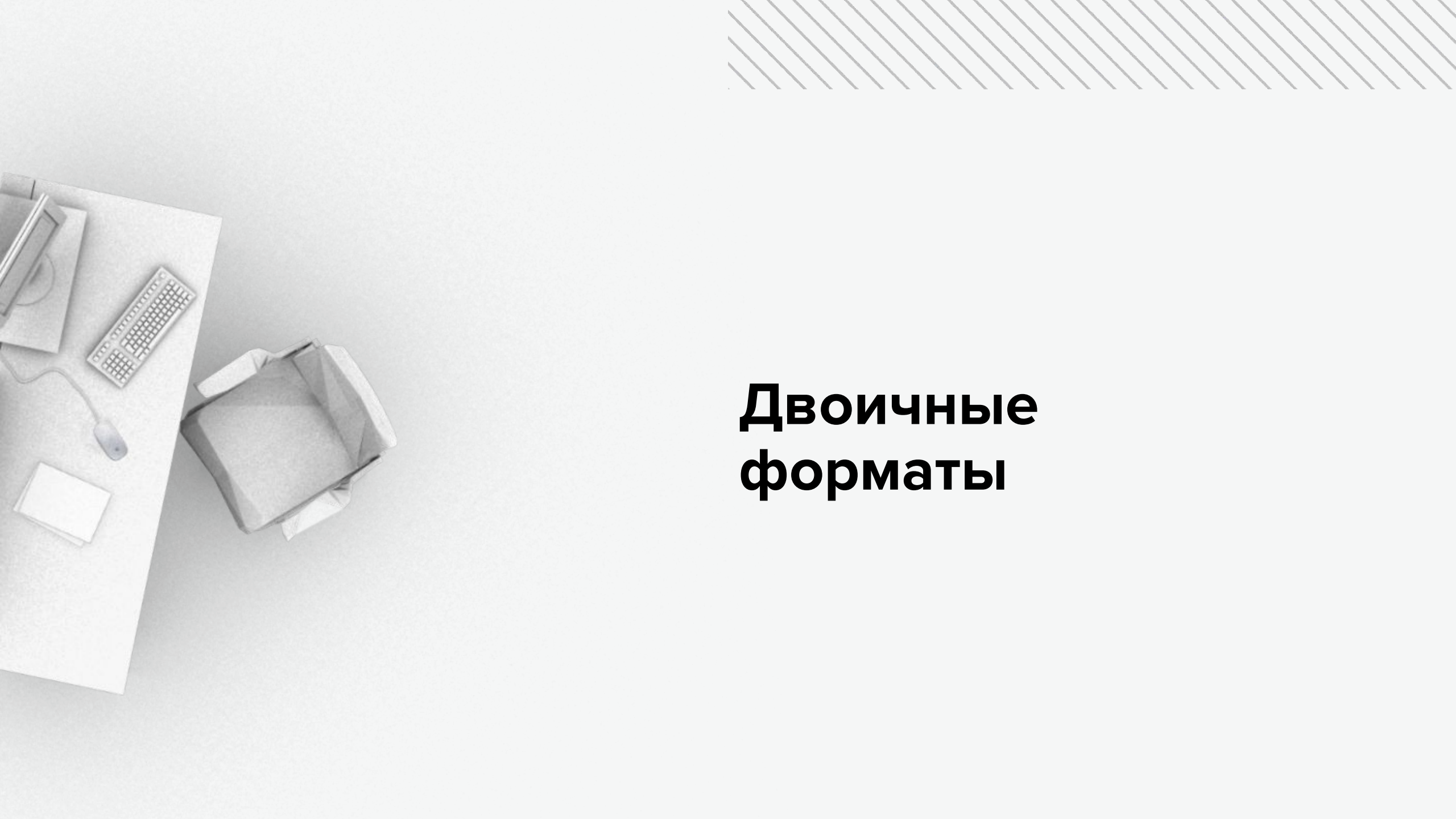
value





Преимущества JSON

- + Легко читается человеком;
- + Компактный;
- + Для работы с JSON есть множество библиотек;
- + Больше структурной информации в документе.



Двоичные форматы



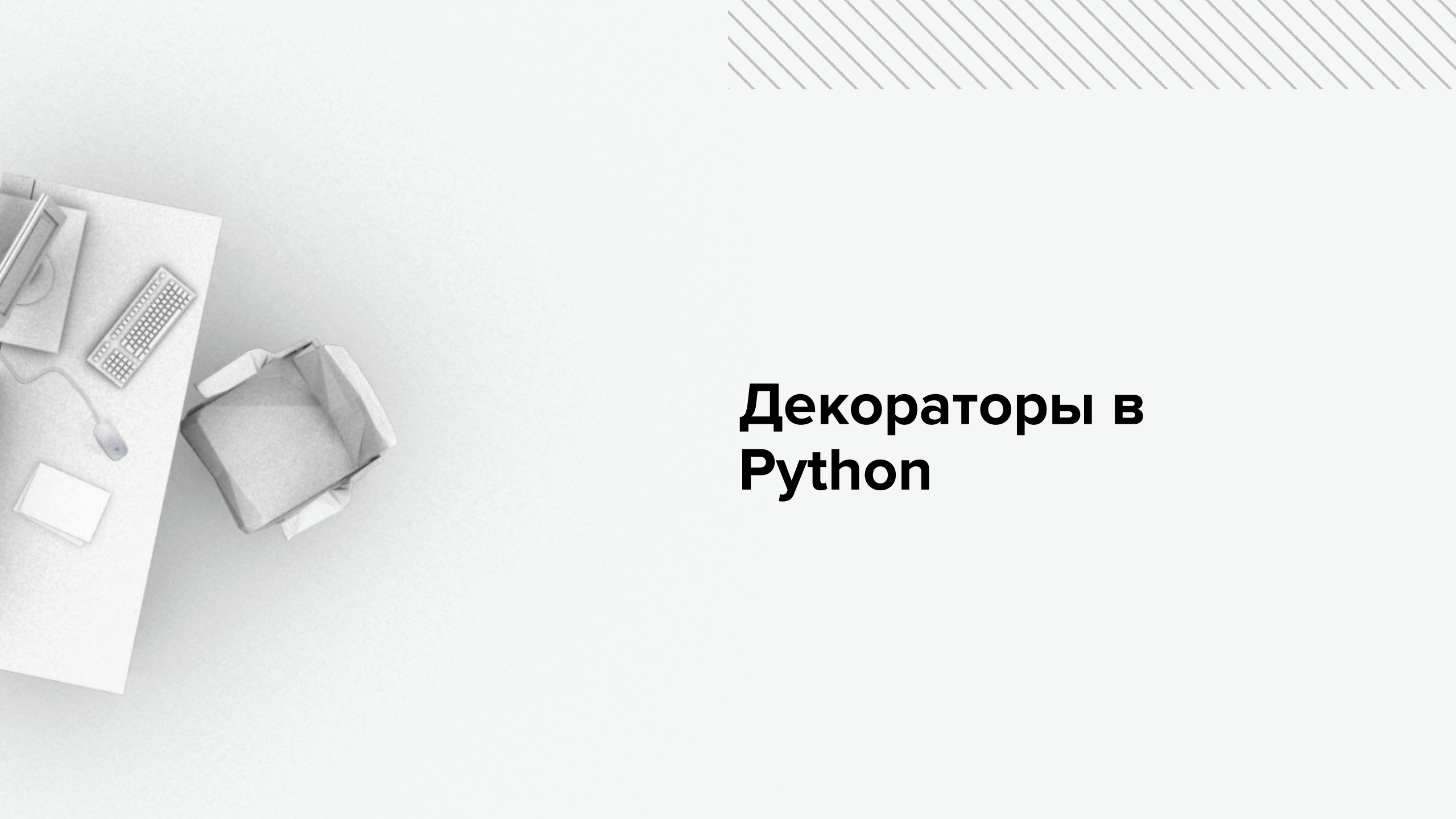
Преимущества двоичного кодирования

- Они могут быть намного компактнее различных вариантов “двоичного JSON”, поскольку позволяют не включать названия полей в закодированные данные;
- Схема - важный вид документа, вы всегда можете быть уверены в её актуальности;
- Пользователем языков программирования со статической типизацией окажется полезная возможность генерировать код на основе схемы, позволяющая проверять типы во время компиляции.

Protocol buffers

Protocol Buffers — протокол сериализации (передачи) структурированных данных, предложенный Google как эффективная бинарная альтернатива текстовому формату XML. Проще, компактнее и быстрее, чем XML.

```
message Person {  
    string user_name = 1;  
    int64 favorite_number = 2;  
    repeated string interests = 3;  
}
```



Декораторы в Python

Декораторы в Python

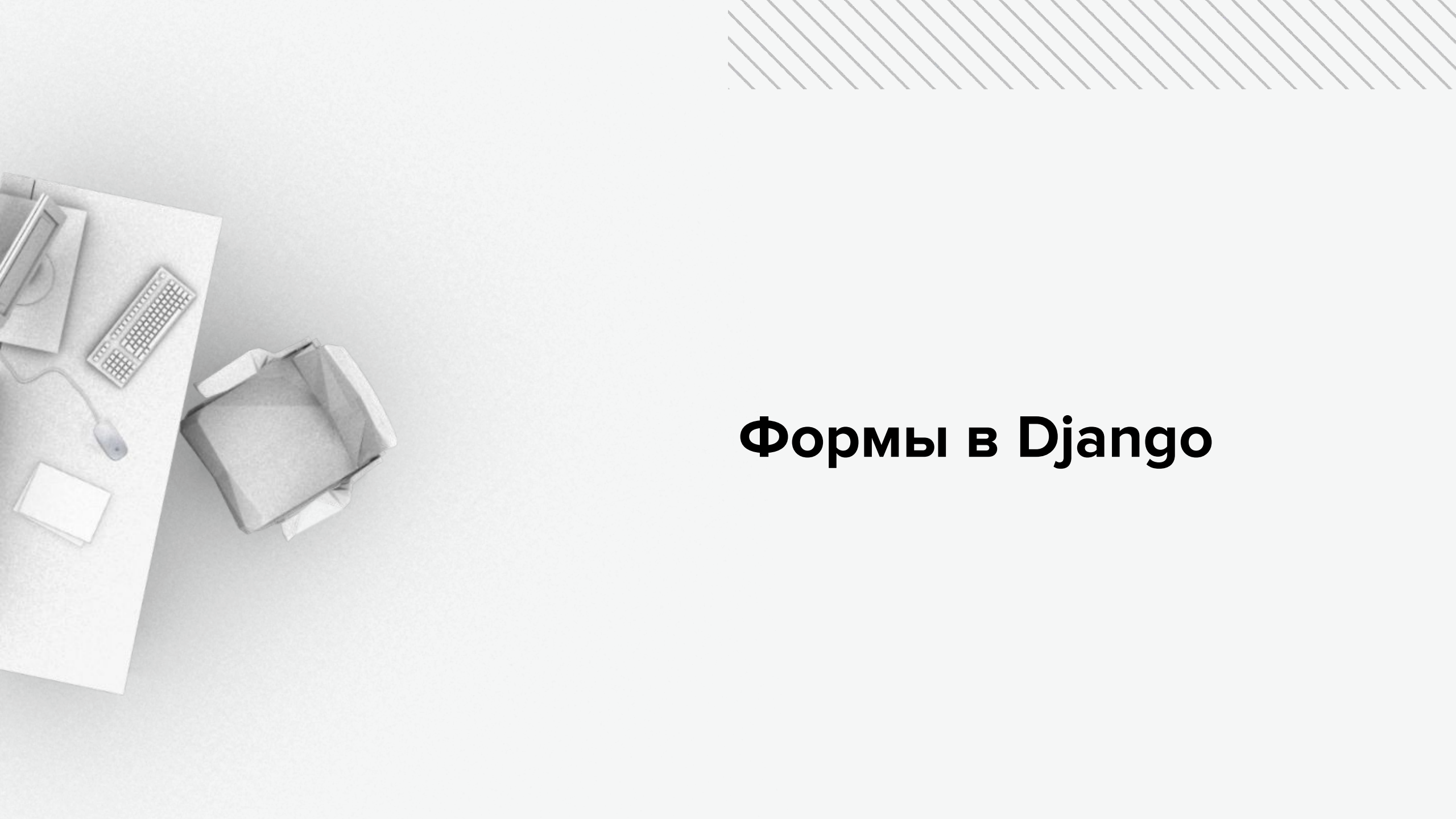
Это функция, которая принимает функцию в качестве единственного аргумента и возвращает новую функцию, с дополнительными функциональными возможностями.

```
def my_decorator(function):  
    def wrapper(*args, **kwargs):  
        print('It is decorator logic')  
        return function(*args, **kwargs)  
    return wrapper  
  
@my_decorator  
def foo():  
    print('It is main function')
```



Декораторы с параметрами

Вызываем функцию с требуемыми параметрами, и она вернёт декоратор, который будет использован для декорирования следующей за ним функцией.



Формы в Django

Best practices

`BooleanField` - флаг

`IntegerField` - целый тип

`CharField` - текстовое поле

`EmailField` - почтовый адрес

`PasswordField` - пароль

`DateTimeField` - дата

`DateTimeField` - время и дата

`FileField` - загрузка файла

Django-формы

```
# forms.py
from django import forms

class FeedbackForm(forms.Form):
    email = forms.EmailField(max_length=100)
    message = forms.CharField()
    def clean(self):
        if is_spam(self.cleaned_data):
            self.add_error('message', 'Это спам')
```

Django-формы

```
# forms.py
from django import forms
class PostForm(forms.Form):
    title = forms.CharField(max_length=100)
    text = forms.CharField()
    days_active = forms.IntegerField(required=False)
    def clean_text(self):
        if is_correct(self.cleaned_data['message']):
            return self.cleaned_data['message']
        return 'Текст содержал нецензурную лексику и был удален'
    def save(self):
        return Post.objects.create(**self.cleaned_data)
```

Типы полей

`BooleanField` — флаг

`IntegerField` — целый тип

`CharField` — текстовое поле

`EmailField` — почтовый адрес

`PasswordField` — пароль

`DateTimeField` — дата

`DateTimeField` — время и дата

`FileField` — загрузка файла

Валидация данных

- По типу поля, например `EmailField`
- `clean_xxx` - доп. проверка поля `xxx`, может изменить значение
- `clean` - доп. проверка всех полей

Методы `clean_xxx` и `clean` должны использовать `self.cleaned_data` для получения данных формы и поднять `ValidationError` в случае некорректных данных.

Model Forms

- метод `save` уже определен
- сохраняем в модель, указанную в `Meta`
- валидация полей проходит через типы, объявленные в модели

```
# forms.py
```

```
from django import forms
```

```
class PostForm(forms.ModelForm):
```

```
    class Meta:
```

```
        model = Post
```

```
        fields = ['title', 'text']
```

Использование во view

```
def add_post(request):  
    form = PostForm(request.POST)  
    if form.is_valid():  
        post = form.save()  
        return JsonResponse({  
            'msg': 'Пост сохранен',  
            'id': post.id  
        })  
    return JsonResponse({'errors': form.errors}, status=400)
```




Django Rest Framework

Django Rest Framework

```
# Устанавливаем DRF
pip install djangorestframework

# Добавляем приложение в settings.py
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

Основная архитектура

1. **Сериализатор**: преобразует информацию, хранящуюся в базе данных и определенную с помощью моделей Django, в формат, который легко и эффективно передается через API.
2. **Вид** (ViewSet): определяет функции (чтение, создание, обновление, удаление), которые будут доступны через API.
3. **Маршрутизатор**: определяет URL-адреса, которые будут предоставлять доступ к каждому виду

Сериализаторы

- Создаём класс, унаследованный от `serializers.ModelSerializer`
- Описываем поля, которые должны быть;
- Или внутри класса `Meta` указываем модель, по которой будем сериализовывать
- Можно сделать дополнительную валидацию, определив метод `validate_xxx` для поля `xxx`

```
class MovieSerializer(serializers.ModelSerializer):  
    genre = serializers.CharField(read_only=True)  
    class Meta:  
        model = Movie  
        fields = ('title', 'genre',)
```

ViewSet

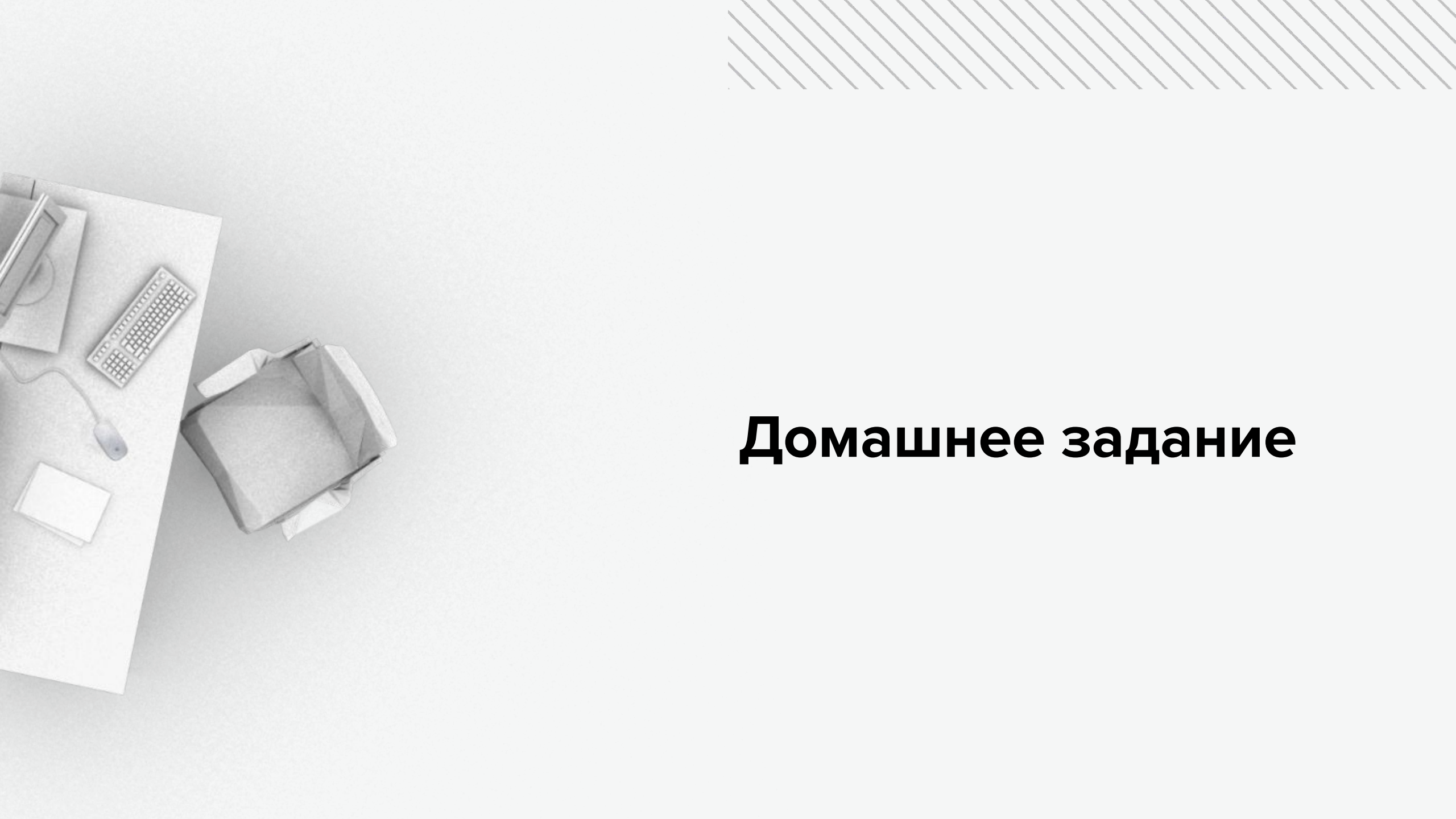
- Несколько классов ViewSet: APIView, GenericAPIView, ModelViewSet;
- Можем определить переменные внутри класса (ModelViewSet):
 - `queryset` - базовый queryset (запрос к базе), который используется для получение объектов.
 - `serializer_class` - класс сериализатора, который используется для проверки и десериализации объектов из базы

```
class ArticleViewSet(viewsets.ModelViewSet):  
  
    serializer_class = ArticleSerializer  
  
    queryset = Article.objects.all()
```

Code time!



Переписываем проект Movies/Genres с использованием DRF.



Домашнее задание

Домашнее задание #7

- Добавить в проект `djangoRESTframework`;
- Переписать заглушки всех предыдущих методов;
- Написать один или несколько форм для валидации форм.

Домашнее задание по уроку #7

Домашнее задание №7

#057

8

Баллов
за задание

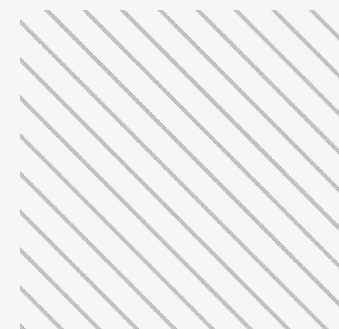
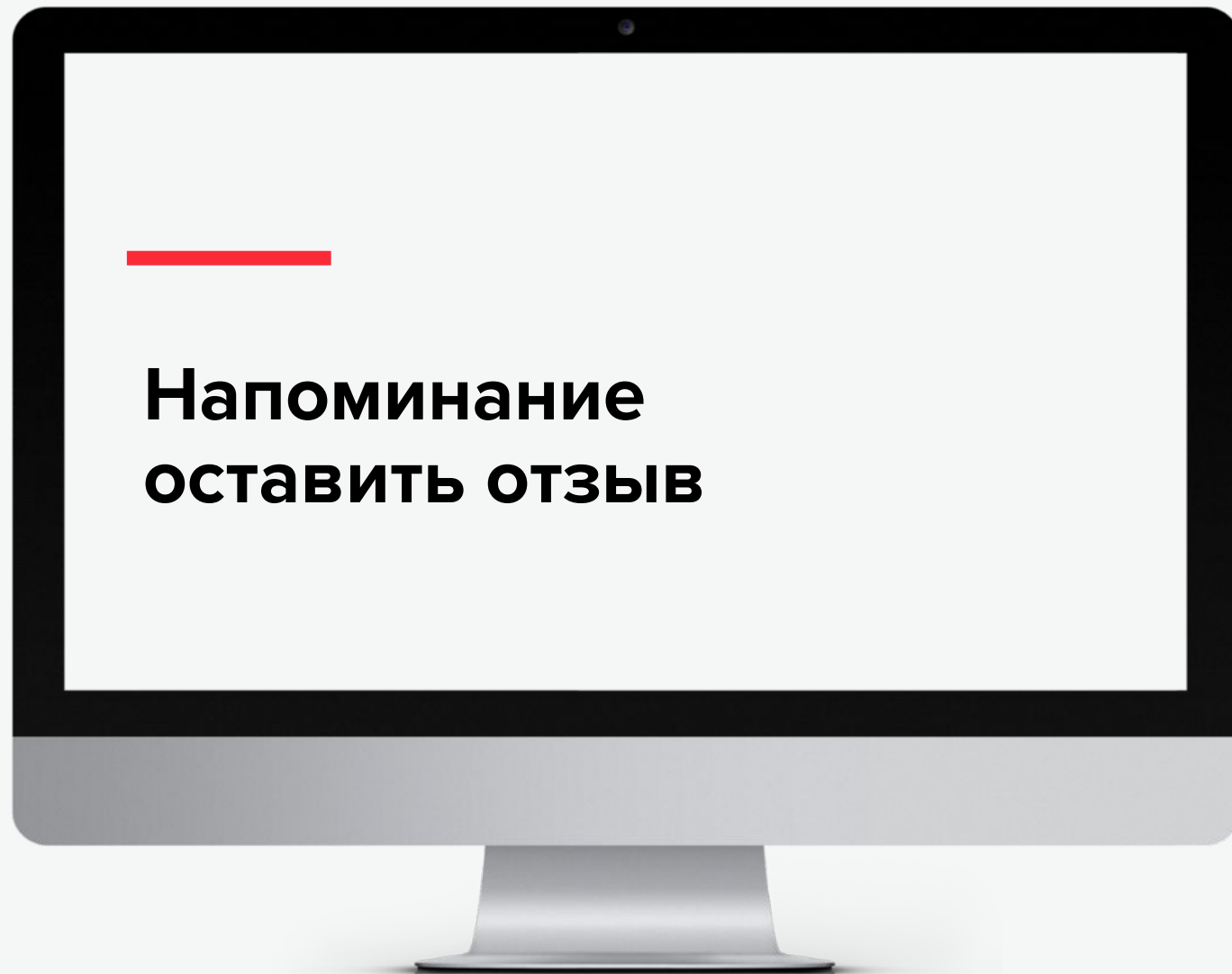
Сроков нет, но
вы держитесь

Срок
сдачи

Смотри подробнее файл
homework.md в репозитории с
лекциями!

Для саморазвития (опционально)
Чтобы не набирать двумя пальчиками





**СПАСИБО
ЗА ВНИМАНИЕ**

