# Подготовка к экзамену по веб-разработке

Краткая самостоятельная шпаргалка: тема, суть простыми словами, ссылка на подробный раздел внутри этого же документа и пример экзаменационного вопроса.

## Карта тем

### 1. Адресация, IPv4, IPv6

**Кратко:** IP-адрес нужен, чтобы доставить пакет до конкретного узла сети. IPv4 записывается как четыре числа и занимает 32 бита, поэтому адресов мало; IPv6 занимает 128 бит и записывается шестнадцатеричными блоками. Доменное имя само по себе не является сетевым адресом: перед подключением DNS превращает его в IP.

**Подробнее:** адресация и IP

**Вопрос:** Чем IPv4 отличается от IPv6?

### 2. TCP, порты, NAT и подсети

**Кратко:** TCP устанавливает соединение и контролирует доставку данных: порядок, подтверждения и повторную отправку. IP выбирает устройство, а порт выбирает приложение на этом устройстве. Маска подсети отделяет адрес сети от адреса хоста, а NAT позволяет приватным адресам локальной сети выходить наружу через публичный IP.

**Подробнее:** TCP, порты, NAT

**Вопрос:** Зачем нужен NAT и как он связан с портами?

## 3. DNS, URI, URL, URN

**Кратко:** DNS сопоставляет доменное имя с IP-адресом, чтобы браузер понял, к какому серверу подключаться. URI - общее понятие для идентификатора ресурса. URL является частным случаем URI: он указывает, где находится ресурс и как его получить; URN задаёт устойчивое имя без обязательного сетевого адреса.

**Подробнее:** DNS и идентификаторы

**Вопрос:** Почему URL является URI, но не каждый URI является URL?

## 4. HTTP, HTTPS, HTTP/3

**Кратко:** HTTP - прикладной протокол обмена сообщениями: клиент отправляет запрос, сервер возвращает ответ со статусом, заголовками и, возможно, телом. GET обычно получает данные, POST отправляет данные в теле запроса. HTTPS - это HTTP внутри TLS-соединения, а HTTP/3 сохраняет смысл HTTP, но использует QUIC поверх UDP.

**Подробнее:** HTTP-протоколы

**Вопрос:** Чем GET отличается от POST?

## 5. Идентификация, аутентификация, OAuth 2.0

**Кратко:** Идентификация отвечает на вопрос "кто пользователь?", аутентификация проверяет доказательство личности, например пароль или код. Авторизация решает, какие действия разрешены уже проверенному пользователю. OAuth 2.0 используется, чтобы стороннее приложение получило ограниченный access token без получения пароля пользователя.

**Подробнее:** доступ и OAuth

**Вопрос:** Почему OAuth 2.0 называют протоколом авторизации?

## 6. HTML-документ

**Кратко:** HTML описывает смысловую структуру страницы: заголовки, абзацы, ссылки, изображения, таблицы и разделы. В `head` находятся служебные данные: кодировка, viewport, заголовок вкладки, подключение CSS и скриптов. В `body` находится видимый контент; семантические теги помогают показать назначение частей страницы.

**Подробнее:** HTML-структура

**Вопрос:** Для чего нужен meta viewport?

## 7. HTML-формы

**Кратко:** HTML-форма собирает ввод пользователя и отправляет его на сервер по адресу из `action` методом `GET` или `POST`. Атрибут `name` определяет имя параметра в запросе, а `id` обычно нужен для связи поля с `label`. `required`, `type` и другие атрибуты дают базовую проверку в браузере.

**Подробнее:** HTML Forms

**Вопрос:** Чем radio отличается от checkbox?

## 8. CSS selectors и специфичность

**Кратко:** CSS-селектор выбирает элементы, к которым применяются свойства. При конфликте правил учитываются каскад, специфичность и порядок записи. ID сильнее класса, класс сильнее тега; если специфичность равна, побеждает правило, написанное ниже.

**Подробнее:** CSS selectors

**Вопрос:** Какой селектор сильнее: `body .content` или `div p`?

### 9. CSS box model и базовые свойства

**Кратко:** Каждый элемент в CSS рассматривается как прямоугольный блок: content, padding, border и margin. При `content-box` ширина задаёт только content, а padding и border добавляются сверху. При `border-box` заданная ширина уже включает content, padding и border, поэтому размеры проще контролировать.

**Подробнее:** блочная модель

**Вопрос:** Какая ширина у блока с `width:100px` и `box-sizing:border-box`?

### 10. Flexbox и Grid

**Кратко:** Flexbox удобен для одномерной раскладки: элементы распределяются вдоль главной оси, а поперечная ось используется для выравнивания. Grid решает двумерные задачи: одновременно управляет строками, столбцами, линиями, ячейками и областями. `fr`, `repeat()` и `minmax()` помогают задавать гибкую сетку.

**Подробнее:** Flexbox и Grid

**Вопрос:** Какое значение `align-items` по умолчанию?

### 11. Django models и ORM queries

**Кратко:** Модель Django описывает таблицу базы данных: класс - таблица, поле - колонка, объект - строка. ORM позволяет строить SQL-запросы через Python-методы вроде `filter()`, `get()`, `annotate()` и `aggregate()`. QuerySet ленивый: запрос обычно выполняется только тогда, когда реально нужны данные.

**Подробнее:** Django models и queries

**Вопрос:** Когда использовать `select_related`, а когда `prefetch_related`?

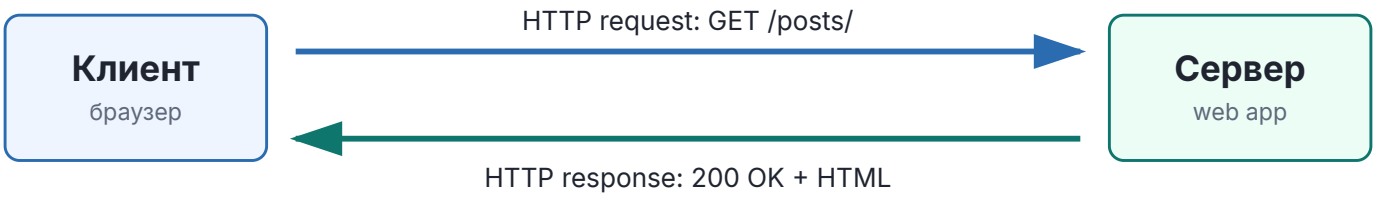
### 12. Django views, URLs, templates, forms

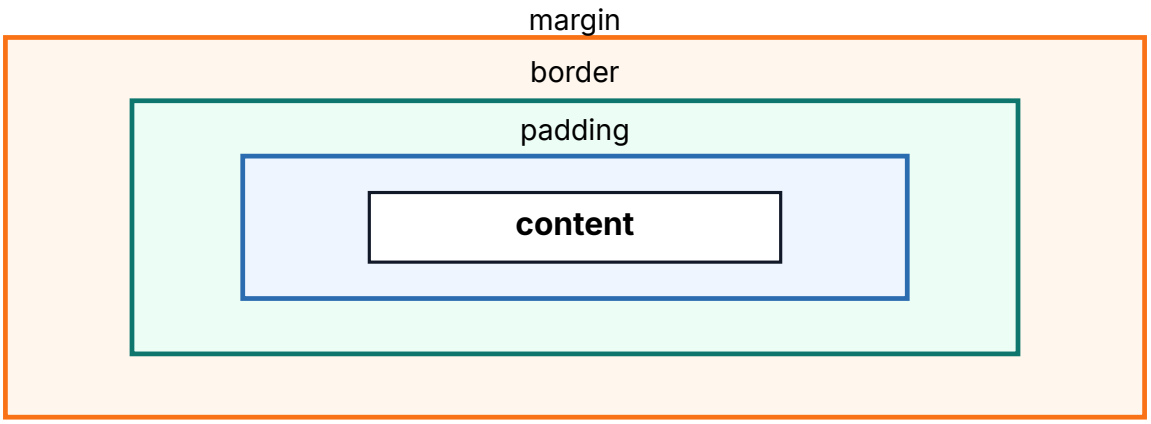
**Кратко:** Django view принимает `request` и возвращает HTTP-ответ: HTML, redirect, JSON или ошибку. URL-маршрут связывает путь с view, шаблон получает context и отображает данные, а форма описывает поля, валидирует ввод и отдаёт очищенные значения через `cleaned_data`. FBV проще как функция, CBV удобнее для типовых страниц.

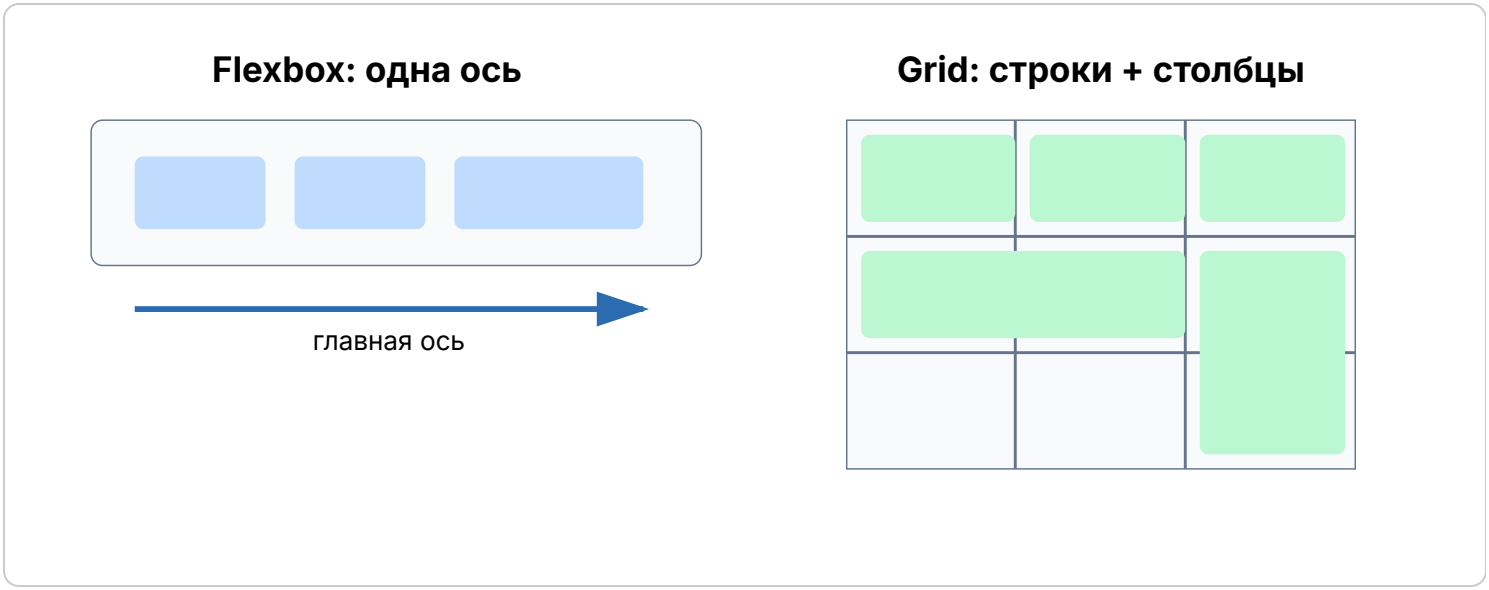
**Подробнее:** Django web layer

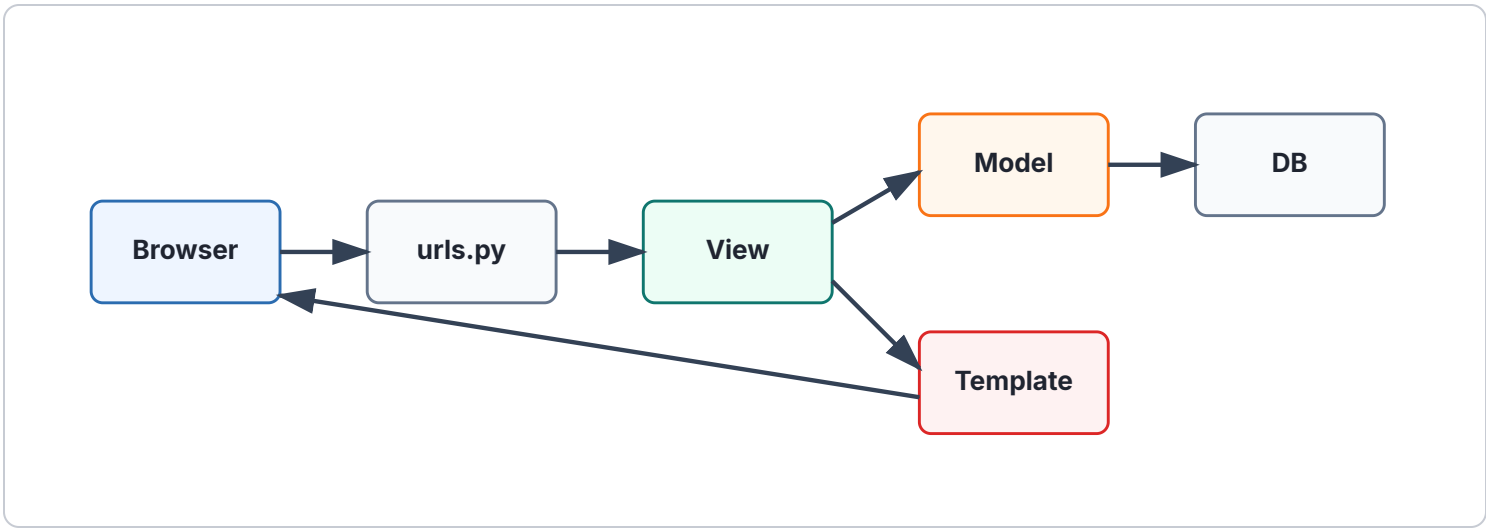
**Вопрос:** Что будет, если в `ListView` не указать ни `model`, ни `queryset`?

## Схемы






## Подробная теория

## Адресация и IP

← к краткой карточке

IP-адрес нужен, чтобы доставить пакет до конкретного узла сети. Доменное имя удобно человеку, но сеть работает с IP.

IPv4 32 бита, запись вроде `192.168.1.10`. Адресов мало, поэтому используют приватные диапазоны и NAT.

IPv6 128 бит, запись вроде `2001:db8::1`. Адресов намного больше, запись можно сокращать через `::`.

Приватные IPv4-диапазоны: `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`. `127.0.0.1` - localhost, обращение к самому себе.

**IP**

Протокол сетевого уровня. Он отвечает за адресацию и доставку пакетов между сетями, но сам не гарантирует доставку.

**IPv4**

Версия IP с 32-битным адресом. Записывается четырьмя октетами: `0.0.0.0` ... `255.255.255.255`.

**IPv6**

Версия IP с 128-битным адресом. Нужна из-за нехватки IPv4 и упрощает глобальную адресацию.

**Октет**

8 бит. В IPv4 четыре октета, каждый отображается числом от 0 до 255.

**Публичный адрес**

Адрес, который маршрутизируется в интернете и может быть доступен извне.

**Приватный адрес**

Адрес для локальных сетей. В интернет напрямую не выходит, обычно используется с NAT.

**Localhost**

Имя или адрес для обращения к самому компьютеру. Обычно `localhost` и `127.0.0.1`.

**Хост**

Устройство или интерфейс в сети: компьютер, сервер, телефон, контейнер, виртуальная машина.

**Как выглядит путь от имени сайта к IP**

1. Пользователь вводит домен, например `example.com`.
2. Браузер или ОС спрашивает DNS, какой IP соответствует имени.
3. После получения IP клиент открывает соединение к этому адресу.
4. Чтобы попасть в нужную программу на сервере, используется порт: например `443` для HTTPS.

> На экзамене важно не путать: домен - имя для человека, IP - адрес узла, порт - адрес приложения внутри узла.

## TCP, порты, NAT и подсети

← к краткой карточке

TCP устанавливает соединение через три шага: `SYN`, `SYN-ACK`, `ACK`. Он следит за порядком данных, потерями и повторной отправкой.

Порт выбирает приложение внутри одного IP-адреса. HTTP обычно работает на `80`, HTTPS - на `443`.

Маска подсети показывает, где в IP-адресе часть сети, а где часть хоста. В `192.168.1.10/24` сеть - `192.168.1.0`, хост - `.10`.

NAT переводит приватные адреса во внешний публичный адрес. PAT дополнительно использует порты, поэтому много внутренних устройств могут выходить через один публичный IP.

**TCP**

Надёжный транспортный протокол. Перед передачей устанавливает соединение и подтверждает получение данных.

**UDP**

Транспортный протокол без соединения и подтверждений. Быстрее, но не гарантирует доставку и порядок.

**SYN**

Первый пакет TCP-рукопожатия: клиент просит открыть соединение.

**SYN-ACK**

Ответ сервера: запрос принят, сервер тоже готов открыть соединение.

**ACK**

Подтверждение клиента. После него TCP-соединение считается установленным.

**Порт**

Число от 0 до 65535, которое указывает процесс или сервис внутри IP-адреса.

**Маска подсети**

Показывает, какие биты IP относятся к сети, а какие к хосту.

**CIDR**

Краткая запись маски, например `/24`. Число показывает количество сетевых бит.

**NAT**

Замена адресов на границе сети. Обычно приватный адрес заменяется публичным.

**PAT**

Вариант NAT с портами. Позволяет многим внутренним устройствам использовать один внешний IP.

### Трёхстороннее рукопожатие TCP

1. `SYN`: клиент говорит "хочу соединиться".
2. `SYN-ACK`: сервер говорит "готов, подтверждаю".
3. `ACK`: клиент подтверждает ответ сервера.

### Подсеть на примере

`192.168.1.10/24`: первые 24 бита - сеть, последние 8 бит - хосты. Адрес сети `192.168.1.0`, типичный шлюз `192.168.1.1`, устройства часто получают адреса `192.168.1.2` ... `192.168.1.254`.

> Если два IP в одной подсети, устройство может отправить кадр напрямую в локальную сеть. Если адрес в другой подсети, пакет идёт через шлюз.

## DNS и URI/URL/URN

← к краткой карточке

DNS - система имён. Она ищет IP-адрес по домену. Упрощённо: кэш браузера, кэш ОС, resolver, root, TLD, авторитативный DNS-сервер.

| Запись | Смысл |
| --- | --- |
| `A` | домен в IPv4 |
| `AAAA` | домен в IPv6 |
| `CNAME` | псевдоним |
| `MX` | почтовый сервер |
| `TXT` | служебный текст, например SPF/DKIM |

URI - общий идентификатор. URL - URI с адресом и способом получения: `https://example.com/path?x=1#top`. URN - устойчивое имя, например ISBN или UUID.

**DNS resolver**

Сервер, который принимает запрос клиента и ищет ответ в DNS-иерархии.

**Root server**

Корневой DNS-сервер. Не знает IP всех сайтов, но знает, куда отправить за зоной верхнего уровня.

**TLD**

Зона верхнего уровня: `.ru`, `.com`, `.org`.

**Authoritative server**

Авторитативный сервер домена. Он хранит окончательный ответ по доменному имени.

**URI**

Uniform Resource Identifier. Любой идентификатор ресурса.

**URL**

Uniform Resource Locator. URI, который указывает адрес и способ получения ресурса.

**URN**

Uniform Resource Name. Устойчивое имя ресурса без обязательного расположения.

**Фрагмент**

Часть URL после `#`. Обычно указывает на элемент внутри документа.

### Компоненты URL

```
https://example.com:443/articles/1?print=true#comments
```

- `https` - схема, то есть способ доступа.
- `example.com` - хост: домен или IP.
- `443` - порт. Можно не писать, если он стандартный для схемы.
- `/articles/1` - путь к ресурсу на сервере.
- `?print=true` - query string, параметры запроса.
- `#comments` - фрагмент внутри страницы.

> Формула: URL всегда URI, потому что он идентифицирует ресурс. Но URI не всегда URL, потому что может быть просто именем, например URN.

## HTTP, HTTPS, HTTP/3

← к краткой карточке

HTTP-сообщение состоит из стартовой строки, заголовков, пустой строки и необязательного тела.

```
GET /posts/?page=2 HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>...</html>
```

`GET` обычно получает данные, параметры видны в URL. `POST` обычно отправляет данные в теле запроса.

Коды: `2xx` успех, `3xx` редирект, `4xx` ошибка клиента, `5xx` ошибка сервера.

HTTPS - HTTP поверх TLS. HTTP/3 - HTTP поверх QUIC/UDP; смысл методов и кодов не меняется.

**Метод**

Глагол HTTP-запроса: что клиент хочет сделать с ресурсом.

**Заголовок**

Пара "имя: значение" с метаданными запроса или ответа.

**Тело**

Данные после заголовков. В POST там часто форма или JSON.

**Статус-код**

Числовой результат обработки запроса: `200`, `404`, `500`.

**TLS**

Криптографический слой HTTPS: шифрует трафик и проверяет сервер.

**QUIC**

Транспорт для HTTP/3 поверх UDP со встроенным TLS 1.3 и потоками.

### Методы HTTP

| Метод | Назначение |
| --- | --- |
| `GET` | Получить ресурс. Не должен менять состояние сервера. |
| `POST` | Отправить данные: форму, JSON, файл, команду создания. |
| `HEAD` | Получить только заголовки, без тела ответа. |
| `PUT` | Полностью создать или заменить ресурс. |
| `PATCH` | Частично изменить ресурс. |
| `DELETE` | Удалить ресурс. |
| `OPTIONS` | Узнать возможности сервера или разрешённые методы. |

### Частые заголовки

| Заголовок | Смысл |
| --- | --- |
| `Host` | Домен или хост, к которому обращается клиент. |
| `User-Agent` | Информация о клиенте: браузер, ОС, версия. |
| `Accept` | Какие типы ответа клиент готов принять. |
| `Content-Type` | Тип данных в теле сообщения. |
| `Cookie` | Cookies, которые браузер отправляет серверу. |
| `Authorization` | Данные доступа, например `Bearer token`. |

> HTTPS не меняет HTML, URL-маршруты, методы GET/POST и коды ответа. Он добавляет защищённое соединение под HTTP.

## Идентификация, аутентификация, авторизация, OAuth 2.0

← к краткой карточке

Идентификация: пользователь сообщает, кто он. Аутентификация: система проверяет доказательство. Авторизация: система проверяет права.

OAuth 2.0 позволяет приложению получить access token без знания пароля пользователя. Права ограничиваются scope, сроком жизни токена и настройками сервиса.

1. Клиент отправляет пользователя на authorization server.
2. Пользователь подтверждает доступ.
3. Клиент получает authorization code.
4. Код меняется на access token.
5. Access token используется для API-запросов.

**Идентификация**

Пользователь сообщает идентификатор: логин, email, номер телефона.

**Аутентификация**

Проверка доказательства: пароль, код, ключ, сертификат.

**Авторизация**

Проверка прав: можно ли читать, создавать, редактировать, удалять.

**Сессия**

Состояние входа пользователя на сервере, обычно связывается с cookie в браузере.

**Cookie**

Небольшое значение, которое браузер хранит для сайта и отправляет с запросами.

**Bearer token**

Токен доступа. Кто владеет токеном, тот может выполнить разрешённые действия.

**Access token**

Короткоживущий токен для доступа к API.

**Refresh token**

Токен для получения нового access token без повторного входа пользователя.

**Scope**

Список разрешений токена: например читать профиль, но не удалять данные.

**OpenID Connect**

Надстройка над OAuth 2.0 для аутентификации пользователя.

> OAuth 2.0 сам по себе отвечает за доступ к ресурсам, а не за доказательство личности. Для "входа через провайдера" обычно нужен OpenID Connect.

## HTML-структура

← к краткой карточке

HTML задаёт смысловую структуру документа. Комментарий пишется так: `<!-- comment -->`.

`head` содержит `meta`, `title`, `link`, `script`. `meta viewport` нужен, чтобы мобильный браузер корректно рассчитывал ширину страницы.

Блочные элементы занимают строку: `div`, `p`, `section`, `form`. Строчные идут внутри текста: `span`, `a`, `strong`, `em`.

Семантические теги: `header`, `nav`, `main`, `section`, `article`, `aside`, `footer`.

### Минимальный документ

```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Страница</title>
  </head>
  <body>
    <main>Контент</main>
  </body>
</html>
```

**`<!doctype html>`**

Сообщает браузеру, что документ написан по HTML5 и его нужно обрабатывать в стандартном режиме.

**`<html>`**

Корневой элемент всей HTML-страницы. Атрибут `lang` задаёт язык документа.

**`<head>`**

Служебная часть документа. Не отображает основной контент пользователю.

**`<body>`**

Видимое содержимое страницы: текст, формы, изображения, таблицы, разделы.

**`<meta charset>`**

Задаёт кодировку. Для русских текстов обычно нужен `UTF-8`.

**`<meta viewport>`**

Говорит мобильному браузеру использовать ширину устройства, а не виртуальную широкую страницу.

**`<title>`**

Название вкладки браузера и заголовок страницы для поисковиков.

**`<link>`**

Подключает внешний ресурс, чаще всего CSS: `rel="stylesheet"`.

**`<script>`**

Подключает или содержит JavaScript. Может быть в `head` или в конце `body`.

**`<style>`**

Внутренние CSS-правила прямо в HTML-документе.

**Теги содержимого**

| Тег | Назначение |
|---|---|
| `<h1>...<h6>` | Заголовки уровней. `h1` главный, `h6` самый низкий. |
| `<p>` | Абзац текста. |
| `<br>` | Принудительный перенос строки. Не использовать вместо абзацев. |
| `<strong>` | Важное смысловое выделение, обычно отображается жирным. |
| `<em>` | Смысловой акцент, обычно отображается курсивом. |
| `<code>` | Фрагмент кода или технического текста. |
| `<div>` | Нейтральный блочный контейнер без собственного смысла. |
| `<span>` | Нейтральный строчный контейнер без собственного смысла. |
| `<a>` | Ссылка. Главный атрибут: `href`. |

| | |
|---|---|
| `<img>` | Изображение. Важны `src` и `alt`. |
| `<ul>`, `<ol>`, `<li>` | Маркированный список, нумерованный список и элемент списка. |
| `<table>`, `<tr>`, `<th>`, `<td>` | Таблица, строка, заголовочная ячейка, обычная ячейка. |
| `<caption>` | Название таблицы. |
| `<thead>`, `<tbody>`, `<tfoot>` | Группы строк таблицы: заголовок, тело, итог. |

**Семантические теги документа**

| Тег | Когда использовать |
|---|---|
| `<header>` | Верхняя часть страницы или секции: логотип, заголовок, навигация. |
| `<nav>` | Блок основных навигационных ссылок. |
| `<main>` | Главный контент страницы. Обычно один на страницу. |
| `<section>` | Смысловой раздел, обычно с заголовком. |
| `<article>` | Самостоятельный материал: статья, пост, карточка новости. |
| `<aside>` | Дополнительный контент: боковая панель, подсказки, связанные ссылки. |
| `<footer>` | Нижняя часть страницы или секции. |

**Атрибуты HTML**

**`class`**
Класс элемента. Один и тот же класс можно дать многим элементам для CSS и JS.

**`id`**
Уникальный идентификатор элемента на странице. Используется для якорей, label, JS.

**`href`**
Адрес ссылки у `a`.

**`src`**

Источник файла у `img`, `script` и некоторых других тегов.

**`alt`**

Текстовое описание изображения. Нужно для доступности и если картинка не загрузилась.

**`target="_blank"`**

Открывает ссылку в новой вкладке.

**`colspan`**

Объединяет ячейку таблицы по столбцам.

**`rowspan`**

Объединяет ячейку таблицы по строкам.

> В обычном HTML несколько пробелов подряд схлопываются в один. Перенос строки в исходном коде не означает перенос на странице.

## HTML Forms

← к краткой карточке

`form` отправляет данные. `action` задаёт URL, `method` - способ отправки: `get` или `post`.

`id` уникален на странице и связывает поле с label. `name` - имя параметра, которое уйдёт на сервер.

```
<label for="email">Email</label>
<input id="email" name="email" type="email" required>
```

`radio` выбирает один вариант из группы с одинаковым `name`. `checkbox` - независимый флажок. `fieldset` и `legend` группируют поля.

`placeholder` - подсказка внутри поля. `required` - поле обязательно. У кнопки лучше явно писать `type="submit"`, `type="button"` или `type="reset"`.

### Теги формы

| Тег | Назначение |
|---|---|
| `<form>` | Контейнер формы. Определяет, куда и как отправлять данные. |
| `<input>` | Универсальное поле. Поведение зависит от `type`. |
| `<label>` | Подпись поля. Связывается с полем через `for` и `id`. |
| `<button>` | Кнопка формы или обычного действия. |
| `<textarea>` | Многострочное текстовое поле. |
| `<select>` | Выпадающий список. |
| `<option>` | Вариант внутри `select`. |
| `<fieldset>` | Логическая группа полей. |
| `<legend>` | Заголовок группы `fieldset`. |

### Атрибуты формы и полей

**`action`**
URL, куда отправляется форма.

**`method`**
HTTP-метод отправки: чаще `get` или `post`.

**`name`**
Имя параметра, которое попадёт в запрос на сервер.

**`id`**
Идентификатор поля, нужен для связи с `label`.

**`value`**
Значение поля или значение варианта.

**`required`**
Поле обязательно для заполнения в браузере.

**`placeholder`**
Подсказка внутри пустого поля. Не заменяет подпись.

**`checked`**
Предварительно выбранный checkbox или radio.

**`disabled`**
Поле выключено и обычно не отправляется.

**`readonly`**
Поле нельзя редактировать, но значение может отправляться.

#### Основные input type

| Тип | Что делает |
|---|---|
| `text` | Обычная строка. |
| `password` | Пароль, символы скрываются. |
| `email` | Email с базовой браузерной проверкой. |

| | |
|---|---|
| `number` | Число, можно использовать `min` и `max`. |
| `radio` | Один выбор из группы с одинаковым `name`. |
| `checkbox` | Независимый флажок. Можно выбрать несколько. |
| `submit` | Кнопка отправки формы. |
| `file` | Выбор файла. |
| `hidden` | Скрытое поле, не видно пользователю, но отправляется. |

> Браузерная HTML-валидация удобна, но не защищает сервер. Любые данные формы нужно проверять на сервере.

## CSS selectors и специфичность

← к краткой карточке

Селекторы: `*`, `p`, `.class`, `#id`, `div p`, `div > p`, `h2 + p`, `input[required]`, `:hover`, `:nth-child(2)`.

Специфичность: inline-style сильнее id, id сильнее class/attribute/pseudo-class, class сильнее tag.

`body .content` сильнее `div p`, потому что класс имеет больший вес, чем селектор тега.

Если специфичность одинаковая, побеждает правило ниже в CSS. Наследуются, например, `color` и `font-family`; не наследуются `margin`, `padding`, `border`.

### Типы селекторов

| Селектор | Пример | Что выбирает |
|---|---|---|
| Универсальный | `*` | Все элементы. |
| Элемент | `p` | Все теги `p`. |
| Класс | `.card` | Все элементы с `class="card"`. |
| ID | `#main` | Элемент с `id="main"`. |
| Потомок | `article p` | Все `p` внутри `article` на любой глубине. |
| Дочерний | `article > p` | Только прямые дочерние `p`. |
| Соседний | `h2 + p` | Первый `p` сразу после `h2`. |
| Общий соседний | `h2 ~ p` | Все `p` после `h2` на том же уровне. |
| Атрибут | `input[required]` | Элементы с указанным атрибутом. |
| Псевдокласс | `a:hover` | Состояние элемента или структурное условие. |
| Псевдоэлемент | `p::first-line` | Виртуальную часть элемента. |

### Псевдоклассы

**`:hover`**

Когда указатель мыши над элементом.

**`:focus`**
Когда элемент в фокусе, например поле ввода активно.

**`:active`**
Момент активации, например нажатие мышью.

**`:first-child`**
Элемент является первым ребёнком родителя.

**`:last-child`**
Элемент является последним ребёнком родителя.

**`:nth-child(n)`**
Элемент на заданной позиции среди детей.

**`:not(...)`**
Элементы, которые не подходят под вложенный селектор.

**`:checked`**
Выбранный checkbox или radio.

**Расчёт специфичности**

Удобно считать как четыре группы: inline, id, class/attribute/pseudo-class, tag/pseudo-element.

| Селектор | Вес | Комментарий |
|---|---|---|
| `div p` | `0-0-0-2` | Два тега. |
| `body .content` | `0-0-1-1` | Один класс и один тег. |
| `#app .content p` | `0-1-1-1` | ID почти всегда перебивает классы и теги. |
| `style="..."` | `1-0-0-0` | Inline-стиль сильнее обычного CSS. |

> Группировка селекторов через запятую не увеличивает специфичность. `h1, h2` - это два отдельных правила с одинаковыми объявлениями.

## CSS box model и базовые свойства

← к краткой карточке

Блок состоит из content, padding, border и margin. `content-box` считает `width` только как ширину контента. `border-box` включает padding и border в width.

```
width: 100px;
padding: 10px;
border: 5px solid;
box-sizing: border-box;
```

Реальная ширина такого блока - `100px`.

Шрифты: `font-family`, `font-size`, `font-weight`, `font-style`. Текст: `text-align`, `text-decoration`, `line-height`, `letter-spacing`, `text-shadow`.

### Части блочной модели

**Content**

Содержимое элемента: текст, картинка, дочерние элементы.

**Padding**

Внутренний отступ между content и border. Увеличивает визуальный размер блока.

**Border**

Рамка вокруг padding и content.

**Margin**

Внешний отступ между элементом и соседями.

**`content-box`**

Значение по умолчанию: `width` задаёт только content.

**`border-box`**

`width` включает content, padding и border.

### Display

| Значение | Поведение |
| --- | --- |
| `block` | Элемент начинается с новой строки и обычно занимает всю доступную ширину. |
| `inline` | Идёт внутри строки, ширина и высота задаются плохо или не работают как у блока. |
| `inline-block` | Снаружи ведёт себя как строчный, внутри как блочный: можно задавать размеры. |
| `none` | Элемент полностью убирается из раскладки. |
| `flex` | Создаёт flex-контейнер. |
| `grid` | Создаёт grid-контейнер. |

**Фон и границы**

| Свойство | Смысл |
| --- | --- |
| `background-color` | Цвет фона. |
| `background-image` | Фоновое изображение. |
| `background-repeat` | Повторять ли фон. |
| `background-position` | Положение фонового изображения. |
| `background-size` | Размер фонового изображения: например `cover` или `contain`. |
| `border` | Краткая запись рамки: ширина, стиль, цвет. |
| `border-radius` | Скругление углов. |
| `box-shadow` | Тень блока. |

**Текст и шрифты**

| Свойство | Смысл |
| --- | --- |
| `font-family` | Семейство шрифта. |
| `font-size` | Размер шрифта. |
| `font-weight` | Толщина: например `400`, `700`, `bold`. |

| | |
|---|---|
| `font-style` | Наклон: например `italic`. |
| `text-align` | Горизонтальное выравнивание текста. |
| `text-decoration` | Линии текста: подчёркивание, зачёркивание, отсутствие линии. |
| `line-height` | Высота строки, расстояние между строками. |
| `letter-spacing` | Расстояние между буквами. |
| `text-shadow` | Тень текста. |

> `margin: 0 auto` центрирует блочный элемент по горизонтали, если у него задана ширина меньше контейнера.

## Flexbox и Grid

← к краткой карточке

Flexbox создаётся через `display: flex` или `display: inline-flex`. Главная ось зависит от `flex-direction`.

`justify-content` выравнивает по главной оси, `align-items` - по поперечной. Значение `align-items` по умолчанию - `stretch`.

`flex-grow` отвечает за рост, `flex-shrink` - за сжатие, `flex-basis` - за базовый размер.

Grid создаётся через `display: grid`. `grid-template-columns` задаёт столбцы, `grid-template-rows` - строки. `fr` - доля свободного места. `repeat()` сокращает повтор, `minmax()` задаёт минимум и максимум.

### Flexbox: термины

**Flex container**

Элемент с `display: flex` или `inline-flex`.

**Flex item**

Прямой дочерний элемент flex-контейнера.

**Main axis**

Главная ось. Её направление задаёт `flex-direction`.

**Cross axis**

Поперечная ось, перпендикулярная главной.

**`flex-direction`**

Направление главной оси: `row`, `column`, обратные варианты.

**`justify-content`**

Распределение элементов по главной оси.

**`align-items`**

Выравнивание элементов по поперечной оси.

**`flex-wrap`**

Разрешает или запрещает перенос элементов на новую строку.

**Flex item properties**

| Свойство | Смысл |
| --- | --- |
| `flex-grow` | Как элемент забирает свободное место. |
| `flex-shrink` | Как элемент сжимается при нехватке места. |
| `flex-basis` | Базовый размер до распределения свободного места. |
| `order` | Порядок отображения элемента. |
| `align-self` | Индивидуальное выравнивание одного элемента по поперечной оси. |

**Grid: термины**

**Grid container**
Элемент с `display: grid`.

**Grid item**
Прямой дочерний элемент grid-контейнера.

**Grid lines**
Линии, которые образуют сетку. Именно это правильный ответ на вопрос про линии сетки.

**Grid tracks**
Полосы между линиями: строки или столбцы.

**Grid cells**
Ячейки на пересечении строки и столбца.

**Grid areas**
Именованные или прямоугольные области из одной или нескольких ячеек.

**`fr`**

Доля свободного пространства.

**gap**

Расстояние между строками и столбцами.

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: auto minmax(200px, 1fr);
  gap: 16px;
}
```

> Flexbox чаще выбирают для меню, кнопок и выравнивания в одном направлении. Grid - для макета страницы и таблиц карточек в два измерения.

## Django models и ORM queries

← к краткой карточке

Модель - Python-класс таблицы. Поле - колонка. Объект - строка. Если не указать primary key, Django создаёт `id`.

`CharField` требует `max_length`. `null=True` разрешает NULL в БД, `blank=True` разрешает пустое значение в форме.

`ForeignKey` - многие к одному. `related_name` задаёт обратную связь. `ManyToManyField(..., through="Model")` задаёт промежуточную модель.

| Метод | Что возвращает |
|---|---|
| `all()` | QuerySet всех объектов |
| `filter()` | QuerySet по условию |
| `get()` | один объект или ошибка |
| `aggregate()` | словарь с итогом по выборке |
| `annotate()` | QuerySet с вычисляемым полем у каждого объекта |

`select_related` делает SQL JOIN для ForeignKey/OneToOne. `prefetch_related` делает отдельный запрос для ManyToMany и обратных связей.

### Основные термины Django models

**Model**

Класс Python, который наследуется от `models.Model` и описывает таблицу БД.

**Field**

Поле модели. Определяет колонку таблицы и правила работы со значением.

**Primary key**

Первичный ключ. Уникально идентифицирует строку таблицы.

**Migration**

Файл изменения схемы БД. Создаётся из изменений моделей.

**`makemigrations`**

Создаёт файлы миграций.

**`migrate`**

Применяет миграции к базе данных.

**`Class Meta`**

Внутренний класс настроек модели: сортировка, имена, ограничения.

**Manager**

Объект доступа к запросам, обычно `objects`.

### Поля модели

| Поле | Смысл |
|---|---|
| `CharField` | Короткая строка. Требует `max_length`. |
| `TextField` | Длинный текст без обязательного лимита длины. |
| `IntegerField` | Целое число. |
| `DecimalField` | Точное десятичное число, удобно для денег. Нужны `max_digits`, `decimal_places`. |
| `BooleanField` | `True` или `False`. |
| `DateField` | Дата без времени. |
| `DateTimeField` | Дата и время. |
| `EmailField` | Строка с email-валидацией на уровне формы/модели. |
| `FileField` | Путь к загруженному файлу. |
| `ImageField` | Файл изображения, требует поддержку обработки изображений. |
| `ForeignKey` | Связь многие-к-одному. |
| `OneToOneField` | Связь один-к-одному. |
| `ManyToManyField` | Связь многие-ко-многим. |

### Атрибуты полей

**`max_length`**
Максимальная длина строки. Обязателен для `CharField`.

**`null=True`**
Разрешает хранить `NULL` в базе данных.

**`blank=True`**
Разрешает пустое значение в форме и админке.

**`default`**
Значение по умолчанию.

**`primary_key=True`**
Делает поле первичным ключом.

**`unique=True`**
Значение должно быть уникальным.

**`verbose_name`**
Человекочитаемое имя поля.

**`related_name`**
Имя обратной связи для связанных объектов.

#### Связи

```
class Author(models.Model):
    name = models.CharField(max_length=100)

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.PROTECT,
related_name="books")
```

Прямая связь: `book.author`. Обратная связь через `related_name`: `author.books.all()`.

| `on_delete` | Что делает |
| --- | --- |

| | |
|---|---|
| `CASCADE` | Удаляет зависимые объекты вместе с главным. |
| `PROTECT` | Запрещает удалить объект, если на него есть ссылки. |
| `SET_NULL` | Записывает `NULL`, требует `null=True`. |
| `SET_DEFAULT` | Записывает значение по умолчанию. |
| `DO_NOTHING` | Django ничего не делает, ответственность остаётся на БД/коде. |

**QuerySet и ленивость**

QuerySet можно строить цепочкой, и Django не обязательно сразу идёт в БД. Запрос выполняется, когда данные реально нужны: цикл, индексация, вывод, `list()`, `count()`, `exists()`.

```
qs = Product.objects.filter(is_active=True).order_by("-price")
# БД ещё может не быть затронута
for product in qs:
    print(product.name)
```

**Lookups**

| Lookup | Смысл |
|---|---|
| `exact` | Точное совпадение. |
| `iexact` | Точное совпадение без учёта регистра. |
| `contains` | Содержит подстроку с учётом регистра. |
| `icontains` | Содержит подстроку без учёта регистра. |
| `gt`, `gte` | Больше, больше или равно. |
| `lt`, `lte` | Меньше, меньше или равно. |
| `in` | Значение входит в список. |
| `range` | Значение в диапазоне. |
| `isnull` | Проверка на `NULL`. |

**Q и F**

`Q` нужен для сложной логики `OR`, `AND`, `NOT`. `F` нужен, чтобы обращаться к значениям полей на стороне БД.

```
Product.objects.filter(Q(name__icontains="web") | Q(description__icontains="web"))
Product.objects.update(views=F("views") + 1)
```

> `aggregate()` считает один итог по выборке и возвращает словарь. `annotate()` добавляет вычисляемое поле к каждому объекту.

## Django views, URLs, templates, forms

← к краткой карточке

FBV - функция. Она принимает `request` и возвращает response. В `render()` первым аргументом должен быть `request`.

```
def home(request):
    return render(request, "index.html", {"title": "Home"})
```

CBV вызывается через `.as_view()`. `TemplateView` показывает шаблон, `ListView` показывает список, `DetailView` - один объект.

В `ListView` нужны `model` или `queryset`. Дополнительный context добавляют через `get_context_data()`, список меняют через `get_queryset()`.

`path(route, view, kwargs, name)` задаёт маршрут. `app_name` задаёт namespace. В шаблоне URL строится так: `{% url 'app:name' %}`.

В шаблонах переменные пишутся как `{{ value }}`, теги как `{% tag %}`, фильтры как `{{ value|lower }}`. Если переменной нет, обычно выводится пустая строка.

`forms.Form` - обычная форма. `forms.ModelForm` - форма на основе модели. Для связи между полями используют общий метод `clean()`. После `is_valid()` доступны `cleaned_data` и `errors`.

### Views

**View**

Код, который принимает HTTP-запрос и возвращает HTTP-ответ.

**FBV**

Function-Based View. Представление как обычная функция.

**CBV**

Class-Based View. Представление как класс с готовыми методами.

**`request`**

Объект HTTP-запроса: метод, пользователь, GET/POST-данные, cookies.

**`render()`**

Собирает HTML из шаблона и context, возвращает HTTP-ответ.

**`get_object_or_404()`**
Получает объект или возвращает 404, если объект не найден.

### CBV-классы

| Класс | Назначение |
|---|---|
| `TemplateView` | Показать шаблон без обязательной модели. |
| `ListView` | Показать список объектов. Нужен `model` или `queryset`. |
| `DetailView` | Показать один объект, обычно по `pk` или `slug`. |
| `CreateView` | Создать объект через форму. |
| `UpdateView` | Изменить объект через форму. |
| `DeleteView` | Удалить объект, обычно после подтверждения. |

### Методы CBV

| Метод/атрибут | Смысл |
|---|---|
| `as_view()` | Превращает класс view в callable для URL-маршрута. |
| `get_queryset()` | Возвращает список объектов для `ListView`. |
| `get_object()` | Возвращает один объект для `DetailView`. |
| `get_context_data()` | Возвращает словарь context для шаблона. |
| `context_object_name` | Задаёт имя переменной объекта/списка в шаблоне. |
| `template_name` | Явно задаёт путь к шаблону. |
| `paginate_by` | Включает пагинацию списка. |

### URLs

```
app_name = "blog"

urlpatterns = [
    path("posts/", PostListView.as_view(), name="post_list"),
```

```
    path("posts/<int:pk>/", PostDetailView.as_view(), name="post_detail"),
]
```

**`urlpatterns`**
Список маршрутов приложения или проекта.

**`path()`**
Функция маршрутизации с простыми path-конвертерами.

**`re_path()`**
Маршрут на основе регулярного выражения.

**`reverse()`**
Строит URL по имени маршрута в Python.

**`name`**
Имя маршрута, чтобы не хардкодить URL.

**`app_name`**
Namespace приложения, например `blog:post_list`.

### Path-конвертеры

| Конвертер | Что принимает |
|---|---|
| `<str:name>` | Строка без слеша. По умолчанию. |
| `<int:pk>` | Положительное целое число. |
| `<slug:slug>` | Строка из букв, цифр, дефиса и подчёркивания. |
| `<uuid:id>` | UUID. |
| `<path:subpath>` | Строка, которая может содержать слеши. |

### Django Template Language

| Синтаксис | Смысл |
|---|---|

| | |
|---|---|
| `{{ variable }}` | Вывод переменной. |
| `{% tag %}` | Шаблонный тег: цикл, условие, url, block. |
| `{# comment #}` | Комментарий шаблона. |
| `{{ value\|lower }}` | Фильтр. |
| `{% for item in items %}` | Цикл. |
| `{{ forloop.counter }}` | Номер итерации цикла с 1. |
| `{% extends "base.html" %}` | Наследование шаблона. |
| `{% block content %}` | Область, которую переопределяет дочерний шаблон. |
| `{% include "part.html" %}` | Включение другого шаблона. |
| `{% load static %}` | Загрузка тегов static. |
| `{% static "css/app.css" %}` | Путь к статическому файлу. |

### Django forms

**`forms.Form`**
Форма без прямой привязки к модели.

**`forms.ModelForm`**
Форма на основе модели. Поля задаются через `Meta`.

**`is_valid()`**
Запускает валидацию формы.

**`cleaned_data`**
Очищенные и приведённые к Python-типам данные.

**`errors`**
Ошибки формы.

**`clean_field()`**
Валидация одного поля.

**`clean()`**
Валидация всей формы, подходит для связи между полями.

**`widget`**
HTML-представление поля, например `PasswordInput`.

| Поле формы | Назначение |
|---|---|
| `CharField` | Строка. |
| `IntegerField` | Целое число. |
| `BooleanField` | Логическое значение. |
| `ChoiceField` | Выбор из списка. |
| `EmailField` | Email. |
| `DateField` | Дата. |
| `FileField` | Файл. |

```
class RegisterForm(forms.Form):
    password = forms.CharField(widget=forms.PasswordInput)
    password_repeat = forms.CharField(widget=forms.PasswordInput)

    def clean(self):
        cleaned_data = super().clean()
        if cleaned_data.get("password") != cleaned_data.get("password_repeat"):
            raise forms.ValidationError("Пароли не совпадают")
        return cleaned_data
```

> Для проверки связи между полями нужен общий `clean()`, а не `clean_password()`, потому что сравниваются два значения.