

Подготовка к экзамену по веб-разработке

Краткий конспект по темам из файла подготовки. Оглавление ведёт к короткой карточке темы; ссылка в карточке ведёт к подробному блоку внизу; подробный блок возвращает обратно к краткому разделу.

Оглавление

Блок 1. Общие вопросы

- Адресация, IPv4, IPv6
- TCP
- Маска подсети
- NAT, localhost, подсети
- DNS
- HTTP
- HTTP-запрос/ответ
- HTTPS
- HTTP/3
- Порты
- Идентификация/auth
- OAuth 2.0
- URI, URL, URN

Блок 2. HTML

- Структура документа
- Комментарии
- head
- Мета-теги
- Нормальный поток
- Блочные и строчные
- Семантика
- Атрибуты
- class
- id
- Текст, списки, таблицы
- Картинки и ссылки
- Пробелы и переносы

Блок 3. HTML Forms

- Форма
- id/name
- radio/checkbox/submit
- label
- Группировка
- required
- placeholder
- button type
- textarea

Блок 4. CSS

- Универсальный селектор
- Элемент/class/id
- Комбинаторы
- Атрибуты
- Псевдоклассы
- Группировка
- Каскад
- Специфичность
- Блочная модель
- Фон
- Margin
- Шрифты
- Текст
- Интервалы
- Тень текста

Блок 5. Flexbox и Grid

Flex-контейнер

inline-flex

Выравнивание

grow/shrink/basis

Grid-контейнер

fr

repeat/minmax

Блоки 6-12. Django

Модели

Первичный ключ

Миграции

Class Meta

null/blank

CharField

ForeignKey

Many-to-Many

FBV

render

Контекст

get_object_or_404

Маршрутизация FBV

Template/List/DetailView

CBV и модели

Доп. контекст

Переменные по умолчанию

get_queryset/get_object

Паджинация

Шаблоны CBV

super()

Маршрутизация CBV

Django URLs

DTL

Django queries

Django forms

Блок 1. Общие вопросы

Адресация в Интернете. IPv4. IPv6.

В TCP/IP есть физический адрес, IP-адрес и символьное DNS-имя. IPv4 — 32 бита, 4 октета по 0-255; IPv6 — 128 бит, шестнадцатеричная запись и намного больше адресов.

[Подробнее: IP-адресация](#)

Протокол TCP

TCP — транспортный протокол с установлением соединения, порядком пакетов, подтверждениями и повторной передачей. Надёжнее UDP, но тяжелее по задержкам.

[Подробнее: TCP](#)

Маска подсети

Маска отделяет сетевую часть IPv4-адреса от хостовой. `/24` значит 24 бита сети и 8 бит хоста; `255.255.255.0` — классический пример такой маски.

[Подробнее: подсети](#)

Распределение IP, NAT, localhost, подсети

Публичные адреса распределяются через ICANN/IANA и региональные регистраторы. NAT переводит приватные адреса во внешний адрес. `127.0.0.1`/`localhost` указывает на сам хост.

[Подробнее: NAT и localhost](#)

DNS. DNS-серверы

DNS сопоставляет доменные имена с IP-адресами. Цепочка обычно идёт через кэширующий resolver, корневые серверы, TLD-серверы и авторитативные серверы домена.

[Подробнее: DNS](#)

Протокол HTTP

HTTP — прикладной протокол WWW. Клиент отправляет запрос, сервер возвращает ответ. HTTP stateless: связь между запросами не хранится без cookies, сессий или токенов.

[Подробнее: HTTP](#)

HTTP-запрос, ответ, методы, заголовки, коды

Сообщение состоит из стартовой строки, заголовков, пустой строки и тела. Методы: `GET`, `POST`, `HEAD`, `PUT`, `PATCH`, `DELETE`. Коды: `2xx` успех, `3xx` редирект, `4xx` ошибка клиента, `5xx` ошибка сервера.

[Подробнее: HTTP-сообщения](#)

Протокол HTTPS

HTTPS — HTTP поверх TLS. Даёт шифрование, проверку подлинности сервера через сертификат и защиту от подмены данных. Порт по умолчанию — `443`.

[Подробнее: HTTPS](#)

Протокол HTTP/3

HTTP/3 работает поверх QUIC/UDP. Семантика HTTP сохраняется, но транспорт меняется: независимые потоки, встроенный TLS 1.3, меньше блокировок при потере пакетов.

[Подробнее: HTTP/3](#)

Сетевые порты. HTTP и HTTPS

Порт указывает процесс внутри IP-адреса. Всего 0-65535. HTTP обычно `80/tcp`, HTTPS `443/tcp`, DNS `53`, SSH `22`, альтернативный HTTP часто `8080`.

[Подробнее: порты](#)

Идентификация, аутентификация, авторизация

Идентификация: "кто ты?". Аутентификация: "докажи". Авторизация: "что тебе разрешено?". В HTTP часто встречаются Basic, Digest и Bearer.

[Подробнее: auth](#)

OAuth 2.0

OAuth 2.0 — протокол авторизации: приложение получает ограниченный доступ к ресурсам пользователя через токен, не получая пароль пользователя.

[Подробнее: OAuth 2.0](#)

URI, URL, URN

URI — общий идентификатор ресурса. URL указывает местоположение и способ получения. URN задаёт устойчивое имя без привязки к расположению. Все URL — URI, но не все URI — URL.

[Подробнее: URI/URL/URN](#)

Блок 2. HTML

Структура документа

HTML5-документ начинается с `<!doctype html>`, затем `<html>`, внутри `<head>` с метаданными и `<body>` с видимым содержимым.

[Подробнее: HTML](#)

Комментарии в HTML

Комментарий пишется как `<!-- текст -->`. Браузер его не отображает, но он виден в исходном коде.

[Подробнее: HTML](#)

Теги внутри head

В `<head>` обычно находятся `<title>`, `<meta>`, `<link>`, `<style>`, `<script>` и технические данные страницы.

[Подробнее: HTML](#)

Мета-теги и viewport

`<meta charset="utf-8">` задаёт кодировку. `viewport` нужен адаптивной вёрстке: `width=device-width, initial-scale=1.0`.

[Подробнее: HTML](#)

Нормальный поток

Нормальный поток — базовое расположение элементов без `flex/grid/position`: блочные идут сверху вниз, строчные текут внутри строки.

[Подробнее: HTML](#)

Блочные и строчные элементы

Блочные занимают доступную ширину и начинаются с новой строки: ``div``, ``p``, ``section``. Строчные занимают место по содержимому: ``span``, ``a``, ``strong``.

[Подробнее: HTML](#)

Семантические теги

``header``, ``nav``, ``main``, ``section``, ``article``, ``aside``, ``footer`` описывают смысл блоков. Это важно для доступности, SEO и читаемости кода.

[Подробнее: HTML](#)

Атрибуты тегов

Атрибуты уточняют элемент: ``href``, ``src``, ``alt``, ``class``, ``id``, ``name``, ``type``. Обычно пишутся в открывающем теге.

[Подробнее: HTML](#)

Атрибут class

`class` задаёт один или несколько CSS-классов. Один класс можно использовать на многих элементах.

[Подробнее: HTML](#)

Идентификатор элемента

`id` должен быть уникален на странице. Используется для якорей, JS и CSS-селектора `#id`.

[Подробнее: HTML](#)

Заголовки, абзацы, списки, таблицы

`h1-h6` задают структуру заголовков, `p` — абзац, `ul/ol/li` — списки, `table/tr/th/td` — табличные данные.

[Подробнее: HTML](#)

Картинки и ссылки

Ссылка: `<a href="...">`. Картинка: ``; `alt` важен для доступности и случаев, когда изображение не загрузилось.

[Подробнее: HTML](#)

Переносы строк и пробелы

В HTML последовательности пробелов схлопываются. Перенос строки в коде обычно не равен переносу на странице; для явного переноса есть `<br>`.

[Подробнее: HTML](#)

Блок 3. HTML Forms

Форма и атрибуты

``form`` собирает пользовательский ввод. Главные атрибуты: ``action`` — куда отправлять, ``method`` — ``get`` или ``post``.

[Подробнее: HTML forms](#)

id и name

``id`` связывает поле с ``label`` и должен быть уникальным. ``name`` — имя параметра, под которым значение уйдёт на сервер.

[Подробнее: HTML forms](#)

radio, checkbox, submit

``radio`` выбирает один вариант в группе с одинаковым ``name``. ``checkbox`` — независимый флажок. ``submit`` отправляет форму.

[Подробнее: HTML forms](#)

Label

``label`` подписывает поле и улучшает доступность. Связь делается через ``for="field-id"`` или вложением `input` внутрь `label`.

[Подробнее: HTML forms](#)

Группировка элементов

``fieldset`` объединяет связанные поля, ``legend`` задаёт подпись группы. Часто используется для `radio`-групп.

[Подробнее: HTML forms](#)

required

``required`` запрещает отправку формы браузером, если поле пустое. Серверную валидацию всё равно нужно делать отдельно.

[Подробнее: HTML forms](#)

Placeholder

``placeholder`` показывает подсказку внутри пустого поля. Это не замена ``label``, потому что исчезает при вводе.

[Подробнее: HTML forms](#)

type для button

``button`` без ``type`` внутри формы часто ведёт себя как ``submit``. Для обычной кнопки явно пишут ``type="button"``.

[Подробнее: HTML forms](#)

Textarea

`textarea` — многострочное текстовое поле. Текст по умолчанию пишется между открывающим и закрывающим тегом.

[Подробнее: HTML forms](#)

Блок 4. CSS selectors и базовые свойства

Универсальный селектор

`\*` выбирает все элементы. Часто применяют для сброса `box-sizing`, но специфичность у него нулевая.

[Подробнее: CSS selectors](#)

Элемент, класс, идентификатор

`p` выбирает теги, `.card` — класс, `#main` — уникальный id. ID специфичнее класса и тега.

[Подробнее: CSS selectors](#)

Дочерний, потомок, сестринский

`A B` — потомок, `A > B` — прямой ребёнок, `A + B` — следующий сосед, `A ~ B` — последующие соседи.

[Подробнее: CSS selectors](#)

Селектор атрибута

`[attr]`, `[type="email"]`, `a[href^="https"]` выбирают элементы по наличию или значению атрибута.

[Подробнее: CSS selectors](#)

Псевдоклассы

`:hover`, `:focus`, `:checked`, `:nth-child()` выбирают состояние или позицию элемента. Структурные псевдоклассы зависят от места в DOM.

[Подробнее: CSS selectors](#)

Группировка селекторов

Селекторы через запятую получают одно правило: `h1, h2, h3 { margin: 0; }`.

[Подробнее: CSS selectors](#)

Наследование и каскадирование

Каскад решает конфликт правил по важности, специфичности и порядку. Наследуются в основном текстовые свойства, например `color` и `font-family`.

[Подробнее: CSS selectors](#)

Специфичность

Условный вес: inline > ID > классы/атрибуты/псевдоклассы > теги/псевдоэлементы. При равенстве побеждает правило ниже в коде.

[Подробнее: CSS selectors](#)

Блочная модель

Блок состоит из content, padding, border и margin. ``box-sizing: border-box`` включает padding и border в заданную ширину.

[Подробнее: CSS свойства](#)

Фон элемента

Основные свойства: ``background-color``, ``background-image``, ``background-repeat``, ``background-position``, ``background-size``.

[Подробнее: CSS свойства](#)

Margin и выравнивание блоков

``margin`` задаёт внешний отступ. Для центрирования блочного элемента с заданной шириной часто используют ``margin-left/right: auto``.

[Подробнее: CSS свойства](#)

Шрифты

``font-family``, ``font-size``, ``font-weight``, ``font-style`` задают семейство, размер, толщину и наклон текста.

[Подробнее: CSS свойства](#)

Выравнивание и подчёркивание текста

``text-align`` управляет горизонтальным выравниванием текста, ``text-decoration`` — подчёркиванием и другими линиями.

[Подробнее: CSS свойства](#)

line-height и letter-spacing

``line-height`` задаёт высоту строки, ``letter-spacing`` — расстояние между буквами. На читаемость влияют сильнее, чем кажется.

[Подробнее: CSS свойства](#)

Тень текста

``text-shadow: x y blur color;`` добавляет тень к тексту. В тестах важно помнить порядок смещения, размытия и цвета.

[Подробнее: CSS свойства](#)

Блок 5. Flexbox Layout. Grid Layout

Flex-контейнер и оси

`display: flex` создаёт flex-контейнер. Главная ось задаётся `flex-direction`, поперечная идёт перпендикулярно.

[Подробнее: Flex/Grid](#)

Строчный flex-контейнер

`display: inline-flex` делает контейнер flex-контейнером, но сам контейнер ведёт себя как строчный элемент во внешнем потоке.

[Подробнее: Flex/Grid](#)

Выравнивание по осям

`justify-content` выравнивает по главной оси, `align-items` — по поперечной. `align-content` работает при нескольких строках.

[Подробнее: Flex/Grid](#)

flex-grow, shrink, basis

`flex-basis` — базовый размер, `flex-grow` — как элемент растёт, `flex-shrink` — как сжимается при нехватке места.

[Подробнее: Flex/Grid](#)

Grid-контейнер

`display: grid` создаёт двумерную сетку. Колонки задаются `grid-template-columns`, строки — `grid-template-rows`.

[Подробнее: Flex/Grid](#)

Величина fr

`fr` — доля свободного пространства grid-контейнера. `1fr 2fr` делит свободное место в пропорции 1 к 2.

[Подробнее: Flex/Grid](#)

repeat и minmax

`repeat(3, 1fr)` повторяет трек.
`minmax(200px, 1fr)` задаёт минимум и максимум размера трека.

[Подробнее: Flex/Grid](#)

Блоки 6-12. Django

Создание моделей

Модель — класс-наследник `models.Model`; поля класса описывают столбцы таблицы. Модели обычно лежат в `models.py` приложения.

[Подробнее: Django models](#)

Первичный ключ

Первичный ключ уникально идентифицирует запись. Если не задать явно, Django создаёт поле `id` автоматически.

[Подробнее: Django models](#)

Миграции

`makemigrations` создаёт файлы миграций из изменений моделей, `migrate` применяет их к базе данных.

[Подробнее: Django models](#)

Class Meta

`class Meta` задаёт метаданные модели: `ordering`, `verbose_name`, `db_table`, ограничения и другие настройки.

[Подробнее: Django models](#)

null=True и blank=True

`null=True` разрешает `NULL` в БД.
`blank=True` разрешает пустое значение на уровне форм/валидации Django.

[Подробнее: Django models](#)

CharField

`CharField` хранит короткую строку и требует `max_length`. Для длинного текста обычно берут `TextField`.

[Подробнее: Django models](#)

ForeignKey

`ForeignKey` задаёт связь многие-к-одному. `on_delete` определяет поведение при удалении связанного объекта; `related_name` задаёт имя обратной связи.

[Подробнее: Django models](#)

Many-to-Many

`ManyToManyField` задаёт связь многие-ко-многим. Django создаёт промежуточную таблицу автоматически, либо можно указать свою через `through`.

[Подробнее: Django models](#)

FBV

Function-Based View — функция, которая принимает `request` и возвращает `HttpResponse`, `render`, `redirect` или ошибку.

[Подробнее: Django FBV](#)

render

`render(request, template, context)` рендерит шаблон с контекстом и возвращает `HttpResponse`.

[Подробнее: Django FBV](#)

Передача данных в шаблон

Данные передаются словарём `context`: ключи становятся именами переменных в шаблоне.

[Подробнее: Django FBV](#)

Объект или 404

`get_object_or_404(Model, pk=pk)` возвращает объект или генерирует HTTP 404, если записи нет.

[Подробнее: Django FBV](#)

Маршрутизация FBV

В `urlpatterns` функция передаётся напрямую: `path("posts/<int:pk>/", views.detail, name="detail")`.

[Подробнее: Django FBV](#)

TemplateView, DetailView, ListView

`TemplateView` показывает шаблон, `DetailView` — один объект, `ListView` — список объектов.

[Подробнее: Django CBV](#)

CBV и модели

Для `ListView`/`DetailView` можно указать `model = Post`; Django сам построит `queryset` и стандартные имена шаблонов.

[Подробнее: Django CBV](#)

Дополнительные данные

Дополнительный контекст обычно добавляют через `get_context_data`, вызывая `super()` и дополняя словарь.

[Подробнее: Django CBV](#)

Переменные по умолчанию

`ListView` отдаёт `object_list` и `modelname_list`; `DetailView` отдаёт `object` и `modelname`.

[Подробнее: Django CBV](#)

Переопределение выгрузки

`get_queryset()` меняет список объектов, `get_object()` — получение одного объекта. Часто используют фильтрацию по пользователю или статусу.

[Подробнее: Django CBV](#)

Паджинация

В `ListView` достаточно задать `paginate_by = N`; в шаблоне появится `page_obj`.

[Подробнее: Django CBV](#)

Шаблоны по умолчанию

Для `DetailView`: `app/model_detail.html`; для `ListView`: `app/model_list.html`, если не задан `template_name`.

[Подробнее: Django CBV](#)

Вызов базовой логики

При переопределении методов вроде `get_context_data` обычно сначала вызывают `super()`, чтобы не потерять стандартный контекст.

[Подробнее: Django CBV](#)

Маршрутизация CBV

Класс подключается через `.as_view()`:
`path("", PostListView.as_view(), name="post_list")`.

[Подробнее: Django CBV](#)

Django URLs

`path()` задаёт маршрут, `view`, `kwargs` и имя. `re_path()` использует `regex`. `reverse()` строит URL по имени маршрута.

[Подробнее: Django URLs](#)

Язык шаблонов Django

Переменные пишутся как `{{ value }}`, теги как `{% for %}`, фильтры как `{{ name|lower }}`. Наследование делается через `extends` и `block`.

[Подробнее: Django templates](#)

Django queries

`QuerySet` ленивый и цепочечный. Важно знать `get`, `filter`, `exclude`, `order_by`, `annotate`, `aggregate`, `Q`, `F`, `select_related`, `prefetch_related`.

[Подробнее: Django queries](#)

Django forms

`forms.Form` — ручная форма, `ModelForm` строится по модели. Валидация через `is_valid`, `cleaned_data`, `clean_field`, `clean`.

[Подробнее: Django forms](#)

Подробные заметки

IP-адресация, IPv4, IPv6

Vault: [IP-адрес, IPv6](#)

В интернете используются адреса разных уровней: физический адрес интерфейса, сетевой IP-адрес и символическое DNS-имя. IPv4 записывается четырьмя октетами, например `192.168.1.10`. IPv6 использует 128 бит и записывается группами шестнадцатеричных чисел.

В тесте часто спрашивают: IPv4 = 32 бита, IPv6 = 128 бит, IPv4 октет = 0-255, IPv6 не совместим напрямую с IPv4.

[Назад к краткой теме](#)

TCP

Vault: [TCP vs UDP](#)

TCP устанавливает соединение перед передачей данных, делит данные на сегменты, нумерует их, подтверждает доставку и повторяет потерянные данные. За это платит дополнительными задержками и служебным трафиком.

В тесте: TCP — надёжный, connection-oriented; UDP — быстрее и проще, но без гарантий доставки и порядка.

[Назад к краткой теме](#)

Подсети и маска подсети

Vault: [Подсети и маска подсети](#)

Маска подсети определяет, какие биты IPv4-адреса относятся к сети, а какие к хосту. Запись `/24` равна маске `255.255.255.0`: первые 24 бита — сеть, последние 8 — хосты.

Запомнить: чем больше префикс `/N`, тем меньше адресов в подсети.

[Назад к краткой теме](#)

NAT, приватные адреса, localhost

Vault: [NAT](#), [Подсети и маска подсети](#)

NAT переводит приватные IPv4-адреса во внешний публичный адрес. Частые приватные диапазоны: `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`. Loopback-диапазон `127.0.0.0/8` не используется для обмена между узлами сети; `127.0.0.1` и `localhost` указывают на текущую машину.

В тесте: NAT экономит публичные IPv4 и скрывает внутреннюю адресацию, но ломает прямую end-to-end связность.

[Назад к краткой теме](#)

DNS и DNS-серверы

Vault: [DNS](#)

DNS — распределённая база данных, которая по доменному имени возвращает IP-адрес или другие записи. Типичная цепочка: локальный кэш/hosts, resolver провайдера, root-серверы, TLD-серверы, авторитативные серверы домена.

В тесте: root не знает IP конкретного сайта, он направляет к TLD; авторитативный сервер знает записи домена.

[Назад к краткой теме](#)

HTTP и HTTP-сообщения

Vault: [HTTP, HTTPS и HTTP/3](#)

HTTP — протокол прикладного уровня. Запрос и ответ состоят из стартовой строки, заголовков, пустой строки и необязательного тела. Методы описывают действие над ресурсом, заголовки передают метаданные, коды ответа показывают результат обработки.

Метод	Смысл
`GET`	Получить ресурс
`POST`	Отправить данные
`PUT`	Создать/заменить ресурс
`PATCH`	Частично изменить ресурс
`DELETE`	Удалить ресурс

В тесте: `4xx` — проблема на стороне клиента, `5xx` — проблема на стороне сервера.

[Назад к HTTP](#) · [Назад к HTTP-сообщениям](#)

HTTPS

Vault: [HTTP, HTTPS и HTTP/3](#)

HTTPS — это HTTP, защищённый TLS. Сертификат связывает домен с ключами и подтверждает подлинность сервера. TLS обеспечивает шифрование, целостность и защиту от подмены трафика.

В тесте: HTTP порт `80`, HTTPS порт `443`; Basic auth без HTTPS небезопасен.

[Назад к краткой теме](#)

HTTP/3

Vault: [HTTP, HTTPS и HTTP/3](#)

HTTP/3 переносит HTTP на QUIC поверх UDP. QUIC делает потоки независимыми: потеря пакета не блокирует все запросы внутри соединения, как это возможно в HTTP/2 поверх TCP. TLS 1.3 встроен в QUIC.

В тесте: HTTP/3 не меняет методы и коды HTTP, меняет транспорт: UDP/QUIC вместо TCP.

[Назад к краткой теме](#)

Сетевые порты

Vault: [Сетевые порты](#)

Порт уточняет приложение-получателя внутри хоста. Номер порта находится в диапазоне `0-65535`. Порты TCP и UDP независимы: `53/tcp` и `53/udp` — разные конечные точки.

Минимум к запоминанию: HTTP `80`, HTTPS `443`, DNS `53`, SSH `22`, SMTP `25`.

[Назад к краткой теме](#)

Идентификация, аутентификация, авторизация

Vault: [Веб-аутентификация и авторизация](#)

Идентификация сообщает системе идентификатор субъекта. Аутентификация проверяет подлинность. Авторизация выдаёт права на действия. В HTTP встречаются схемы Basic, Digest и Bearer.

В тесте не путать: логин — идентификация, пароль/код — аутентификация, доступ к админке — авторизация.

[Назад к краткой теме](#)

OAuth 2.0

Vault: [OAuth 2.0](#)

OAuth 2.0 позволяет стороннему приложению получить ограниченный доступ к ресурсам пользователя. Вместо пароля приложение получает токен с конкретными правами и сроком жизни.

В тесте: OAuth 2.0 — авторизация, не парольная аутентификация. Для входа поверх OAuth обычно нужен OpenID Connect.

[Назад к краткой теме](#)

URI, URL, URN

Vault: [URI, URL, URN](#)

URL показывает, где находится ресурс и как его получить: схема, хост, порт, путь, параметры, фрагмент. URN задаёт стабильное имя ресурса без места получения. URI — общее понятие для идентификаторов ресурсов.

Главная формула: $URL \subset URI$; $URN \subset URI$; не каждый URI является URL.

[Назад к краткой теме](#)

HTML

Vault: [Основы HTML \(структура, таблицы, формы\)](#)

HTML описывает структуру документа. Важно знать базовый каркас, роль `head` и `body`, блочные и строчные элементы, семантические теги, атрибуты `class`/`id`, ссылки, изображения, списки и таблицы.

В тесте: `class` можно повторять, `id` должен быть уникальным; таблицы — для табличных данных, не для макета.

[Назад к краткому блоку HTML](#)

HTML Forms

Vault: [Основы HTML \(структура, таблицы, формы\)](#)

Форма отправляет данные на сервер. ``action`` задаёт URL, ``method`` — способ отправки. ``label`` связывается с полем через ``for`` и ``id``. ``name`` определяет имя параметра при отправке. ``required`` и типы `input` помогают браузерной валидации, но не заменяют серверную проверку.

В тесте: у `radio`-кнопок одной группы должен быть одинаковый ``name``; у ``button`` внутри формы лучше явно задавать ``type``.

[Назад к краткому блоку HTML Forms](#)

CSS selectors

Vault: [CSS: Основы и селекторы](#)

CSS-селектор выбирает элементы, к которым применяются правила. Нужно различать селекторы тега, класса, `id`, атрибута, псевдокласса и комбинаторы. Каскад учитывает важность, специфичность и порядок правил.

Специфичность: `inline` > `ID` > класс/атрибут/псевдокласс > тег/псевдоэлемент.

[Назад к краткому блоку CSS](#)

CSS базовые свойства

Vault: [CSS: Основы и селекторы](#)

Блочная модель состоит из `content`, `padding`, `border` и `margin`. ``box-sizing`` меняет расчёт ширины. Текстовые свойства управляют шрифтами, выравниванием, межстрочным расстоянием, расстоянием между буквами и декором текста. Фон задаётся через группу ``background-*``.

В тесте: ``width`` при ``content-box`` не включает `padding/border`, при ``border-box`` включает.

[Назад к краткому блоку CSS](#)

Flexbox и Grid

Vault: [CSS: Адаптивность, Flexbox и Grid](#)

Flexbox — одномерная модель раскладки по главной и поперечной оси. Grid — двумерная сетка строк и колонок. Flex удобен для распределения элементов в строке или колонке, Grid — для общего макета страницы.

В тесте: `justify-content` работает по главной оси flex, `align-items` — по поперечной; `fr` — доля свободного пространства Grid.

[Назад к краткому блоку Flex/Grid](#)

Django models

Vault: [Django Модели и ORM](#)

Модели описывают структуру таблиц и связи между ними. `CharField` требует `max_length`; `ForeignKey` задаёт `on_delete`; `ManyToManyField` создаёт промежуточную таблицу или использует указанную через `through`. Миграции переносят изменения моделей в БД.

В тесте: `null=True` относится к БД, `blank=True` — к формам/валидации Django.

[Назад к краткому блоку Django](#)

Django FBV

Vault: [Django: Представления, шаблоны и CBV](#)

FBV — обычная функция-представление. Она принимает `request`, получает данные, вызывает `render` или возвращает другой `HttpResponse`. Маршрутизируется через `path()` прямой передачей функции.

В тесте: `render(request, template, context)` возвращает HTTP-ответ; `get_object_or_404` возвращает объект или 404.

[Назад к краткому блоку Django](#)

Django CBV

Vault: [Django: Представления, шаблоны и CBV](#)

CBV — классовые представления с готовой логикой. `TemplateView` показывает шаблон, `ListView` выводит список, `DetailView` выводит один объект. В URL подключаются через `.as_view()`.

В тесте: `ListView` по умолчанию даёт `object_list`, `DetailView` — `object`; шаблоны по умолчанию `model_list.html` и `model_detail.html`.

[Назад к краткому блоку Django](#)

Django URLs

Vault: [Django: Представления, шаблоны и CBV](#)

`urlpatterns` связывает URL с представлениями. `path()` использует `route`, `view`, `kwargs` и `name`. Конвертеры вроде `int`, `slug`, `uuid`, `path` извлекают параметры. `reverse()` и `{% url %}` строят URL по имени маршрута.

В тесте: не хардкодить URL, если есть именованный маршрут и `reverse()`.

[Назад к краткой теме](#)

Язык шаблонов Django

Vault: [Django: Представления, шаблоны и CBV](#)

DTL использует переменные `{{ value }}`, теги `{% tag %}` и фильтры `{{ value|filter }}`. Наследование строится через `{% extends %}` и `{% block %}`. Статика подключается через `{% load static %}` и `{% static 'path' %}`.

В тесте: функции Python с произвольными аргументами напрямую из шаблона вызывать нельзя; используют теги, фильтры или подготовку данных во `view`.

[Назад к краткой теме](#)

Django queries

Vault: [Django Модели и ORM](#)

QuerySet ленив: запрос выполняется при фактическом обращении к данным. Методы можно цепочечно комбинировать. `get()` возвращает один объект или ошибку, `filter()` возвращает QuerySet, `aggregate()` возвращает итоговый словарь, `annotate()` добавляет вычисляемые поля к объектам.

В тесте: `select_related` — для ForeignKey/OneToOne через JOIN; `prefetch_related` — для ManyToMany и обратных связей отдельными запросами.

[Назад к краткой теме](#)

Django forms

Vault: [Django: Работа с формами и CUD](#)

`forms.Form` описывает форму вручную, `forms.ModelForm` строится по модели через `Meta`. `is_valid()` запускает валидацию, `cleaned_data` содержит очищенные данные, `clean_()` валидирует поле, `clean()` — форму целиком. `widget` управляет HTML-представлением поля.

В тесте: `form.save()` есть у ModelForm; серверная валидация обязательна, HTML `required` не защита.

[Назад к краткой теме](#)

[Наверх](#)