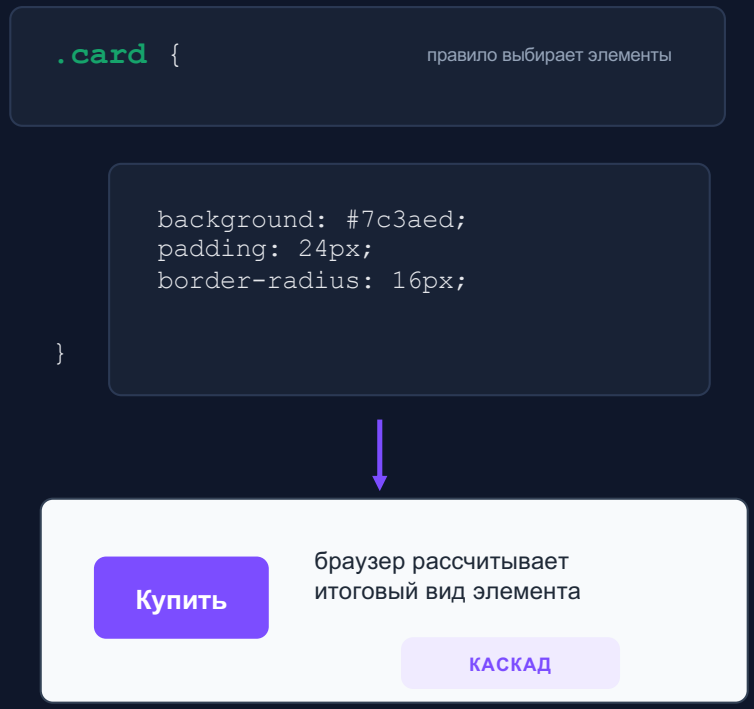
<css>

CSS — каскадность, селекторы и раскладка

style, специфичность, наследование,
box model и position



Максим

старший разработчик · Яндекс Еда

Что успеем за 90 минут

01	**От атрибута style к правилам CSS**	ОСНОВЫ	0–15 мин
02	Селекторы, состояния и части элементов	АДРЕСАЦИЯ	15–35 мин
03	Каскад, специфичность и наследование	ВЫБОР ЗНАЧЕНИЯ	35–55 мин
04	**Нормальный поток, display и box model**	ГЕОМЕТРИЯ	55–75 мин
05	**position, Flexbox**	РАСКЛАДКА	75–90 мин

По ходу — короткие вопросы аудитории и две мини-практики в DevTools.

КОНТЕКСТ

HTML отвечает «что это», CSS — «как выглядит»

HTML

Структура и смысл

index.html

```
1 <article class="product">
2   <h2>Кофе</h2>
3   <button class="buy">
4     Купить
5   </button>
6 </article>
```

- какие элементы есть
- как они связаны
- что они означают

CSS

Представление

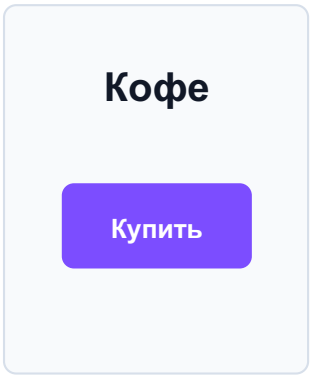
styles.css

```
1 .product {
2   padding: 24px;
3   border-radius: 16px;
4 }
5 .buy { color: white; }
```

- цвета и типографика
- размеры и отступы
- раскладка и состояния

БРАУЗЕР

Результат



Один HTML можно оформить по-разному

Атрибут style: CSS прямо внутри HTML

index.html

```
<button
  style="
    background: #7c3aed;
    color: white;
    padding: 12px 20px;
    border: 0;
    border-radius: 10px;
  ">
  Купить
</button>
```

АТРИБУТ Внутри — несколько CSS-объявлений, разделённых точкой с запятой.

Результат

Купить

Когда уместно

- ✓ быстрый эксперимент
- ✓ значение вычисляется JavaScript-ом

Почему не для основной вёрстки

- ● дублирование и трудно переиспользовать
- ● нет :hover, media queries и общих правил

Объявление CSS: свойство, значение и единица

```
font-size: 20px;
```

свойство — значение — разделитель

ЦВЕТ

```
#7c3aed
rgb(124, 77, 255)
hsl(258, 100%, 65%)
```

РАЗМЕР

```
16px · 1.2rem
50% · 100vw
2em · 40vh
```

КЛЮЧЕВОЕ СЛОВО

```
auto · none
center · block
inherit · transparent
```

ФУНКЦИЯ

```
calc(100% - 24px)
clamp(16px, 2vw, 24px)
var(--accent)
```

Не любое значение подходит любому свойству: браузер игнорирует невалидное объявление.

DevTools: откуда пришёл стиль и что вычислил браузер

http://localhost:5500/card.html

Карточка товара

Купить

выбранный DOM-узел

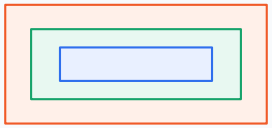
Elements | Styles | Computed | Layout

```
element.style {
    color: white;
}

.button {
    background: #7c3aed;
    color: blue;
    padding: 12px 20px;
}
```

COMPUTED

color	rgb(255, 255, 255)
padding	12px 20px



Styles: правила, источники, зачёркнутые значения

Computed: итоговое значение и геометрия элемента

МИНИ-ДЕМО

Три способа подключить CSS

style="..."

Inline

index.html

```
<p style="color:red;">
  Текст
</p>
```

- очень локально
- не переиспользуется
- нет :hover и @media

<style>

Внутри документа

index.html

```
<style>
  .note { color: red; }
</style>
```

- удобно для демо
- работает на одной странице
- HTML и CSS смешаны

<link>

Отдельный файл

HTML + styles.css

```
<link rel="stylesheet"
      href="styles.css">

.note { color: red; }
```

- переиспользование
- кэширование
- основной вариант

РЕКОМЕНДУЕТСЯ

Правило CSS: селектор + блок объявлений

styles.css

```
.card {
  background: white;
  padding: 24px;
  border-radius: 16px;
}
```

СЕЛЕКТОР — СВОЙСТВО — ЗНАЧЕНИЕ

Декларация = property: value;

Одно правило

может выбрать много элементов

`.card` `.card` `.card`

Один элемент

может совпасть с несколькими правилами

`button` `.button` `#buy`

Кто победит? Это и решает каскад.

Карта селекторов: как CSS находит элементы

ПРОСТЫЕ

`div · .card · #app · *`

по имени, классу, id

СВЯЗИ

`.card > h2 · h2 + p`

по положению в DOM

АТРИБУТЫ

`[disabled] · [type="email"]`

по данным HTML

СОСТОЯНИЯ

`:hover · :checked · :focus-visible`

что происходит сейчас

СТРУКТУРА

`:first-child · :nth-child(2n)`

место среди соседей

ЧАСТИ

`::before · ::placeholder`

псевдоэлементы

Тег, класс, id и универсальный селектор

`p`

По тегу

все элементы данного типа

базовая типографика

`.card`

По классу

переиспользуемая группа

основной способ стилизации

`#app`

По id

уникальный элемент

якорь / JS, редко стили

`*`

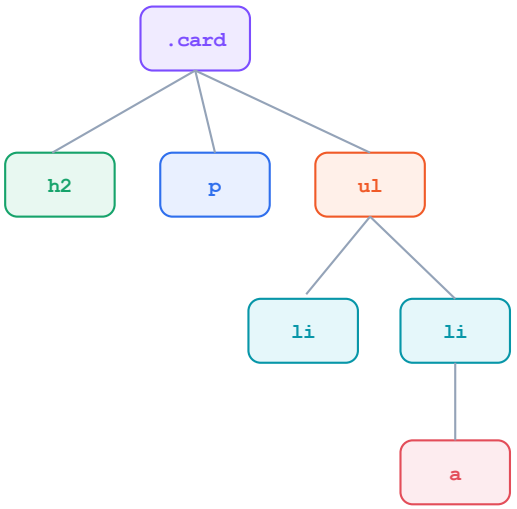
Универсальный

любой элемент

reset и служебные правила

```
*, *::before, *::after { box-sizing: border-box; }
```

Комбинаторы описывают отношения в DOM



DOM-дерево

A B	**Потомок**	`.card a`	любой a внутри .card
A > B	**Прямой ребёнок**	`ul > li`	li ровно на один уровень ниже
A + B	**Следующий сосед**	`h2 + p`	первый p сразу после h2
A ~ B	**Все последующие соседи**	`h2 ~ p`	все p после h2 у того же родителя

Селекторы по атрибутам

styles.css

```css
[disabled] { opacity: .5; }

input[type="email"] { ... }

a[href^="https"] { ... }

a[href$=".pdf"] { ... }

[data-state="open"] { ... }

[class~="tag"] { ... }
```

ПРИСУТСТВИЕ	`[disabled]`	атрибут есть независимо от значения
ТОЧНОЕ ЗНАЧЕНИЕ	`[type="email"]`	тип поля равен email
НАЧАЛО / КОНЕЦ СТРОКИ	`^=` · `$=`	внешние ссылки, PDF и другие паттерны
СОСТОЯНИЕ КОМПОНЕНТА	`[data-state="open"]`	стили следуют данным HTML / JS

Селектор лишь выбирает элемент — он не меняет сам атрибут.

Псевдоклассы: состояние и положение элемента

Состояния взаимодействия

`:hover` `:focus-visible` `:checked` `:disabled`

```
.button:hover { transform: translateY(-1px); }

.button:focus-visible { outline: 3px solid #2563eb; }

input:checked + label { font-weight: 700; }
```

КЛАВИАТУРА **:focus-visible нужен не меньше, чем :hover**

Положение в структуре

`:first-child`	первый ребёнок
`:last-child`	последний ребёнок
`:nth-child(2n)`	чётные позиции
`:not(.active)`	всё, кроме…

Псевдокласс не создаёт новый узел — он уточняет условие выбора.

Псевдоэлементы: части элемента и генерируемый контент

Бейдж без дополнительного тега

styles.css

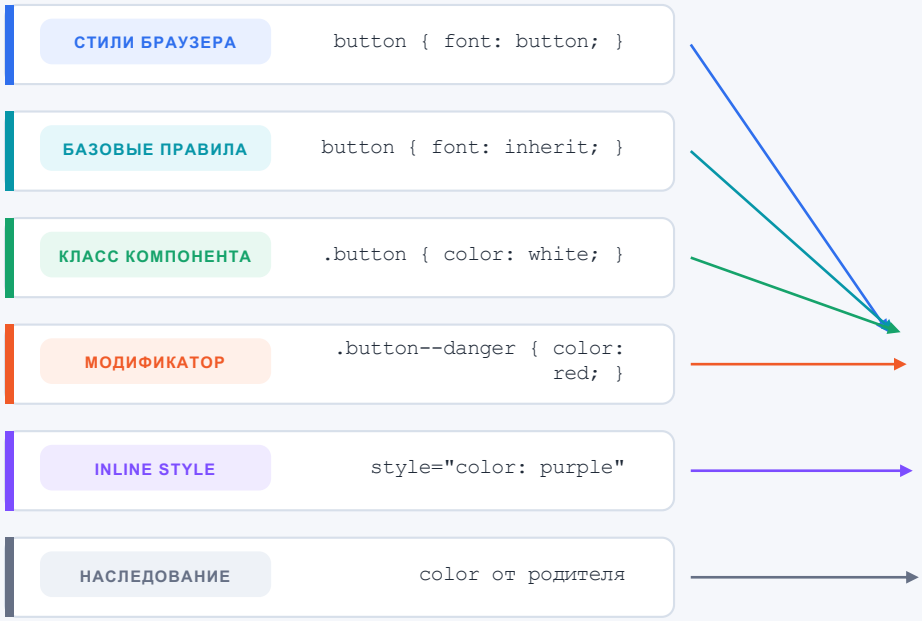
```
.card::before {
  content: "NEW";
  position: absolute;
  top: 8px;
  right: 8px;
}
```

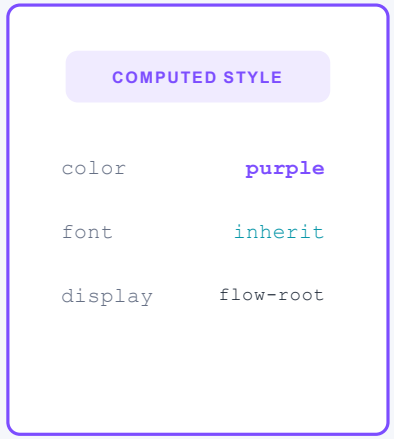
Карточка NEW

::before / ::after	генерируемая часть, нужен content
::placeholder	подсказка внутри поля
::selection	выделенный пользователем текст
::first-line	первая строка форматированного текста

Не помещайте важный смысл только в content: он может быть недоступен или не копироваться.

Почему таблицы стилей называются каскадными





КАСКАД

выбирает одно значение из конкурирующих деклараций

НАСЛЕДОВАНИЕ

передаёт значение от родителя потомку

Не путать: каскад ≠ наследование

Как браузер выбирает итоговое значение

01 **Совпало ли правило?** — селектор подходит, условие @media истинно, значение валидно

↓

02 **Источник и важность** — слои, origin, обычное или !important

↓

03 **Специфичность** — насколько точно селектор описывает элемент

↓

04 **Порядок в коде** — при полном равенстве побеждает более поздняя декларация

ПРИМЕР

styles.css + HTML

```
p { color: blue; }

.note { color: green; }



<p class="note">Текст</p>
```

Побеждает .note

оба правила совпали, но класс специфичнее тега

Для начала используем упрощённую модель. Слои каскада и анимации разберём позже.

Источники стилей и ловушка !important

Упрощённая лестница приоритета

!important

```
color: red !important;
```

- переворачивает приоритет обычных деклараций
- но среди important снова работают specificity и order

Почему это не «быстрое решение»

- сложнее переопределять компонент
- возникает гонка !important
- скрывает проблему архитектуры селекторов

ИСКЛЮЧЕНИЯ БЫВАЮТ

Пользовательские important-стили могут побеждать авторские — это важно для доступности.

Специфичность: сравниваем селекторы по разрядам

INLINE › ID › CLASS › TYPE

`style="…"` | `#app` | `.card [type] :hover` | `div ::before`

Сравниваем слева направо: 0-1-2-0 сильнее 0-0-20-0

Селектор	Вес	Комментарий
`*`	0-0-0-0	универсальный
`p`	0-0-0-1	один тип
`.card p`	0-0-1-1	класс + тип
`#app .card`	0-1-1-0	id + класс
`style="…"`	1-0-0-0	inline

Не десятичное число

0-1-0-0 не равно «100»
и не складывается как обычная сумма.

* и комбинаторы > + ~
не добавляют специфичности.

`:where(...) = 0`

Кто победит: специфичность, затем порядок

index.html

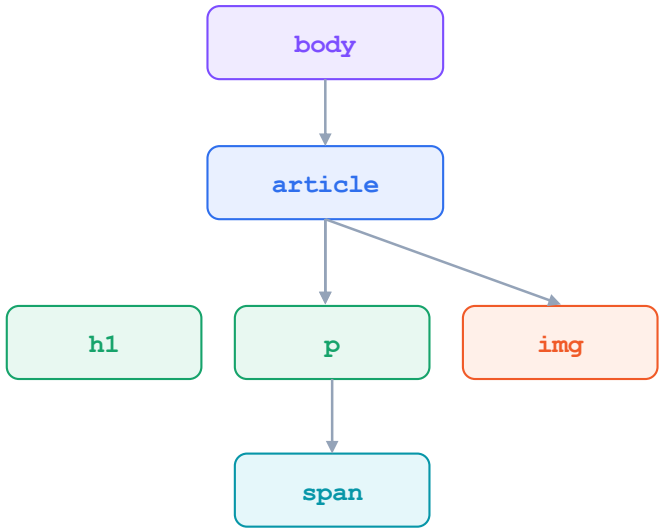
```
1 <article class="card">
2   <button
3     id="buy"
4     class="button primary"
5     style="color:red">
6     Купить
7   </button>
8 </article>
```

`button { color: blue; }`	0-0-0-1	проигрывает
`.card .button { color: orange; }`	0-0-2-0	ничья по весу
`.button.primary { color: green; }`	0-0-2-0	ниже в коде → выигрывает
`#buy { color: purple; }`	0-1-0-0	побеждает классы
`style="color:red"`	1-0-0-0	побеждает обычные правила

Итоговый цвет: RED

Пауза перед ответом: попросите аудиторию проговорить алгоритм.

Наследование: значения идут по DOM-дереву



color · font-family · line-height — часто наследуются

styles.css

```
1 body {
2   color: #334155;
3   font-family: Arial;
4 }
5 article { border: 1px solid; }
```

Наследуются

color · font-* · line-height · text-align

Обычно не наследуются

margin · padding · border · width · position

Явно управляем наследованием

inherit

взять вычисленное значение у родителя

```
color: inherit;
```

initial

вернуться к начальному значению из спецификации

```
display: initial;
```

unset

inherit для наследуемых, иначе initial

```
margin: unset;
```

revert

откатить к предыдущему источнику или слою

```
all: revert;
```

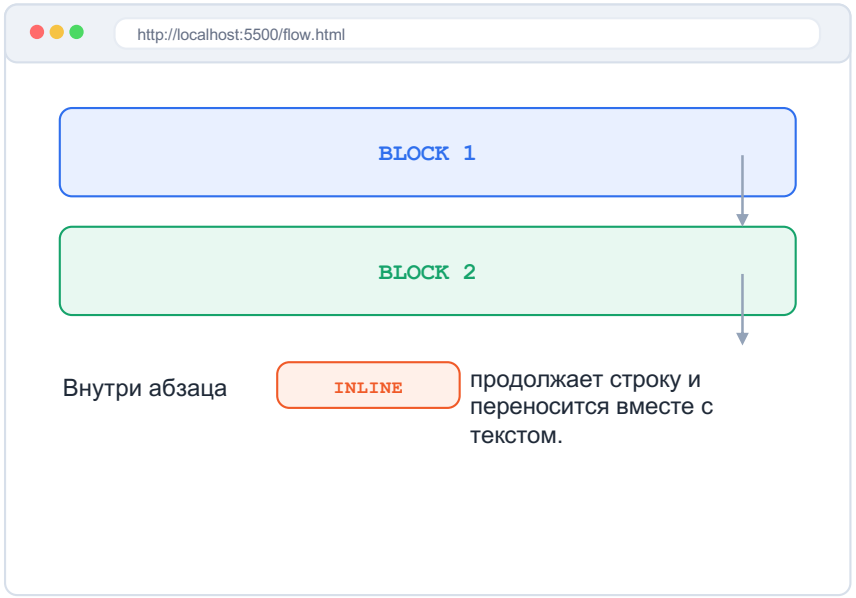
Практический reset для элементов формы

```
button, input, textarea, select { font: inherit; color: inherit; }
```

Осторожно с all: unset

Можно случайно убрать визуальные признаки фокуса и привычное поведение контролов.

Нормальный поток: браузер уже умеет раскладывать элементы



Три опоры normal flow

01 Порядок DOM

элементы идут в той последовательности, в которой описаны

02 Тип форматирования

block строит вертикальный поток, inline — строковый

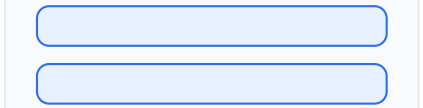
03 Место резервируется

элемент в потоке влияет на положение соседей

Сначала доверяем потоку — потом меняем layout.

display: block, inline и inline-block

block



- начинается с новой строки
- width: auto занимает доступную ширину
- width / height работают

inline

Текст `span` продолжается на следующей строке

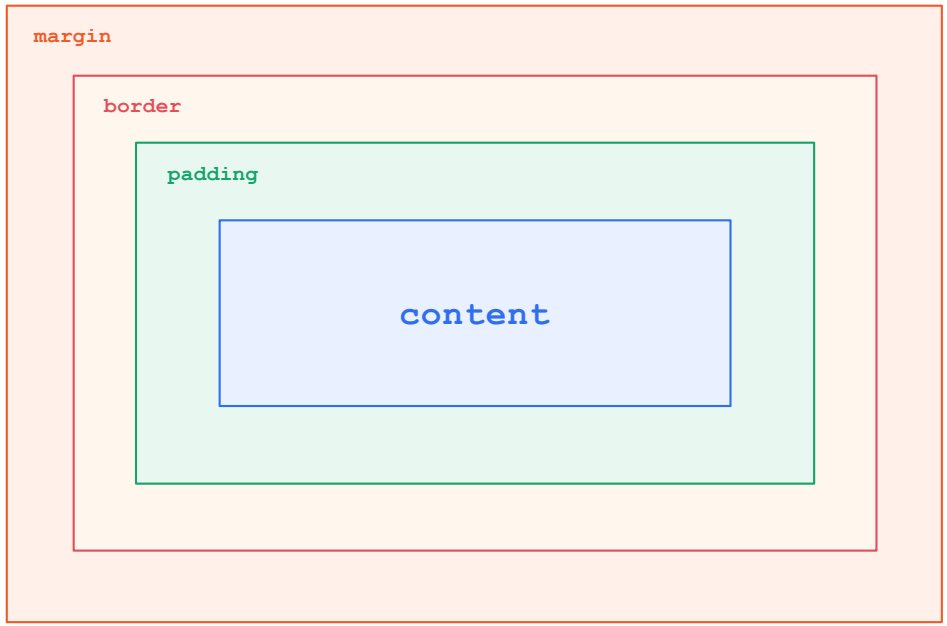
- остаётся внутри строки
- размер зависит от содержимого
- width / height обычно не задают коробку

inline-block

- располагается в строке
- сохраняет собственную коробку
- width / height и отступы работают

`display: none` удаляет элемент из раскладки; `visibility: hidden` скрывает, но сохраняет место.

Box model: каждый элемент — прямоугольный блок



DevTools → Computed показывает эту схему для выбранного узла

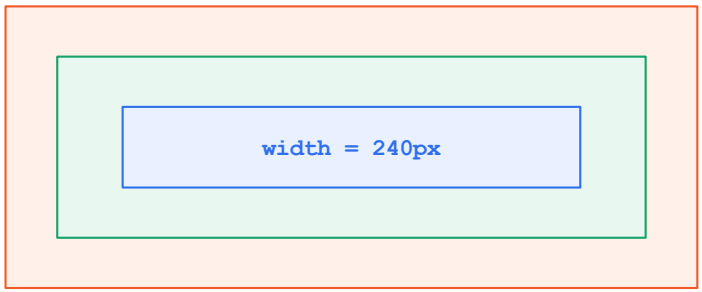
content	текст, изображение или дочерние элементы
padding	пространство между контентом и рамкой
border	рамка вокруг padding и content
margin	прозрачный внешний отступ между блоками

Фон рисуется в области content + padding и под прозрачной рамкой. Margin всегда прозрачен.

Размеры элемента и box-sizing

content-box · по умолчанию

width задаёт только content

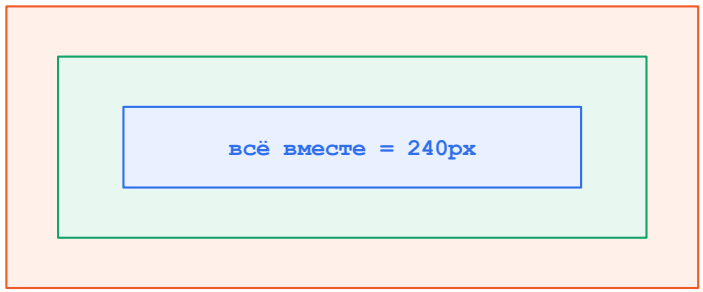


итого = width + padding + border

margin остаётся снаружи и в ширину блока не входит

border-box · удобнее

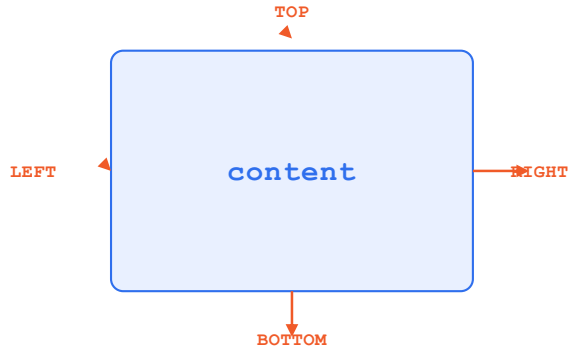
width включает padding и border




```
*, *::before, *::after { box-sizing: border-box; }
```

Padding, border, margin и сокращённая запись

Порядок значений — по часовой стрелке




```
padding: 20px 12px 16px 8px; /* T R B L */
```

```
padding: 12px;
```
все стороны

```
padding: 12px 20px;
```
вертикаль · горизонталь

```
padding: 12px 20px 16px;
```
top · horizontal · bottom

```
padding: 12px 20px 16px 8px;
```
top · right · bottom · left

Рамка и скругление

```
border: 1px solid #cbd5e1;
```

```
border-radius: 12px;
```

Нюансы размеров: min/max, overflow и margin collapse

MIN / MAX

```
.card {
  width: 100%;
  max-width: 520px;
  min-height: 180px;
}
```

- ✓ max-width защищает от слишком широкого блока
- ✓ min-height задаёт нижнюю границу, но не обрезает контент

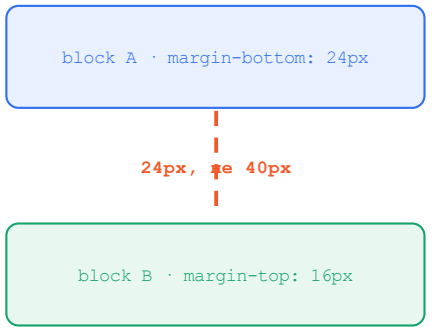
OVERFLOW

Очень длинный текст, который не помещается в заданную высоту контейнера…

`overflow: auto`

- visible — по умолчанию
- hidden / clip — обрезать
- auto — прокрутка при необходимости

VERTICAL MARGIN



Вертикальные margin соседних block-элементов иногда схлопываются: берётся больший, а не сумма.

Flex/Grid-контейнеры этого эффекта между items не имеют.

position меняет участие в потоке и систему координат

Значение	В потоке?	Относительно чего?	Типичный случай
static	да	normal flow	значение по умолчанию
relative	да	своей исходной позиции	небольшой сдвиг, containing block
absolute	нет	предка с position ≠ static	бейдж, overlay, иконка в поле
fixed	нет	viewport	модалка, плавающая кнопка
sticky	да → «липнет»	scroll-контейнера	заголовок секции, таблицы

Смещения задаются top / right / bottom / left или сокращённо inset.

relative и absolute: локальная система координат

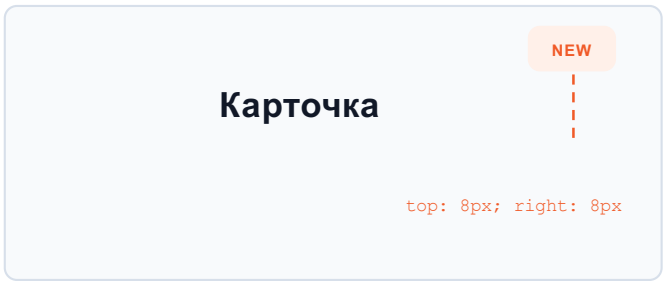
position: relative

Элемент остаётся в потоке


```
.item { position: relative; top: 12px; left: 24px; }
```

position: absolute

Элемент выпадает из потока

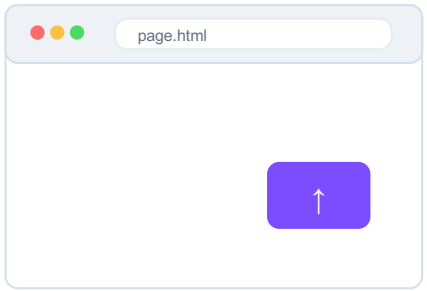



```
.card { position: relative; }

.badge { position: absolute; inset: 8px 8px auto auto; }
```

fixed, sticky и z-index

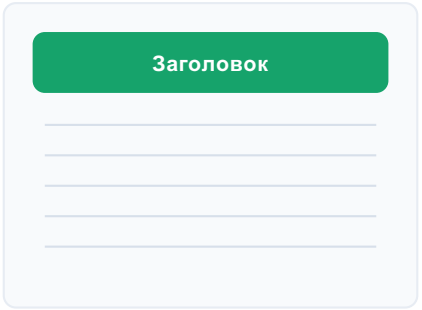
position: fixed



Привязан к viewport и не двигается при прокрутке.

Модалка · FAB · панель

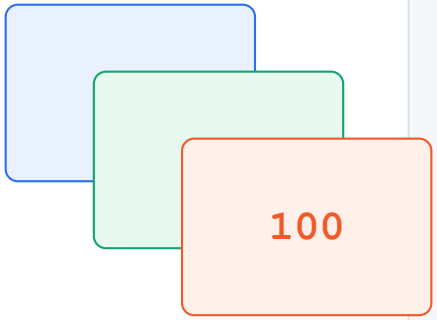
position: sticky



Сначала в потоке, затем «липнет» при достижении top.

Нужен top / bottom и scroll-контейнер

z-index



Сравнивает слои внутри одного stacking context.

Большое число не «пробивает» другой контекст.

Выбираем инструмент раскладки: flow, Flexbox, Grid или position

Normal flow	Текст, документ, вертикальная последовательность?	оставить поток
Flexbox	Ряд или колонка, выравнивание и расстояния?	display: flex
Grid	Две оси: строки и колонки?	display: grid
position	Наложение, бейдж, sticky/fixed-интерфейс?	position: …

FLEXBOX · ПРЕВЬЮ

```
.toolbar {
  display: flex;
  align-items: center;
  justify-content: space-between;
  gap: 16px;
}
```

Не используйте absolute для построения обычной сетки страницы.

Итоги занятия

CSS — это система выбора значений и раскладки, а не набор случайных свойств.

1. **style — CSS без селектора**

   удобен точечно, но не заменяет правила

2. **Каскад выбирает победителя**

   важность → специфичность → порядок

3. **Селекторы адресуют DOM**

   классы, связи, атрибуты и состояния

4. **Раскладка начинается с потока**

   display → box model → Flex/Grid → position

Спасибо! Вопросы?