ДИСЦИПЛИНА	Фронтенд и бэкенд разработка
ИНСТИТУТ	ИПТИП
КАФЕДРА	Индустриального программирования
ВИД УЧЕБНОГО МАТЕРИАЛА	Методические указания к практическим занятиям
ПРЕПОДАВАТЕЛЬ	Загородних Николай Анатольевич
СЕМЕСТР	1 семестр, 2025/2026 уч. год

Практическое занятие 15. Комбинирование методов адаптивной вёрстки

Цель: собрать устойчивую и доступную главную страницу, комбинируя «жидкий» фундамент, медиазапросы (MQ), Flex/Grid, контейнерные запросы (CQ) и responsive images.

База макета: упрощённая структура из примера методички (занятие 13): header / nav / hero / sections / footer.

Контекст: в этом занятии — прикладная укладка с рабочими фрагментами кода, разбором и типичными ошибками. На лекциях уже обсуждали теорию — здесь практикуем.

0) Preflight: корректный viewport и каркас страницы

Вставьте в <head>:

```
<meta name="viewport" content="width=device-width, initial-scale=1, viewport-fit=cover">
```

Эта строка говорит мобильному браузеру:

- **width=device-width** — выставь «макетную ширину» равной ширине устройства (а не ~980px, как было на старых мобилах по умолчанию). Иначе сайт будет «сжат» и медиазапросы по ширине работать криво.
- **initial-scale=1** — открой страницу без начального зума (1 CSS-px мапится на 1 «логический» пиксель при текущем DPR).
- **viewport-fit=cover** — **разреши занимать всю физическую область экрана**, включая зоны под вырезами/скруглениями/домашней полосой на iOS. После этого ответственность «не залезть под вырез» на нас → учитываем **safe-area** через env() в CSS.

На десктопах и большинстве Android всё это безопасно игнорируется или ведёт себя «по учебнику». Особая магия — это iOS с вырезами.

Базовый каркас:

```
:root { color-scheme: light dark; }

html, body { height: 100%; }

body {
  margin: 0;
  font: 16px/1.6 system-ui, -apple-system, "Segoe UI", Roboto, Arial, sans-
serif;
  background: #f8fafc;
  color: #0b1020;
}

.page {
  min-height: 100vh;   /* фолбэк для старых браузеров (может не учитывать
панели на мобилках) */
  min-height: 100svh;  /* стабильный минимум: высота экрана с панелями
(всегда влезет) */
  min-height: 100dvh;  /* на новых браузерах — актуальная видимая высота,
обновляется при скролле */
 display: flex;
 flex-direction: column;
}

main { flex: 1; }
```

Почему так:

- viewport-fit=cover учитывает безопасные области на устройствах с вырезами.
- min-height с svh/dvh стабилизирует высоту на мобильных без «прыжков».
- color-scheme: light dark; включает адекватные системные цвета для светлой/тёмной тем.
- **vh** — старое «1% высоты *вьюпорта*». На мобилках часто **не равно** видимой области: адресная строка/жестовая панель не учитываются → блок может «выползать» за нижний край или прыгать.
- **svh** = **S**mall Viewport Height — «самая маленькая» высота экрана: когда все панели **видны**. Даёт **стабильный минимум**, контент гарантированно влезет без перекрытий.
- **dvh** = **D**ynamic Viewport Height — «текущая» видимая высота. Когда панели прячутся/появляются при скролле, значение **обновляется**. Полезно, если нужно реально заполнять экран «от края до края» даже после скрытия панелей.

В CSS действует правило: **последняя поддерживаемая строка выигрывает**.

Поэтому каскад из трёх деклараций — это **прогрессивное улучшение**:

- Браузер без новых единиц возьмёт 100vh.
- С поддержкой новых, но без dvh — возьмёт 100svh.
- Современный — возьмёт 100dvh.

1) Семантика + «ссылка-пропуск» + видимый фокус

HTML:

```
<body>
```

```
  <a class="skip-link" href="#main">Перейти к основному содержимому</a>

  <header class="header">…</header>
  <nav class="nav" aria-label="Основная навигация">…</nav>
  <main id="main" class="main">…</main>
  <footer class="footer">…</footer>
</body>
```

CSS (skip-link и фокус):

```
.skip-link{
  position: absolute;
  inset-inline-start: 1rem;
  inset-block-start: -100%;
  z-index: 10000;
  padding: .5rem .75rem;
  background: #111;
  color: #fff;
  border-radius: .5rem;
  text-decoration: none;
}

.skip-link:focus,
.skip-link:focus-visible { inset-block-start: 1rem; }

:focus-visible { outline: 3px solid #2563eb; outline-offset: 2px; }
a, button, input, textarea { outline-offset: 2px; }

/* Крупные цели на устройствах с pointer:coarse */
@media (pointer: coarse){
  a, button, .nav__link { min-height: 44px; padding: .75rem 1rem; }
}
```

Разбор:

skip-link позволяет клавиатурным пользователям миновать повторяющиеся блоки. :focus-visible подсвечивает фокус только при навигации с клавиатуры.

Типичные ошибки: прятать skip-link через display:none; отключать фокус без альтернативы.

2) Sticky-хедер и безопасные области (вырезы)

```
.nav{
  position: sticky;
  top: 0;

  /* Прибавки для вырезов/жестовых панелей — только где они есть */
  padding-block-start: calc(.75rem + env(safe-area-inset-top));
  padding-block-end: .75rem;
  padding-inline-start: calc(var(--space-md, 1rem) + env(safe-area-inset-
left));
  padding-inline-end:   calc(var(--space-md, 1rem) + env(safe-area-inset-
right));

  background: #2563eb;
  color: #fff;
  z-index: 100;
  isolation: isolate; /* свой контекст для теней/фильтров */
  box-shadow: 0 1px 3px rgba(0,0,0,.1);
}
```

Разбор: env(safe-area-inset-*) даст отступы на устройствах с вырезами; на обычных — прибавка 0. isolation:isolate не даёт «протекать» эффектам.

Типичные ошибки: фиксированные «прокладки» под «чёлку»; position: fixed без нужды.

3) Герой-картинка (LCP) без CLS + современные форматы

В нашем макете «герой» — это **круглая аватарка до 200px**. Значит слот реальной ширины ≈ **200px**. Соответственно, sizes должен быть 200px, а srcset — 150/200/300w.

HTML:

```
<figure class="hero__image">
  <picture>
    <!-- Современные форматы -->
    <source
      type="image/avif"
      srcset="images/hero-150.avif 150w, images/hero-200.avif 200w,
images/hero-300.avif 300w"
      sizes="200px">
    <source
      type="image/webp"
      srcset="images/hero-150.webp 150w, images/hero-200.webp 200w,
images/hero-300.webp 300w"
      sizes="200px">

    <!-- Запасной JPEG -->
    <img class="hero__photo"
         src="images/hero-200.jpg"
         srcset="images/hero-150.jpg 150w, images/hero-200.jpg 200w,
images/hero-300.jpg 300w"
         sizes="200px"
         width="200" height="200"
         alt="Портрет студента"
         fetchpriority="high">
  </picture>
</figure>
```

CSS:

```
.hero__photo{
  inline-size: 200px;      /* или: clamp(150px, 22vw, 200px); тогда sizes
меняем соответствующе */
  block-size: auto;
  border-radius: 50%;
  object-fit: cover;
  box-shadow: 0 10px 15px -3px rgb(0 0 0 / 0.1), 0 4px 6px -4px rgb(0 0 0 /
0.1);
}
```

Разбор:

width/height фиксируют соотношение сторон — нет «прыжков» (CLS≈0). Поскольку слот ≈200px, sizes="200px" и лёгкие варианты в srcset экономят трафик. fetchpriority="high" — только у LCP.

Типичные ошибки: указывать sizes="100vw" или 72rem для маленькой аватарки; не ставить width/height.

4) Проекты/карточки — авто-адаптивная Grid-сетка

```
.projects-preview { margin-block: var(--space-2xl, 3rem); }

.projects-preview__grid{
  display: grid;  /* включаем CSS Grid */
  grid-template-columns: repeat(auto-fit, minmax(18rem, 1fr));
  gap: var(--space-lg, 1.5rem);  /* промежутки между колонками/строками */
}

.project-card{
  background: #fff;
  border-radius: .75rem;
  box-shadow: 0 4px 6px -1px rgb(0 0 0 / 0.1), 0 2px 4px -2px rgb(0 0 0 /
0.1);
  transition: transform .2s ease, box-shadow .2s ease;
  min-width: 0; /* разрешаем контенту ужиматься */
}

.project-card:hover{ transform: translateY(-4px); }

.project-card__title,
.project-card__description { overflow-wrap: anywhere; }
```

Разбор:

display: grid — включаем грид-раскладку.

gap — расстояние между карточками (и по горизонтали, и по вертикали).

grid-template-columns: repeat(auto-fit, minmax(18rem, 1fr)) — вся «магия» адаптива:

- minmax(18rem, 1fr) — каждая колонка **не уже 18rem**, но может **расти до доли свободного места** (1fr).

- repeat(auto-fit, …) — «создай столько таких колонок, сколько **поместится** в контейнер».

Итог: браузер сам подбирает число колонок так, чтобы **ни одна** не стала уже 18rem. Если места хватает — колонок больше. Если места меньше — колонок меньше. Остаток ширины внутри ряда делится поровну (по 1fr) между видимыми колонками.

Как браузер «считает» это пошагово

1. Берём минимальную ширину колонки **MIN = 18rem**.

   (Если у пользователя базовый шрифт 16px, то 18rem ≈ **288px**.)

2. Учитываем **gap** (по умолчанию 1.5rem ≈ **24px** при 16px базового размера).
3. Сколько колонок влезет? Примерная формула для порогов:

N колонок поместятся, если

width_container ≥ N * MIN + (N - 1) * gap

При MIN=288px и gap=24px получаются «естественные» пороги:

- 1 → 2 колонки: 2×288 + 1×24 = **600px**
- 2 → 3 колонки: 3×288 + 2×24 = **912px**
- 3 → 4 колонки: 4×288 + 3×24 = **1224px**

Т.е. при ширине контейнера:

- < 600px — **1 колонка**
- 600–911px — **2 колонки**
- 912–1223px — **3 колонки**
- ≥ 1224px — **4 колонки**

…и так далее. **Ни одного медиазапроса не писали** — брейкпоинты «получились сами».

Важно: если базовый размер шрифта у пользователя не 16px (например, 18px ради доступности), то и rem-ы больше — пороги **масштабируются автоматически**. Это плюс в сторону доступности.

Почему карточки красивые и равные

1fr у верхней границы minmax говорит: «раздай остаток поровну». Поэтому все колонки в ряду **одинаковой ширины** и аккуратно заполняют строку.

gap управляет «жёлобами», а не margin у карточек, поэтому сетка не «ломается».

Зачем часто добавляют min-width: 0 у карточки

По умолчанию грид-элемент имеет min-width: auto (то есть по содержимому). Если внутри длинное слово/URL, колонку может распирать.

Добавьте у карточки:

```
.project-card{ min-width: 0; }
.project-card__title,
.project-card__description{ overflow-wrap: anywhere; } /* или break-word */
```

Теперь контент **будет ужиматься и переноситься**, а не ломать сетку.

auto-fit vs auto-fill — какую разницу вы вообще увидите

auto-fit (как у тебя): «пустые» треки **схлопываются** → если карточек в последней строке мало, оставшиеся колонки **растянутся** и заполнят строку.

auto-fill: «пустые» треки **сохраняются** → остаются «слоты» под несуществующие карточки, строка визуально может не растягиваться.

Для карточек проектов обычно **auto-fit** выглядит естественнее.

Что, если карточки «раздуваются» на очень широком экране

Если не хочешь, чтобы каждая колонка становилась слишком широкой, ограничь максимум и выровняй по центру:

```
.projects-preview__grid{
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(18rem, 22rem)); /* максимум
22rem */
  gap: var(--space-lg, 1.5rem);
  justify-content: center; /* центрируем сетку, когда есть свободное место */
}
```

Браузер пытается разместить как можно больше колонок, где каждая — не уже **18rem**. Когда помещается N колонок с учётом gap, он ставит N. Оставшееся пространство делит поровну между колонками (за счёт 1fr). Поэтому на узких экранах будет 1 колонка, потом 2, 3, 4 — **без медиазапросов**. Значения в rem масштабируются вместе с пользовательским размером шрифта. Если текст внутри карточки длинный, добавьте у .project-card min-width: 0 и overflow-wrap: anywhere;, чтобы контент не распирал колонку.

5) Контейнерные запросы (CQ) для компонента — подробно

Идея. Вместо «слушать окно» (@media) компонент «слушает» **ширину своего контейнера**. Так он корректно ведёт себя и в узком сайдбаре, и в широкой колонке без глобальных брейкпоинтов.

Разметка (важно, нужен внешний контейнер):

```
<section class="projects-area">
  <div class="projects-list">
    <!-- карточки -->
  </div>
</section>
```

CSS с фолбэком через @supports:

```
/* Базовое поведение для всех браузеров */
.projects-list{
  display: grid;
  grid-template-columns: 1fr;                    /* одна колонка по умолчанию
*/
  gap: var(--space-lg, 1.5rem);
}

/* Прогрессивное улучшение: только если CQ поддерживаются */
@supports (container-type: inline-size){
  .projects-area{
    /* Делаем область контейнером для CQ */
    container-type: inline-size;                 /* реагируем на ширину
(inline) */
    container-name: projects;                    /* по имени удобнее
таргетировать */
  }

  /* Когда КОНТЕЙНЕР шире 640px — 2 колонки */
  @container projects (min-width: 640px){
    .projects-list{ grid-template-columns: repeat(2, 1fr); }
```

```
  }
  /* Когда шире 960px — 3 колонки */
  @container projects (min-width: 960px){
    .projects-list{ grid-template-columns: repeat(3, 1fr); }
  }
}
```

Как это работает:

- container-type: inline-size включает «измеритель» ширины у .projects-area.

  (Если нужно реагировать и на высоту — используйте size, но это реже оправдано.)

- container-name: projects даёт имя; дальше в правилах используем

  @container projects (min-width: …) — читаемо и не заденем другие контейнеры.

- Внутри @supports — чтобы старые браузеры просто остались на базовой 1-колоночной сетке.

Как выбирать пороги (640/960):

берём желаемую **минимальную ширину карточки**, например 18rem ≈ 288px при 16px.

С учётом gap ~24px получаем естественные пороги:

- 2 колонки: 2×288 + 24 ≈ 600px ⇒ берём **640px** с запасом под паддинги.
- 3 колонки: 3×288 + 2×24 ≈ 912px ⇒ берём **960px**.

Частые ошибки и ловушки:

- Забыли container-type → условия @container никогда не сработают.
- Дублируют те же пороги в @media и @container → конфликтующие правила. Решение: глобальные @media — для **глобального** (например, шрифт всей страницы), @container — для **локального поведения компонентов**.
- Делают контейнером саму сетку .projects-list. Это допустимо, но надёжнее поместить сетку во **внешнюю** оболочку .projects-area, чтобы паддинги/рамки контейнера не путались с расчётами ширины содержимого.
- Слишком «мелкая» сетка внутри узкого контейнера — следите, чтобы пороги соответствовали реальной минимальной ширине карточки.

Продвинутый приём: контейнерные единицы

Можно «привязать» отступы к ширине контейнера, а не окна:

```
@supports (container-type: inline-size){
  .projects-list{ gap: clamp(12px, 2cqi, 24px); } /* 1cqi = 1% ширины
контейнера */
}
```

Отладка:

- В DevTools (Chrome/Edge/Firefox) у включённого элемента видно активные @container-правила. Убедитесь, что подсвечен именно .projects-area как контейнер.

6) «Плавные» размеры без ступенек — подробно про clamp()

```
.section-title  { font-size: clamp(1.5rem, 1.1rem + 2vw, 2rem); }
.hero__greeting { font-size: clamp(1.25rem, 1rem + 1.2vw, 1.8rem); }
main            { padding-inline: clamp(.75rem, .5rem + 1vw, 2rem); }
```

Как читать clamp(MIN, FLUID, MAX):

- Пока FLUID меньше MIN → берётся **MIN**.
- Когда FLUID в диапазоне MIN..MAX → берётся **сам FLUID** (растёт плавно).
- Когда FLUID превышает MAX → берётся **MAX**.

Откуда берутся «переломы» (точки, где MIN и MAX начинают действовать):

Возьмём font-size: clamp(1.5rem, 1.1rem + 2vw, 2rem) и базовый 1rem=16px:

- MIN = 1.5rem = 24px, MAX = 2rem = 32px, FLUID = 17.6px + 2vw.
- Когда FLUID = MIN: 17.6 + 2vw = 24 → 2vw = 6.4 → vw = 3.2% → **320px** ширина окна.

  До 320px шрифт фиксирован 24px.

- Когда FLUID = MAX: 17.6 + 2vw = 32 → vw = 7.2% → **720px**.

  После 720px шрифт фиксирован 32px.

- Между **320–720px** шрифт **растёт линейно** от 24px до 32px.

Плюс: если пользователь увеличит базовый размер шрифта ОС/браузера, rem-ы увеличатся, а пороги пересчитаются — доступнее.

Когда использовать cqi вместо vw:

Если компонент живёт в колонке, лучше «слушать» контейнер, а не окно:

```
.card__title{ font-size: clamp(1rem, 0.875rem + 2cqi, 1.5rem); }
```

Так шрифт растёт при расширении **колонки**, а не всего экрана (на ультраширокихмониторах заголовки не станут «гигантскими»).

Типичные ошибки:

- Чистые vw без ограничений → текст «раздувается» на 4K. Всегда ставьте MIN/MAX.
- Слишком широкие абзацы. Держите **60–70ch**:

```
.prose{ max-width: 65ch; }
```

- Смешивают em и rem в одной формуле без намерения. rem — от корня (стабильнее), em — от текущего шрифта (может накапливать масштаб в вложенности).
- Ставят vw в отступы центра-колонки и получают «дрожащую» вёрстку. Лучше clamp() с небольшой долей vw или контейнерные cqi.

Проверка себя:

- Измени ширину окна и смотри в DevTools → Computed font-size: должен плавно меняться между расчётными порогами.
- Проверь реальную длину строки — около 60–70 символов (метрика ch).

7) Доступность: формы, контраст, снижение движения

Хелперы:

```
/* Видимый фокус уже задан (:focus-visible) */
.visually-hidden{
  position:absolute!important; width:1px; height:1px; padding:0; margin:-1px;
  overflow:hidden; clip:rect(0 0 0 0); white-space:nowrap; border:0;
}

/* Повышенный контраст (прогрессивно) */
@media (prefers-contrast: more){
  .button, .filter{ box-shadow: inset 0 0 0 2px #fff; }
}

/* Уважение к снижению движения */
@media (prefers-reduced-motion: reduce){
  *{ animation: none !important; transition: none !important; }
}
```

Связка label ↔ input:

```
<label for="email">E-mail</label>
<input id="email" name="email" type="email" aria-describedby="email-hint
email-error" required>
<small id="email-hint" class="visually-hidden">Мы пришлём ответ на этот
адрес</small>
<span id="email-error" class="visually-hidden" role="alert"></span>
```

Замечания:

- Размер целевой области на pointer:coarse — ориентир **44×44 px**.
- Контраст обычного текста ≥ **4.5:1** (WCAG AA).
- color-scheme: light dark; уже подключён.

8) Быстрые вставки прямо в текущий styles.css

```
/* Плавные поля секций */
.section, .main { padding-inline: clamp(1rem, calc(.5rem + 1vw), 2rem); }

/* Кнопки: минимальная зона клика 44px (WCAG) */
.button { padding-block: max(.5rem, calc((44px - 1em)/2)); }

/* Full-bleed для геро-блока (если нужен фон на всю ширину) */
.hero--bleed{
  margin-inline: calc(50% - 50vw);
  padding-inline: calc(50vw - 50%);
}

/* Иконка в поле ввода (если появится поиск) */
.search__input{
  --icon: 20px; --gap: .5rem;
```

```
  background: url(images/search.svg) no-repeat left .75rem center / var(--
icon);
  padding-left: calc(1.5rem + var(--icon) + var(--gap));
}

/* Safe-area в футере (жестовая панель) */
.footer { padding-bottom: calc(1rem + env(safe-area-inset-bottom)); }
```

9) Проверка в DevTools

- **Elements → img → Computed:** Rendered size соответствует sizes.
- **Network (Img):** грузится **один** ресурс из srcset, близкий к slot×DPR.
- **Performance/Web Vitals:** CLS ≈ 0 (если нет — проверьте width/height или aspect-ratio).
- **Lighthouse:** A11y / Best Practices ≥ 90.

10) Самостоятельная работа

Примените те же приёмы к страницам projects.html, diary.html, contacts.html:

- Responsive images у карточек/галерей (srcset + sizes + width/height + loading="lazy").
- Grid-сетки auto-fit + minmax.
- skip-link и :focus-visible на каждой странице.
- Формы: связки <label for> + видимый фокус + сообщения об ошибках.

11) Критерии оценивания

Обязательные:

- корректный viewport;
- семантика + skip-link + видимый фокус;
- герой с srcset/sizes без CLS (желательно AVIF/WebP);
- карточки на Grid;
- sticky-хедер с учётом safe areas.

Бонусы:

- контейнерные запросы (+2);
- поддержка prefers-reduced-motion / prefers-contrast (+1);
- скриншоты Network + Lighthouse (+1).

12) Чек-лист приёмки

- Viewport корректный; на мобильных **нет горизонтального скролла**.
- Семантика: header / nav / main / footer; есть skip-link.
- Видимый :focus-visible; цели клика ≥ 44px на pointer:coarse.
- Герой: width/height + лёгкий srcset/sizes (без CLS), AVIF/WebP по возможности.
- Сетка проектов: Grid repeat(auto-fit, minmax(...)) + gap.
- Sticky-шапка учитывает safe-area; isolation:isolate.
- CQ: проекты «слушают» ширину контейнера (@supports).
- Lighthouse: A11y/Best Practices ≥ 90; CLS ≈ 0 (скрин).

Ссылки

Репозиторий практики (ветка с заданиями):

https://github.com/astafiev-rustam/frontend-and-backend-development/tree/practice-1-15

Дополнительный материал (из методички):

https://habr.com/ru/companies/simpleone/articles/881168/

Приложение 1. calc() — как мыслить и где применять

Арифметика в раскладке: можно складывать/вычитать/умножать/делить **совместимые** величины. Работает в размерах, отступах, шрифтах, позиционировании, Grid/Flex.

Синтаксис и правила (важно):

- Разрешены + - * /. Порядок операций обычный: * и / раньше + и -.
- **Пробелы** вокруг + и - обязательны: calc(1rem + 2vw).
- Умножать/делить можно **только на число**: calc(2rem * 2) ✓, calc(2rem * 2rem) ✗.
- Единицы должны быть совместимы: «длина ± длина», «угол ± угол», % там, где он определён (например, width: calc(100% - 2rem)).
- Вложенность разрешена: calc(1rem + calc(1vw + 2px)).
- **Вертикальные проценты** в padding/margin считаются от **ширины** контейнера (так устроен CSS).
- Скобки повышают читаемость/снимают вопросы с приоритетом: calc((100% - 2rem) / 3).

10 практических рецептов

1. **«Жидкие» поля без MQ**

```
:root { --side: 1rem; }
.container{ padding-inline: calc(var(--side) + 1vw); }

/* ограниченный диапазон */
.container{ padding-inline: clamp(1rem, calc(.75rem + 1vw), 2rem); }
```

2. **Минимальная зона клика 44px (WCAG)**

```
.button{ padding-block: max(.5rem, calc((44px - 1em)/2)); }
```

3. **Full-bleed полоса (без «ловушки 100vw»)**

```
.full-bleed{
  margin-inline: calc(50% - 50vw);
  padding-inline: calc(50vw - 50%);
  background: linear-gradient(135deg, #2563eb, #1d4ed8);
  color:#fff;
}
```

4. **Иконка внутри поля ввода**

```
:root{ --icon: 20px; --gap: .5rem; }
.input{
  background: url(icons/search.svg) no-repeat left center / var(--icon);
  padding-left: calc(var(--icon) + var(--gap));
}
```

5. **Две колонки «контент + фикс-сайдбар» (классика)**

```
.layout{ display: flow-root; }
.main { width: calc(100% - 18rem - 1rem); float: left; }
.aside{ width: 18rem; float: right; margin-left: 1rem; }
/* В проде чаще Grid, но приём полезен. */
```

6. **Grid-колонки с явным «жёлобом», когда gap нельзя**

```
:root{ --gutter: 1rem; }
.two-cols{
  display: grid;
  grid-template-columns:
    calc(50% - var(--gutter)/2)
    calc(50% - var(--gutter)/2);
}
```

7. **Компенсация safe areas**

```
.footer{ padding-bottom: calc(1rem + env(safe-area-inset-bottom)); }
```

8. **Плавная типографика: «идеал» — формула**

```
h1{ font-size: clamp(1.5rem, calc(1rem + 3vw), 2.5rem); }
```

9. **Контейнерные единицы + calc (локальные масштабы)**

```
.widget{ container-type: inline-size; }
.widget__title{
  font-size: clamp(1rem, calc(.5rem + 4cqi), 2rem);
  margin-block: calc(.5rem + 2cqi);
}
```

10. **Смешивание % и px**

```
.box{
  width: calc(100% - 2rem);
  padding: 1rem;
}
```

Частые ошибки и как избежать:

- calc(100%-1rem) → ошибка парсинга (нет пробелов). Пишите calc(100% - 1rem).
- Несовместимые единицы: calc(1s + 1px) ✖, calc(1turn + 90deg) ✔.
- «Магические» числа без смысла — используйте переменные --gutter, --sidebar и комментарии.
- Перегибы с vw — оборачивайте в clamp() или применяйте контейнерные cqi/cqmin.
- Не путайте calc() с задачами min()/max() — это разные инструменты.

Где calc() особенно уместен в учебном проекте:

- Герой/контейнер: padding-inline: clamp(1rem, calc(.5rem + 1vw), 2rem);
- Кнопки/фильтры: зона клика (рецепт №2)
- Межсекционные «полосы» на всю ширину (рецепт №3)
- Формы: иконки в полях (рецепт №4)
- Sticky-footer с safe areas (рецепт №7)

Приложение 2. Мини-демо (полные страницы)

Демо A — Полоса во всю ширину без «ловушки 100vw»

```
<!doctype html>
<html lang="ru">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Full-bleed секция на calc()</title>
<style>
  :root{ --container: 72rem; --pad: 1rem; }
  body{ margin:0; font:16px/1.6 system-ui,sans-serif; }
  .container{ max-width: var(--container); margin:0 auto; padding:0 var(--
pad); }
  .full-bleed{
    margin-inline: calc(50% - 50vw);
    padding-inline: calc(50vw - 50%);
    background: linear-gradient(90deg,#2563eb,#1d4ed8); color:#fff;
  }
  h1{ font-size: clamp(1.5rem, calc(1rem + 2vw), 2.25rem); margin:1rem 0; }
  p{ max-width:65ch; }
</style>
</head>
<body>
  <header class="container"><h1>Центральная колонка</h1></header>
  <section class="full-bleed">
    <div class="container">
      <p>Эта полоса тянется на всю ширину окна благодаря <code>calc(50% -
50vw)</code>, но текст остаётся в привычной колонке.</p>
    </div>
  </section>
  <main class="container"><p>Дальше обычный контент в центре…</p></main>
</body>
</html>
```

Демо B — Кнопка с минимальной высотой 44px (WCAG)

```
<!doctype html>
<html lang="ru">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>44px target на calc()</title>
<style>
  body{ margin:0; font:16px/1.6 system-ui,sans-serif; padding:2rem; }
  .btn{
    display:inline-block; font:inherit; border:1px solid #e2e8f0; border-
radius:.5rem;
    padding-inline: 1rem;
    padding-block: max(.5rem, calc((44px - 1em)/2)); /* ключ */
    background:#2563eb; color:#fff; text-decoration:none;
  }
</style>
</head>
<body>
  <a class="btn" href="#">Нажми меня</a>
</body>
</html>
```

Демо C — Grid-карточки + мягкие поля на calc()

```
<!doctype html>
<html lang="ru">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Grid + calc()</title>
<style>
  body{ margin:0; font:16px/1.6 system-ui,sans-serif; }
  .wrap{
    max-width:72rem; margin:0 auto;
    padding-inline: clamp(1rem, calc(.5rem + 1vw), 2rem);
    padding-block: 1.25rem;
  }
  .grid{
    display:grid;
    grid-template-columns: repeat(auto-fit, minmax(18rem, 1fr));
    gap: 1rem;
  }
  .card{ padding:1rem; border:1px solid #e5e7eb; border-radius:.75rem;
background:#fff; }
</style>
</head>
<body>
  <main class="wrap">
    <section class="grid">
      <article class="card">A</article>
      <article class="card">B</article>
      <article class="card">C</article>
      <article class="card">D</article>
    </section>
  </main>
</body>
</html>
```