Адаптивная вёрстка

План урока

1. Введение
2. Новые единицы измерения vh, vw
3. Адаптивные единицы для текста em, rem
4. Новые адаптивные возможности flexbox
5. Новые адаптивные возможности grid layout

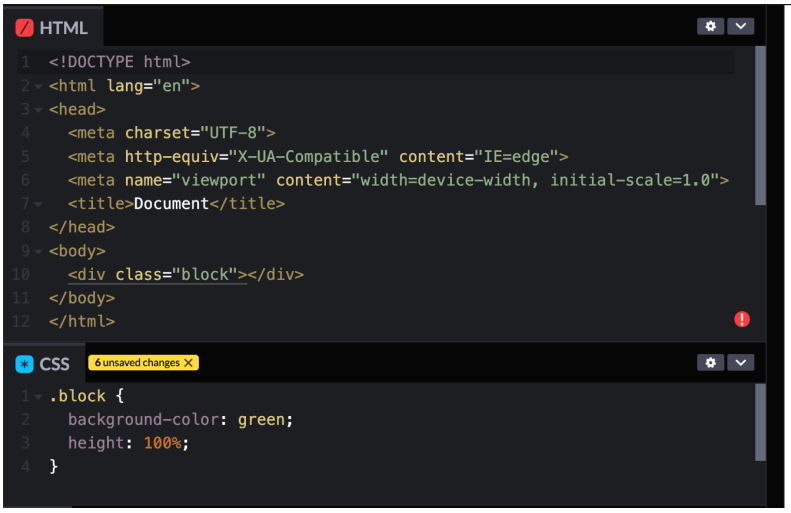
Введение

Мы уже с вами знаем основы создания адаптивного сайта, которых достаточно чтобы создать большинство адаптивных сайтов, но бывают такие задачи, которые решить бывает очень сложно и тут нам не хватает более профессионального подхода к решению задач. В данном уроке мы сможем разобрать новшества, которые появились в CSS3 и которые позволяют упростить создание адаптивных сайтов, а в некоторых моментах даже автоматизировать этот переход.

Новые единицы измерения vh, vw

Казалось бы а зачем нам еще одна единица измерения, с которой вы возможно уже знакомы, всё дело в том, что ее начинают использовать всё чаще и понимать основной механизм работы, плюс где правильно будет применить данную единицу, наша основная задача.

Первое с чего мы начнем, это конечно же с проблемы, которую решают данные единицы измерения. Для этого давайте рассмотрим пример



Самый простой пример, есть block для которого заданы значения цвета фона и высота, равная 100%. Но в правой части мы видим просто белую область, хотя изначально предполагаем что она должна быть зеленого цвета. Почему же блок не стал 100% высоты? Ответ на этот вопрос достаточно простой, мы ведь не знаем 100% от чего, получается что браузер не знает от чего считать высоту и делает ее 0, так как 100% от 0 это и будет 0.

Наша задача, задать высоту именно видимой области сайта, значит значение должно зависеть от высоты экрана. Именно за эту часть и отвечают новые параметры высоты и ширины, они привязаны именно к ширине или высоте экрана.

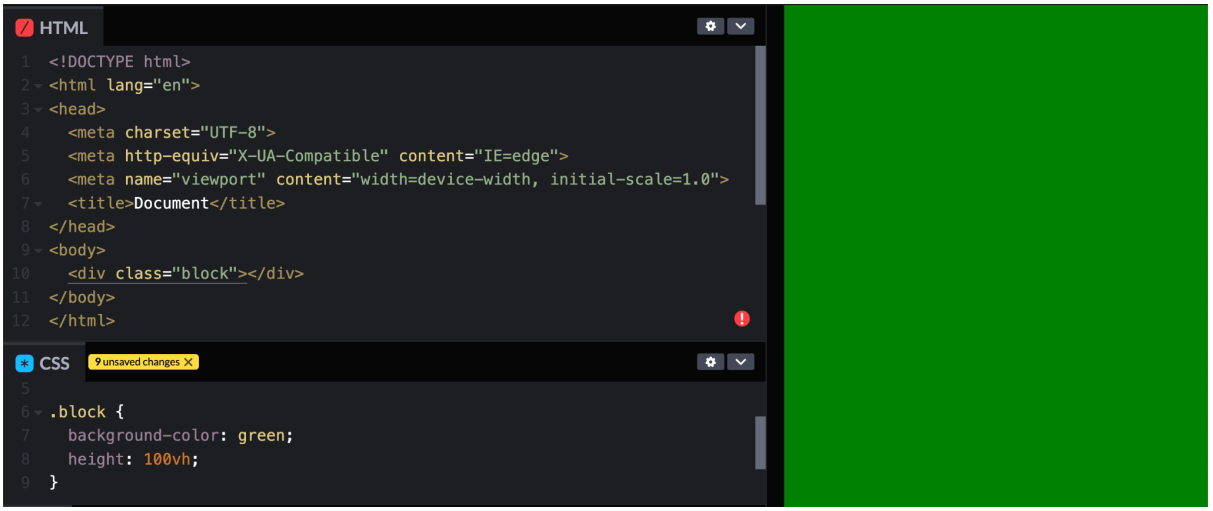
Подведем итог: приемы отзывчивого веб-дизайна базируются на использовании процентных значений. Но проценты далеко не лучшее решение для каждого случая, так как они вычисляются относительно размеров ближайшего родительского элемента. Поэтому, если вы хотите использовать высоту и ширину окна браузера, лучше воспользоваться единицами vh и vw. Например, если высота окна браузера равна 900px, то 1vh будет равен 9px. Аналогично, если ширина окна браузера равна 1600px, то 1vw будет равен 16px.

В каких ситуациях нужно использовать vh, vw

Теперь давайте подумаем, стоит ли полностью отказаться от использования процентов и всегда использовать новые единицы измерения, конечно же не стоит. Давайте для примера возьмём десктопную версию. Первое что нужно отметить, это то что значения ширины и высоты вообще не очень частые явления, тем более если мы говорим про значение высоты, мы его задаём крайне редко, лучше всего в такой ситуации использовать отступы. Поэтому если вам нужно чтобы блок занимал половину ширины контентной части, то заменить значение 50% на 50vw будет совсем неверным, ведь нас интересует именно ширина контента, а не ширина экрана. Во время широкоформатных мониторов вообще использовать vw для десктопной версии будет ошибкой.

Аналогично и для изображений, Так как ширина элемента, указанная с помощью 100vw больше ширины области просмотра, то для создания полноэкранных фоновых изображений лучше использовать ширину 100%, которая будет равна ширине корневого элемента html.

Если мы говорим про единицу измерения vh, то тут есть один очень часто используемый приём, вам необходимо сделать блок, высотой с экран пользователя, это чаще всего используется для первых блоков, которые занимают как раз всё доступное пространство.



В итоге заменив проценты на vh мы получаем тот самый результат, который ждали изначально, плюс нам неважно какое значение высоты экрана будет у пользователя, он всегда будет занимать именно высоту экрана, ни больше, ни меньше.

Закономерный вопрос, а где же так популярны vw, vh? Конечно же начиная с планшетной версии, мы не используем центрирование контента, нам чаще всего нужно использовать именно значение ширины экрана. Вот тут я настоятельно рекомендую использовать данные единицы измерения.

Что мы можем заметить в данном примере, мы идеально распределили 6 элементов и нам абсолютно не важно какие параметры ширины у планшета или мобильного телефона.

Адаптивные единицы измерения для текста

Ну что как обычно предлагаю начать с проблемы, которую мы можем заметить, при создании адаптивного сайта, большая часть контента, который есть у нас на странице, это текст. У нас

есть заголовки, списки, параграфы и все эти элементы должны быть хорошо читабельны, поэтому дизайнер, при создании макета, задаёт параметры размера шрифта. Данный шрифт, для планшетной версии, будет смотреться достаточно объемно, а на мобильной версии вообще может потерять свою читабельно, так как в заголовках размер слова может не помещаться в размер экрана.

Конечно мы знаем про медиа-запросы и они полностью решаю данный вопрос. Давайте рассмотрим простой пример

HTML

```
<h1 class="title">Lorem, ipsum dolor.</h1>

<h2 class="heading">Lorem ipsum dolor sit.</h2>

<p class="text">Lorem ipsum dolor sit amet consectetur adipisicing elit. Et, placeat quod. Deserunt hic fugit doloribus

    necessitatibus quisquam, voluptates placeat tempore officiis, pariatur odio incidunt. Ullam ducimus aut quo enim

  molestias!</p>
```

CSS

```
.title {
  font-size: 32px;
}

.heading {
  font-size: 24px;
}

.text {
  font-size: 16px;
}
```

Казалось бы простой пример, есть 3 текстовых элемента и стили для них, причем только размеры шрифта, чтобы остальные стили нас не отвлекали. Но самое интересное начинается, когда мы зададим параметры для планшетной и мобильной версии проекта.

```
@media (max-width: 1024px) {
  .title {
```

```
        font-size: 28px;
    }

    .heading {
        font-size: 21px;
    }

    .text {
        font-size: 14px;
    }
}

@media (max-width: 1024px) {
    .title {
        font-size: 24px;
    }

    .heading {
        font-size: 18px;
    }

    .text {
        font-size: 12px;
    }
}
```

Как мы можем заметить, нам пришлось продублировать все элементы и задать для них новые параметры размера шрифта. А теперь давайте представим что у нас 50 текстовых элементов, это ведь не много практически для любого сайта, вот тут мы получим дублирование стилей и лишние 100 строчек кода как минимум.

Именно такую проблему решает единица измерения **rem** мы изначально можем задать параметры размера шрифта в новых единицах измерения, который просто будет являться коэффициентом.

Давайте рассмотрим на примере заголовка h1, его размер шрифта для десктопной версии равен 32 пикселя, а для планшета уже 28, в мобильной версии размер шрифта составляет 24.

Нам необходимо число 32 представить как коэффициент умножить на значение

Важно: Вы можете придумать любой коэффициент, на ваше усмотрение, но желательно его выбрать кратным 4.

Одним из популярных значений является 8 или 16, для нашего примера вы возьмём число 16. Получается, что значение 32, мы можем представить как 2 * 16, где 16 как раз будет значение **1rem**

Сначала кажется запутанным, но чем больше ты начинаешь работать с данной единицей измерения, тем проще получается еде высчитывать.

В итоге, мы с вами представили что 1rem == 16px; Получается что размер шрифта заголовка будет равен 2rem, так как 16px * 2 == 32px

Данное значение rem необходимо задать для тега html

```
html {
    font-size: 16px;
}
```

Давайте перепишем все остальные значения, для нашего примера

```
.title {
    font-size: 2rem;
}

.heading {
    font-size: 1.5rem;
}

.text {
    font-size: 1rem;
}
```

Напоминаю, что вам просто исходное значение, необходимо разделить на 16, где 16 вы выбрали самостоятельно, допустим 24 / 16 == 1.5

Но что мы выигрываем, если задали такие единицы измерения? Вся особенность, заключается в том, что мы просто переопределим значение 1rem для меньших разрешений экрана и оно автоматически заменится у всех остальных значений

```
@media (max-width: 1024px) {
    html {
        font-size: 14px;
    }
}
@media (max-width: 1024px) {
    html {
        font-size: 12px;
    }
}
```

Получается, что для планшетной версии значение 1rem составляет 14px, а для мобильной 1rem == 12px

Какой вывод мы можем сделать, что теперь нам абсолютно не важно, сколько текстовых элементов мы можем добавить на сайт, их может быть 50 или больше, нам для каждого их них в медиа запросе не придётся задавать новые параметры размера шрифта.

Ложка дёгтя

Если новые единицы измерения для текста такие удобные, почему же мы продолжаем использовать пиксели или почему все сайты не перешли на использования rem? всё дело в том, что мы задаем коэффициент, который в пропорциях меняет размер шрифта для новых разрешений экрана, но как быть, если мы хотим оставить размер шрифта у текста в 16px и не менять его для мобильной и планшетной версии, получается что тут необходимо оставить px. Поэтому мы не должны использовать rem для всех проектов, мы должны их применять в ситуации, если их предусмотрел дизайне.

Новые адаптивные возможности flexbox

Мы уже очень хорошо знаем свойства flexbox и можем применять их на практике, но есть несколько свойств и значений, которые помогают при создании адаптивного сайта достаточно редко разбираются и самое главное, не совсем понятно где их можно применять, давайте с этими свойствами познакомимся поближе.

flex-grow

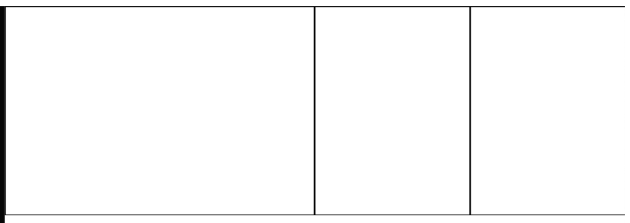
Свойство определяет коэффициент роста одного flex-элемента относительно других flex-элементов в flex-контейнере при распределении положительного свободного пространства. Если сумма значений flex-grow flex-элементов в строке меньше 1, они занимают менее 100% свободного пространства.

Простыми словами свойство flex-grow определяет как много свободного пространства во flex-контейнере должно быть назначено текущему элементу. Свободное пространство — разница между размером flex-контейнера и размером всех его flex-элементов вместе. Если все sibling-элементы (sibling items — элементы одного уровня вложенности, состоящие по отношению друг к другу в родственной связи как брат или сестра), имеют одинаковый коэффициент flex-grow, то все они получат одинаковую долю свободного пространства, в противном случае оно распределяется в соответствии с соотношением, определённым различными коэффициентами flex-grow.

Что мы можем увидеть в данном примере? Для того чтобы блок разбить на N равное количество частей, нам необходимо дочерним элементам задать flex-grow: 1; где 1 эт коэффициент соотношения одного элемента к другому и если нам потребуется чтобы первый блок был в 2 раза больше, чем остальные, мы можем ему задать flex-grow: 2;

```html
<div class="content">
  <div class="item item_1"></div>
  <div class="item item_2"></div>
  <div class="item item_3"></div>
</div>
```

```css
.item {
  border: 1px solid #000;
  flex-grow: 1;
  height: 200px;
}

.item_1 {
  flex-grow: 2;
}
```



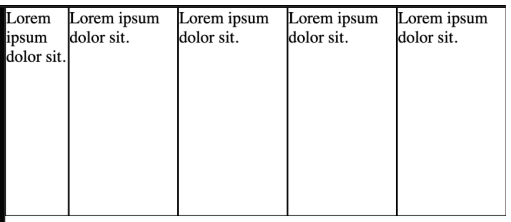
На практике flex-grow используется вместе с другими flex-свойствами flex-shrink и flex-basis, и обычно определяется с помощью сокращения (shorthand) flex, чтобы убедиться, что все значения заданы.

flex-shrink

Свойство указывает коэффициент сжатия flex-элемента относительно других flex-элементов при распределении отрицательного свободного пространства. Умножается на базовый размер flex-basis. Отрицательное пространство распределяется пропорционально тому, насколько элемент может сжаться, поэтому, например, маленький flex-элемент не уменьшится до нуля, пока не будет заметно уменьшен flex-элемент большего размера.

```html
<div class="content">
  <div class="item item_1">Lorem ipsum dolor sit.</div>
  <div class="item item_2">Lorem ipsum dolor sit.</div>
  <div class="item item_3">Lorem ipsum dolor sit.</div>
  <div class="item item_4">Lorem ipsum dolor sit.</div>
  <div class="item item_5">Lorem ipsum dolor sit.</div>
</div>
```

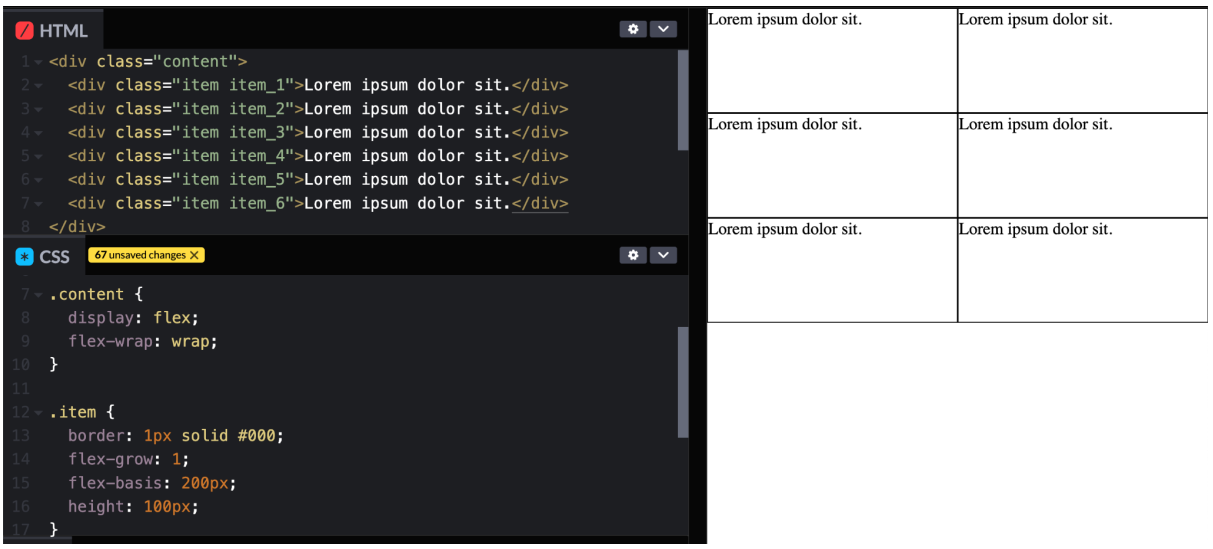
```css
.content {
  display: flex;
}

.item {
  border: 1px solid #000;
  height: 200px;
}

.item_1 {
  flex-shrink: 2;
}
```



Что мы сразу можем заметить, то что первый блок стал значительно меньше, чем остальные, всё дело в том что его коэффициент сжатия в 2 раза больше чем у остальных.

flex-basis

Свойство устанавливает начальный основной размер flex-элемента до распределения свободного пространства в соответствии с коэффициентами гибкости. Для всех значений, кроме auto и content, базовый гибкий размер определяется так же, как width в горизонтальных режимах записи. Процентные значения определяются относительно размера flex-контейнера, а если размер не задан, используемым значением для flex-basis являются размеры его содержимого.



Данный пример очень хорошо подходит для того, чтобы понять как работает flex-basis, если мы элементу задаём значение flex-grow: 1; то он все блоки в одной линиии делает одинаковой ширины, но у нас чаще всего требуется чтобы блоки были выстроены в несколько колонок, поэтому на нужно сообщий браузеру минимальное значение ширины элемента, именно эту задачу решает flex-basis

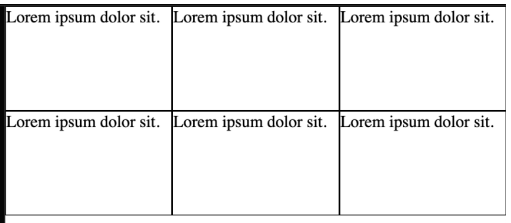
Новые адаптивные возможности grid layout

Мы уже знакомы с grid layout, но частой проблемой данной технологии, является то что мы задаём фиксированные значения строк, столбцов и колонок, получается идеальная сетка, но без адаптивных возможностей, поэтому давайте разбираться с адаптивными возможностями grid layout.

Для начала давайте построим простую сетку, а в дальнейшем зададим ей адаптивные возможности

```html
<div class="content">
  <div class="item item_1">Lorem ipsum dolor sit.</div>
  <div class="item item_2">Lorem ipsum dolor sit.</div>
  <div class="item item_3">Lorem ipsum dolor sit.</div>
  <div class="item item_4">Lorem ipsum dolor sit.</div>
  <div class="item item_5">Lorem ipsum dolor sit.</div>
  <div class="item item_6">Lorem ipsum dolor sit.</div>
</div>
```

```css
.content {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}

.item {
  border: 1px solid #000;
  height: 100px;
}
```

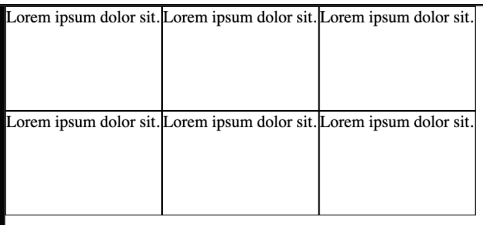


Какой очевидный минус в данном примере, это значение repeat(3, 1fr) а если быть точнее это значение 3, ведь так мы для любого разрешения экрана выставили данные параметры. Конечно мы можем менять эти значения для мобильной и планшетной версии в медиа запросе, но это точно приведет к ошибке на каком-то разрешении экрана.

Именно для таких ситуаций нам и необходимо значение auto-fit или auto-fill. Используя значение auto-fill, вы всегда получите хотя бы один столбец, даже если он по какой-то причине не помещается в контейнер-сетку. Если вы используете auto-fit, то дорожки, которые не содержат элементы сетки, будут сброшены.

```html
<div class="content">
  <div class="item item_1">Lorem ipsum dolor sit.</div>
  <div class="item item_2">Lorem ipsum dolor sit.</div>
  <div class="item item_3">Lorem ipsum dolor sit.</div>
  <div class="item item_4">Lorem ipsum dolor sit.</div>
  <div class="item item_5">Lorem ipsum dolor sit.</div>
  <div class="item item_6">Lorem ipsum dolor sit.</div>
</div>
```

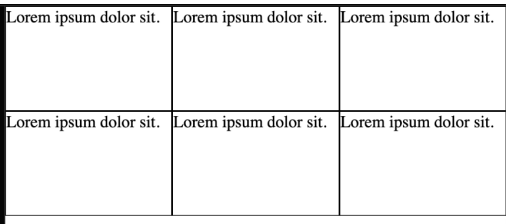
```css
.content {
  display: grid;
  grid-template-columns: repeat(auto-fill, 150px);
}

.item {
  border: 1px solid #000;
  height: 100px;
}
```



Используем один из приёмов, но теперь мы видим что мы потеряли гибкость блоков, они умеют автоматически перестраиваться на новую строку, но не могут занимать все доступное пространство, так как мы задали фиксированное значение 150px, как же решить этот вопрос

Соответствие содержимому

Размеры дорожек можно задавать с помощью значения fit-content(длина или %), представляющее собой формулу min(maximum size, max(minimum size, argument)), которая вычисляется как minmax(auto, max-content), то есть auto. При этом, размер дорожки ограничивается значением, указанным в скобках, и если оно больше, чем автоматический минимум.



Применив новые значения мы получаем полностью адаптивную сетку, которая для большего и для меньшего разрешения экрана будет выглядеть отлично

Источники информации

https://html5book.ru/