Работа с макетом в редакторе графики, определение параметров блоков и текста

Изучение верстки — это не только знание способов стилизации элементов, но и верстка макетов. Чтобы начать верстать макеты, их нужно как-то увидеть и понять, как с ними работать.

На рынке существует несколько основных редакторов. Некоторые из них специфичны только для одной операционной системы, другие могут работать в браузере, а некоторые имеют возможность установки на любую систему. Вот только некоторые из них:

1. Adobe Photoshop. Самый старый редактор, который был стандартом много лет. Сейчас же отступает на второй план ввиду своей сложности в освоении. Перегрузка различными функциями также не красит его — многие функции просто не нужны для верстальщика, но память устройства они благополучно съедают. Редактор доступен на системах под управлением Microsoft Windows и macOS. Официальной версии под систему Linux сейчас нет.
2. Gimp. Аналог Photoshop для систем Linux. Имеет схожий функционал и может корректно открывать некоторые файлы в формате Photoshop. Но не стоит надеяться, что он может стать полноценной заменой. Если дизайнер присылает исходники в Photoshop, то нет полной уверенности в корректном открытии файла через Gimp. Так же, как Photoshop, Gimp не является специализированным инструментом для веб-дизайнеров, поэтому многие функции не нужны.
3. Sketch. Специализированный редактор для проектирования мобильных и веб интерфейсов. Это является большим плюсом, так как разработчики ориентируются именно на удобное проектирование интерфейсов. Для верстальщиков здесь множество плюсов: удобный просмотр макета, размеров, отступов и другой информации, которая поможет при переносе макета. Главным минусом редактора является поддержка одной операционной системы — macOS. Разработчики ориентированы на работу только с этой системой, и переноса редактора на другие ОС не ожидается.

4. Figma. Редактор, который работает в браузере. Просмотр макетов в нем доступен с любого устройства, которое имеет браузер и выход в интернет. Если ваша кофемолка умеет это, то можно верстать и с ее помощью. Редактор, как и Sketch, ориентирован на создание интерфейсов, что положительно сказывается на удобстве работы с ним. Именно этот редактор используется для проектов и некоторых испытаний, где вам дается макет для верстки. Также является бесплатным для личного пользования.

Это неполный набор редакторов, которые существуют на рынке. Возможно, на работе вы встретите и другие варианты, но не переживайте. Нет нужды учить каждый редактор в отдельности. Их интерфейсы и способы взаимодействия часто совпадают. Если вы научитесь работать с одним редактором, то разберетесь и в другом.

В этом уроке мы рассмотрим базовые действия при работе с редактором Figma.

Установка

Как и было сказано выше, Figma не требует установки. Ее работа осуществляется через браузер. Достаточно перейти на сайт figma.com и зарегистрироваться.

Помимо этого есть возможность поставить отдельное приложение. Это скорее вопрос удобства, так как вместо вкладки вы получите отдельное окно для работы с редактором. Сейчас приложение доступно для операционных систем Windows и macOS. Скачать их вы можете на странице загрузки Figma.

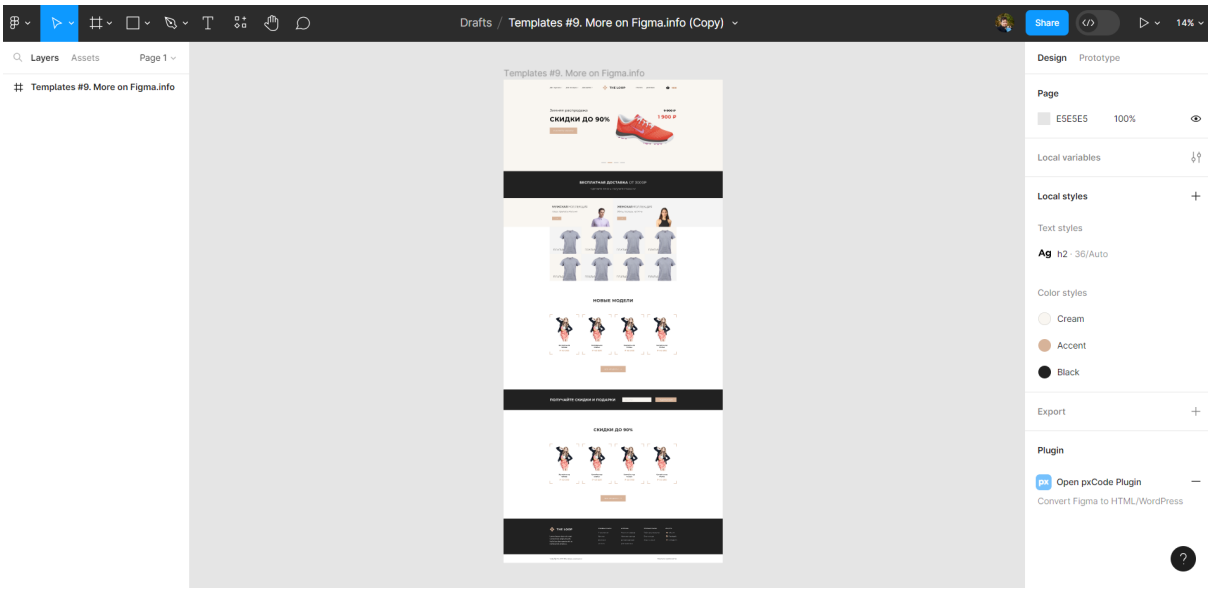
Для систем *Linux* есть специальные варианты установки под разные типы систем. Их можно найти на GitHub.

Используйте тот вариант, который вам удобен.

Работа с макетом

В этом разделе посмотрим на основные действия при работе верстальщика с редактором. В качестве примера будет использован один из учебных шаблонов нашего курса Шаблон.

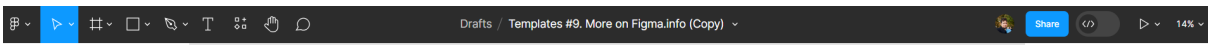
1. Зарегистрируйтесь в сервисе Figma. Это позволит вам удобно просматривать элементы макета.
2. Откройте шаблон.



Окно работы с Figma можно разбить на 4 функциональные области:

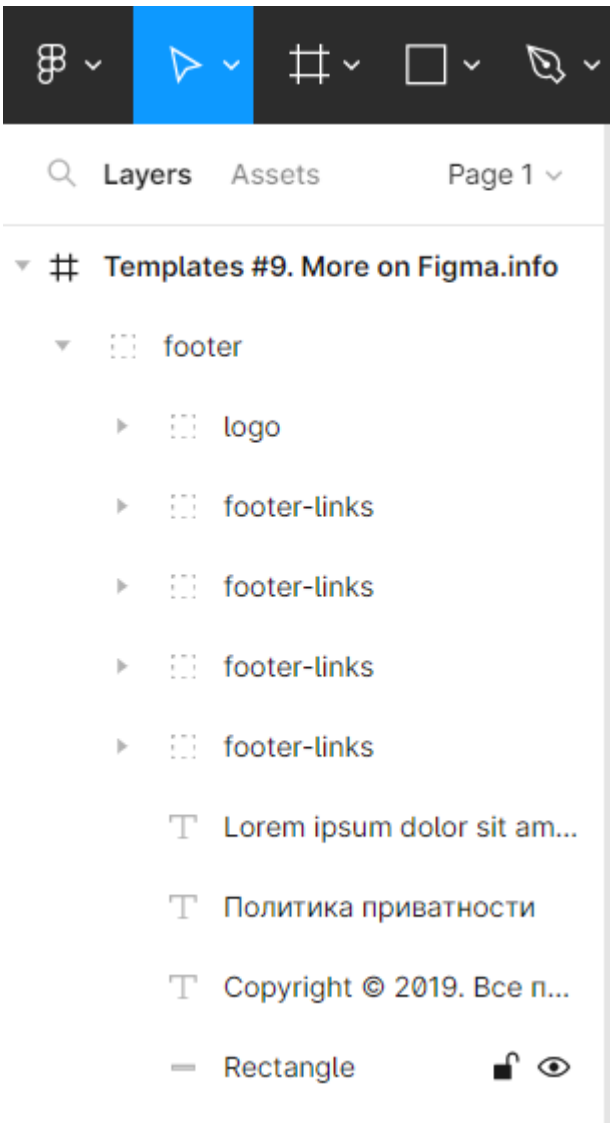
Шапка

В ней находятся различные настройки, название файла, настройки публичных ссылок. Если вы создадите свой макет, то в шапке будут располагаться и инструменты для создания элементов.



Слои

В левой области находятся все элементы, которые добавлены в макете. Это может быть текст, геометрические фигуры, изображения. Такие элементы называются слоями. Это связано с тем, что редактор учитывает порядок элементов. Если поставить изображение выше текста в слоях, то оно и в макете наложится на текст:

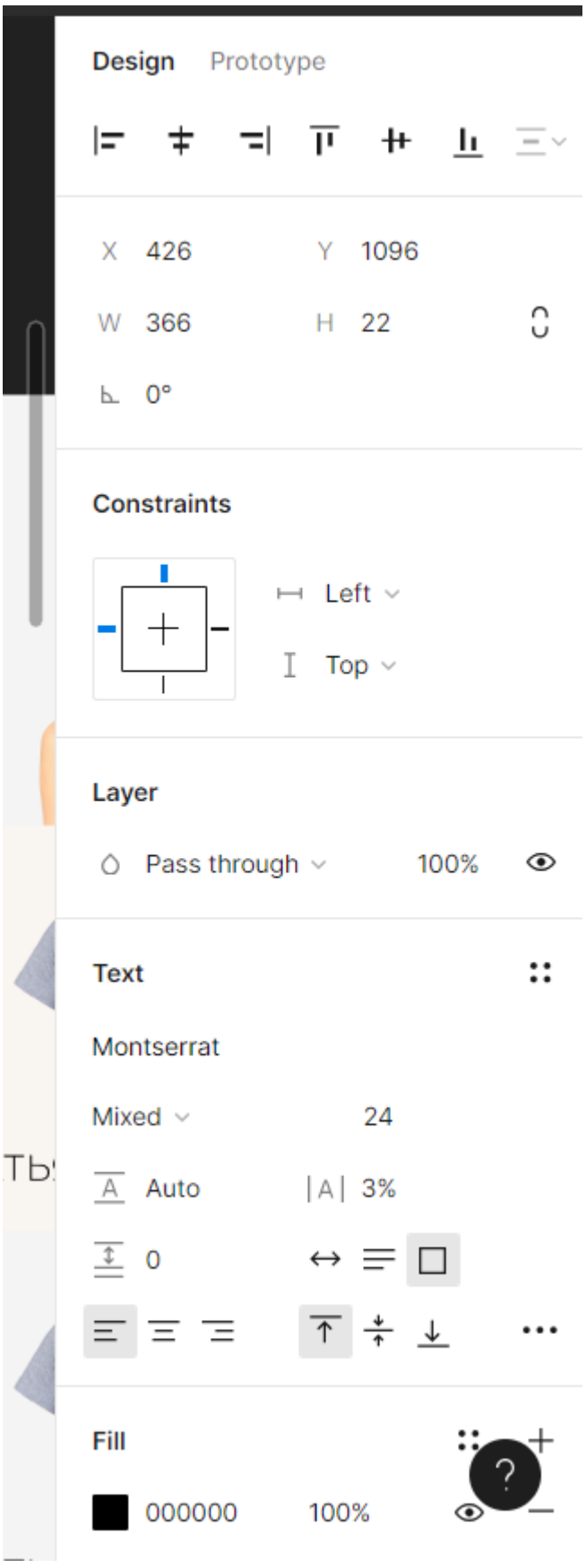


Слои можно объединять в различные группы. Так удобно разделять слои различных частей макета. В макете нашего шаблона есть следующие группы:

- first-screen
- footer
- subscribe
- и т.д.

Design

Самая полезная секция для верстальщика. Инспектор позволяет увидеть все настройки выбранного элемента в секции «Слои».

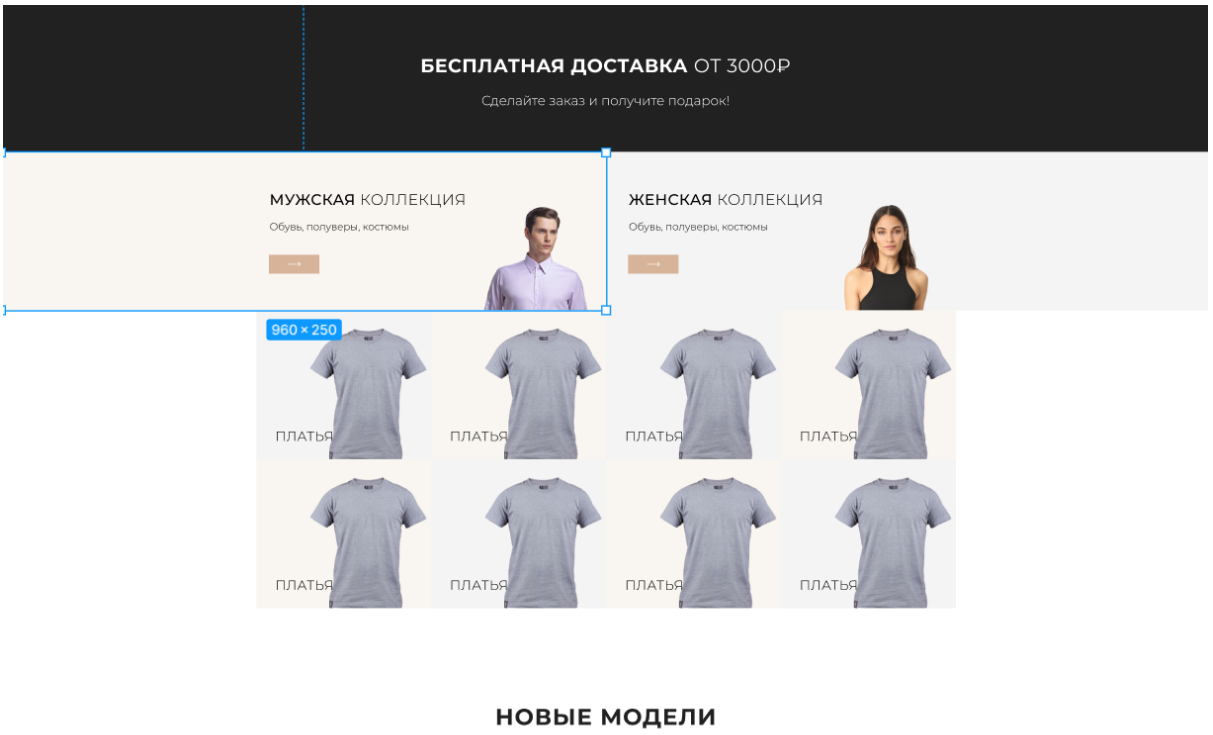

Design
Prototype
X 426
Y 1096
W 366
H 22
0°
Constraints
Left
Top
Layer
Pass through
100%
Text
Montserrat
Mixed
24
Auto
3%
0
Fill
000000
100%

В ней верстальщик может уточнить к какому семейству принадлежит шрифт, размер шрифта, написание и т.д.

Окно просмотра макета

Центральная область содержит визуальное представление макета. В ней можно выбирать различные слои с помощью мыши, а также посмотреть отступы элементов друг от друга.

Если выбрать слой «Узнать больше», зажать клавишу Alt или cmd для macOS и навести на соседний элемент, то Figma автоматически подскажет расстояние между элементами.



По этому изображению можно сразу узнать несколько параметров элемента:

1. Размер элемента.
2. Расстояние от текста до элемента.

2.Форматы графики, экспорт и оптимизация изображений

Хороший верстальщик должен уметь правильно выбирать форматы изображений для своей вёрстки, чтобы изображения отображались без погрешностей и имели оптимальный размер при загрузке. Давайте разберёмся, какие бывают форматы изображений и в каких ситуациях лучше выбрать тот или иной формат.

Растровые форматы

Для начала рассмотрим форматы, которые относятся к растровой графике: GIF, JPEG, PNG и WebP.

Основные характеристики, которые нас будут интересовать при выборе формата — это качество изображения, вес и количество цветов. В вебе тяжёлые изображения непрактичны, поскольку они долго загружаются. Чтобы уменьшить вес файла, используются алгоритмы сжатия. Сжатие может быть с потерями и без потерь. При выборе подходящего формата изображения, нам нужно найти баланс между весом файла и качеством картинки, так как некоторые алгоритмы сжимают изображения с потерей качества. Теперь рассмотрим каждый из форматов подробнее.

GIF (Graphics Interchange Format)

Формат был разработан компанией CompuServe в далёком 1987 для передачи растровых изображений по интернету. GIF имеет цветовую палитру, состоящую из 256 цветов. Алгоритм GIF выбирает 256 наиболее используемых в исходном изображении цветов, а все остальные оттенки создаются путём подмешивания — подбора соседних пикселей таким образом, чтобы человеческий глаз воспринимал их как нужный цвет. По

этой причине GIF не подходит для хранения полноцветных изображений и фотографий.

Формат поддерживает прозрачность — каждый пиксель изображения может быть в двух состояниях: прозрачный или непрозрачный, полупрозрачность не поддерживается.

Особенностью GIF является поддержка анимации, то есть этот формат может хранить несколько кадров, которые сменяют друг друга с определённой частотой.

Таким образом, формат GIF подходит если:

- изображение не многоцветное;
- нужна простейшая прозрачность;
- нужна анимация.

JPEG (Joint Photographic Experts Group)

Формат JPEG получил своё название от объединённого комитета экспертов по фотографии, который и создал этот стандарт в конце 80-х — начале 90-х годов. Он был разработан для сжатия и хранения полноцветных фотографий. Поддерживает более 16 миллионов цветов.

Формат JPEG сжимает изображения с потерей качества. Алгоритм сжатия основан на разбиении исходного изображения на квадраты 8×8 пикселей, и последующей их группировке. Можно получать JPEG изображения очень маленького веса, но только за счёт ухудшения качества картинки, можно

получить и очень качественные JPEG, но тогда картинка будет слишком тяжёлой. Поэтому главная задача при работе с JPEG — подобрать такой уровень качества, чтобы вес был небольшой и качество картинки было приемлемым (обычно, это диапазон от 60 до 70, но нужно тестировать на каждой картинке).

Таким образом, формат JPEG лучше подходит для:

- полноцветных изображений, фотографий;
- изображений, с плавным переходом яркости и контраста;
- рисунков с большим количеством разноцветных деталей.

PNG (Portable Network Graphics)

PNG является относительно недавним форматом, который был введён как альтернатива для GIF-файлов.

PNG является форматом сжатия без потерь и позволяет сохранять изображения, в которых требуется особенная чёткость. Например, чертежи и печатный текст.

Формат имеет две вариации: PNG8 и PNG24. PNG8 может хранить лишь 256 цветов, а PNG24 использует уже более 16 миллионов цветов.

Главная особенность формата PNG — поддержка альфа-прозрачности, то есть каждому пикселю в отдельности можно задать свою степень прозрачности.

Итак, формат PNG подходит для:

- изображений с прозрачностью и полупрозрачностью;
- когда необходима повышенная точность полноцветных изображений;
- изображений с резкими переходами цветов.

WebP

WebP — новый формат, созданный и развиваемый с 2010 года компанией Google.

Главная цель этого проекта — ещё больше уменьшить вес при сохранении такого же качества.

Формат использует новый алгоритм сжатия, в котором искажения отличаются от искажений других форматов. Ухудшается детализация и структура, в то время как края остаются чёткими.

Особенности WebP:

- сжимает изображения без потерь лучше, чем PNG (на 26% по данным Google);
- сжимает изображения с потерями лучше, чем JPEG (на 25–34% по данным Google);
- поддерживает прозрачность (альфа-канал).

Иногда WebP сжимает изображение даже лучше, чем заявляет Google.

Ввиду относительной новизны формата, не все браузеры умеют с ним работать. На сегодняшний день WebP поддерживается только Chrome, Opera и Firefox.

Векторные форматы

GIF, JPEG, PNG, и WebP — растровые форматы, основанные на дискретном (пиксельном, точечном) представлении изображения, в то время как векторные форматы основаны на математических формулах (геометрическом представлении фигур).

SVG (Scalable Vector Graphics)

SVG переводится как — масштабируемая векторная графика. Формат существует с 1999 года.

Размер объектов SVG намного меньше размера растровых изображений, а сами изображения не теряют в качестве при масштабировании. В отличие от растровых форматов мы можем взаимодействовать с изображениями в формате SVG — при помощи CSS можно изменять параметры графики: цвет, прозрачность или границы, а при помощи JavaScript — анимировать изображение.

SVG поддерживается почти всеми браузерами за исключением Internet Explorer 8 и ниже, но и это можно решить подключением JavaScript-библиотек, например, SVGeezy.

Формат SVG отлично подходит для малоцветных схем, логотипов и иконок.

Таким образом, формат SVG подходит если:

- нужно анимировать части изображения;
- изменять цвет элементов изображения;
- необходимо масштабировать изображение без потерь.

Создание файловой структуры проекта, относительные пути на примере графики и стилевых файлов.

Веб-сайт состоит из множества файлов: текстового контента, кода, стилей, медиа-контента, и так далее. Когда вы создаёте веб-сайт, вы должны собрать эти файлы в рациональную структуру на вашем локальном компьютере, убедитесь, что они могут общаться друг с другом, и весь ваш контент выглядит правильно, прежде чем вы, в конечном итоге загрузите их на сервер. В этом разделе рассмотрены некоторые вопросы, о которых вам следует знать, чтобы вы могли рационально настроить файловую структуру для своего веб-сайта.

Когда вы работаете на веб-сайте локально на вашем компьютере, вы должны держать все связанные файлы в одной папке, которая отражает файловую структуру опубликованного веб-сайта на сервере. Эта папка может располагаться где угодно, но вы должны положить её туда, где вы сможете легко её найти, может быть, на ваш рабочий стол, в домашнюю папку или в корень вашего жёсткого диска.

1. Выберите место для хранения проектов веб-сайта. Здесь, создайте новую папку с именем web-projects (или аналогичной). Это то место, где будут располагаться все ваши проекты сайтов.
2. Внутри этой первой папки, создайте другую папку для хранения вашего первого веб-сайта. Назовите её test-site (или как-то более творчески)

Вы заметите, что в этом разделе, вас просят называть папки и файлы полностью в нижнем регистре без пробелов. Это потому что:

1. Многие компьютеры, в частности веб-серверы, чувствительны к регистру. Так, например, если вы положили изображение на свой веб-сайт в test-site/MyImage.jpg, а затем в другом файле вы пытаетесь вызвать изображение как test-site/myimage.jpg, это может не сработать.
2. Браузеры, веб-серверы и языки программирования не обрабатывают пробелы последовательно. Например, если вы используете пробелы в имени файла, некоторые системы могут отнестись к имени файла как к двум именам файлов. Некоторые серверы заменяют пробелы в вашем имени файла на "%20" (символьный код для пробелов в URI), в результате чего все ваши ссылки будут сломаны. Лучше разделять слова дефисами, чем нижними подчёркиваниями: my-file.html лучше чем my_file.html.

Говоря простым языком, вы должны использовать дефис для имён файлов. Поисковая система Google рассматривает дефис как разделитель слов, но не относится к подчёркиванию таким образом. По этим причинам, лучше всего приобрести привычку писать названия ваших папок и файлов в нижнем регистре без пробелов, разделяя слова дефисами, по крайней мере,

пока вы не поймёте, что вы делаете. Так в будущем вы столкнетесь с меньшим количеством проблем.

Далее, давайте взглянем на то, какую структуру должен иметь наш тестовый сайт. Наиболее распространённые вещи, присутствующие в любом проекте сайта, которые мы создаём: индексный файл HTML и папки, содержащие изображения, файлы стилей и файлы скриптов. Давайте создадим их сейчас:

1. index.html: Этот файл обычно содержит контент домашней страницы, то есть текст и изображения, которые люди видят, когда они впервые попадают на ваш сайт. Используя ваш текстовый редактор, создайте новый файл с именем index.html и сохраните его прямо внутри вашей папки test-site.
2. Папка images: Эта папка будет содержать все изображения, которые вы используете на вашем сайте. Создайте папку с именем images внутри вашей папки test-site.
3. Папка styles: Эта папка будет содержать CSS код, используемый для стилизации вашего контента (например, настройка текста и цвета фона). Создайте папку с именем styles внутри вашей папки test-site.
4. Папка scripts: Эта папка будет содержать весь JavaScript-код, используемый для добавления интерактивных функций на вашем сайте (например, кнопки которые загружают данные при клике). Создайте папку с именем scripts внутри вашей папки test-site.

Для того, чтобы файлы общались друг с другом, вы должны указать файлам путь друг к другу - обычно один файл знает, где находится другой. Чтобы продемонстрировать это, мы вставим немного HTML в наш файл index.html и научим его отображать изображение.

1. Скопируйте изображение, которое вы выбрали ранее, в папку images.
2. Откройте ваш файл index.html и вставьте следующий код в файл именно в таком виде.

```
<!doctype html>

<html>

  <head>

    <meta charset="utf-8" />

    <title>Моя тестовая страница</title>

  </head>

  <body>

    <img src="" alt="Моё тестовое изображение" />

  </body>

</html>
```

1. Строка <img src="" alt="Моё тестовое изображение"> - это HTML код, который вставляет изображение на страницу. Мы должны сказать HTML, где находится изображение. Изображение находится

внутри папки *images*, которая находится в той же директории что и index.html. Чтобы спуститься вниз по нашей файловой структуре от index.html до нашего изображения, путь к файлу должен выглядеть так images/your-image-filename. Например наше изображение, названное firefox-icon.png, имеет такой путь к файлу: images/firefox-icon.png.

2. Вставьте путь к файлу в ваш HTML код между двойными кавычками src="".
3. Сохраните ваш HTML файл, а затем загрузите его в вашем браузере (двойной щелчок по файлу). Вы должны увидеть вашу новую веб-страницу, отображающую изображение!

Некоторые общие правила о путях к файлам:

- Для ссылки на целевой файл в той же директории, что и вызывающий HTML файл, просто используйте имя файла, например, my-image.jpg.
- Для ссылки на файл в поддиректории, напишите имя директории в начале пути, плюс косую черту (forwardslash, слеш), например: subdirectory/my-image.jpg.
- Для ссылки на целевой файл в директории выше вызывающего HTML файла, напишите две точки. Например, если index.html находится внутри подпапки test-site, а my-image.png - внутри test-site, вы можете обратиться к my-image.png из index.html, используя ../my-image.png.
- Вы можете комбинировать их так, как вам нравится, например ../subdirectory/another-subdirectory/my-image.png.

Основы CSS: наследование, каскад и специфичность.

CSS (Cascading Style Sheets) означает Каскадные Таблицы Стилей и первое слово *"каскадные"* является невероятно важным для понимания: то, как ведёт себя каскад — ключевой момент в понимании CSS.

В какой-то момент, работая над проектом, вы обнаружите, что CSS, который, по-вашему, должен быть применён к элементу, не работает. Обычно проблема заключается в том, что вы создали два правила, которые могут потенциально применяться к одному и тому же элементу. Каскад и тесно связанная концепция специфичности — это механизмы, которые контролируют, какое именно правило применяется, когда имеется такой конфликт. Стиль вашего элемента может определять не то правило, на которое вы рассчитывали, поэтому вам необходимо понимать, как работают эти механизмы.

Также значимой является концепция наследования, которая заключается в том, что некоторые свойства CSS наследуют по умолчанию значения, установленные для родительского элемента текущего элемента, а некоторые не наследуют. Это также может стать причиной поведения, которое вы, возможно, не ожидаете.

Давайте начнём с краткого обзора ключевых моментов, которых мы касаемся, далее рассмотрим каждый из них по очереди и посмотрим, как они взаимодействуют друг с другом и с вашим CSS. Это может показаться набором сложных для понимания понятий. Однако, когда вы получите больше опыта в написании CSS, для вас станет более очевидным то, как это работает.

Каскад

Каскад таблицы стилей, если говорить упрощённо, означает, что порядок следования правил в CSS имеет значение; когда применимы два правила, имеющие одинаковую специфичность, используется то, которое идёт в CSS последним.

В приведённом ниже примере у нас есть два правила, которые могут применяться к h1. В результате h1 окрасится синим цветом — эти правила имеют идентичный селектор и, следовательно, одинаковую специфичность, поэтому побеждает последний в порядке следования.

This is my heading.

```
h1 {

  color: red;

}

h1 {

  color: blue;

}

<h1>This is my heading.</h1>
```

Специфичность

Специфичность определяет, как браузер решает, какое именно правило применяется в случае, когда несколько правил имеют разные селекторы, но, тем не менее, могут быть применены к одному и тому же элементу. *Различные типы селекторов (селекторы элементов h1{...}, селекторы классов, селекторы идентификаторов и т.д) имеют разной степени влияние на элементы страницы. Чем более общее влияние оказывает селектор на элементы страницы тем меньше его специфичность, конкретность.* По существу, это мера того, насколько специфическим будет отбор по селектору:

- Селектор элементов (*например h1*) менее специфичен — он выберет все элементы этого типа на странице — поэтому получит меньше баллов.
- Селектор класса более специфичен — он выберет только те элементы на странице, которые имеют конкретное значение атрибута class — поэтому получит больше баллов, *селектор класса применится после селектора элемента и поэтому перекроет его стили*.

Например. Как указано ниже, у нас опять есть два правила, которые могут применяться к h1. h1 в результате будет окрашен красным цветом — селектор класса даёт своему правилу более высокую специфичность, поэтому он будет применён, несмотря на то, что правило для селектора элемента расположено ниже в таблице стилей.

This is my heading.

.main-heading {

```
  color: red;
}

h1 {
  color: blue;
}

<h1 class="main-heading">This is my heading.</h1>
```

Позже мы объясним, как сделать оценку специфичности, и прочие детали.

Наследование

Наследование также надо понимать в этом контексте — некоторые значения свойства CSS, установленные для родительских элементов наследуются их дочерними элементами, а некоторые нет.

Например, если вы установили значение color и font-family для элемента, то каждый элемент внутри него также будет иметь этот цвет и шрифт, если только вы не применили к ним напрямую стиль с другим цветом и шрифтом.

As the body has been set to have a color of blue this is inherited through the descendants.

We can change the color by targeting the element with a selector, such as this span.

```
body {
  color: blue;
}

span {
  color: black;
}
```

```
<p>As the body has been set to have a color of blue this is inherited through the descendants.</p>

<p>We can change the color by targeting the element with a selector, such as this <span>span</span>.</p>
```

Некоторые свойства не наследуются — например, если вы установили для элемента width равным 50%, все его дочерние элементы не получат ширину в 50% от ширины своего родительского элемента. Если бы это было так, CSS было бы чрезвычайно трудно использовать!

Понимание взаимодействия этих концепций

Эти три концепции вместе определяют, какая CSS применяется и к какому элементу; в следующих разделах мы увидим, как они взаимодействуют. Это может показаться сложным, но вы начнёте лучше понимать их по мере приобретения опыта работы с CSS, и вы всегда можете обратиться к справочной информации, если что-то забыли. Даже опытные разработчики не помнят всех деталей!

Понимание наследования

Итак, наследование. В примере ниже мы имеем <ul> с двумя уровнями неупорядоченных списков, вложенных в него. Мы установили для внешнего <ul> стиль границы, внутренние отступы и цвет шрифта.

Цвет шрифта применён к прямому потомку, но также и к непрямому потомку — к прямому потомку <li> и к элементам внутри первого вложенного списка. Далее мы добавили класс special ко второму вложенному списку и применили к нему другой цвет шрифта. Теперь это свойство наследуется всеми его потомками.

- Item One
- Item Two
  - 2.1
  - 2.2
- Item Three
  - **3.1**
    - **3.1.1**
    - **3.1.2**
  - **3.2**

```
.main {
  color: rebeccapurple;
  border: 2px solid #ccc;
  padding: 1em;
}

.special {
  color: black;
  font-weight: bold;
}
<ul class="main">
  <li>Item One</li>
  <li>Item Two
    <ul>
      <li>2.1</li>
```

```
      <li>2.2</li>
    </ul>
  </li>
  <li>Item Three
    <ul class="special">
      <li>3.1
        <ul>
          <li>3.1.1</li>
          <li>3.1.2</li>
        </ul>
      </li>
      <li>3.2</li>
    </ul>
  </li>
</ul>
```

Такие свойства, как ширина (как в примере выше), внутренние и внешние отступы и стиль границы не наследуются. Если бы потомки нашего списка наследовали бы границу, то каждый отдельный список и каждая позиция в списке получили бы такую же границу — вряд ли мы хотели бы получить такой эффект!

Какие свойства наследуются по умолчанию, а какие нет, чаще всего определяется здравым смыслом.

Контроль наследования

CSS предоставляет четыре специальных универсальных значения свойства для контроля наследования. Каждое свойство CSS принимает эти значения.

inherit

Устанавливает значение свойства, применённого к элементу, таким же, как у его родительского элемента. Фактически, это "включает наследование".

initial

Устанавливает значение свойства, применённого к выбранному элементу, равным initial value этого свойства (*в соответствии с настройками браузера по умолчанию. Если в таблице стилей браузера отсутствует значение этого свойства, оно наследуется естественным образом.)*

unset (en-US)

Возвращает свойству его естественное значение, что означает, что если свойство наследуется естественным образом, оно действует как inherit, иначе оно действует как initial.

Можно посмотреть список ссылок и изучить, как работают универсальные значения. Пример, следующий ниже, позволяет вам поиграть с CSS и увидеть, что происходит, когда вы вносите изменения. Подобные эксперименты с кодом — лучший способ освоить HTML и CSS.

Например:

1. Второй элемент списка имеет класс my-class-1. Таким образом, цвет для следующего вложенного элемента a устанавливается по наследству. Как изменится цвет, если это правило будет удалено?
2. Понятно ли, почему третий и четвёртый элементы a имеют именно такой цвет? Если нет, перечитайте описание значений, представленное выше.
3. Какая из ссылок изменит цвет, если вы зададите новый цвет для элемента <a> — например: a { color: red; }?
4. Default link color
5. Inherit the link color
6. Reset the link color
7. Unset the link color

```
body {

  color: green;

}
```

```
.my-class-1 a {
  color: inherit;
}

.my-class-2 a {
  color: initial;
}

.my-class-3 a {
  color: unset;
}
<ul>
  <li>Default <a href="#">link</a> color</li>
  <li class="my-class-1">Inherit the <a href="#">link</a> color</li>
```

```
  <li class="my-class-2">Reset the <a href="#">link</a> color</li>

  <li class="my-class-3">Unset the <a href="#">link</a> color</li>

</ul>
```

Возврат всех исходных значений свойств

Стенографическое свойство CSS all можно использовать для того, чтобы присвоить одно из значений наследования к (почти) всем свойствам одновременно. Это одно из четырёх значений (inherit, initial, unset, или revert). Это удобный способ для отмены изменений, внесённых в стили, для того, чтобы вы могли вернуться к стартовой точке перед внесением новых изменений.

В примере ниже имеются два блока <blockquote>. Первый имеет стиль, который применён к самому элементу blockquote, второй имеет класс fix-this, который устанавливает значение all в unset.

> This blockquote is styled

> This blockquote is not styled

```
blockquote {

  background-color: orange;

  border: 2px solid blue;
```

```
}



.fix-this {

  all: unset;

}

<blockquote>

  <p>This blockquote is styled</p>

</blockquote>



<blockquote class="fix-this">

  <p>This blockquote is not styled</p>

</blockquote>
```

Попробуйте установить для all ещё одно из доступных значений и исследуйте, в чём заключается разница.

Понимание каскада

Теперь мы понимаем, почему параграф, следующий по глубине в структуре HTML документа, имеет тот же цвет, что CSS применяет к body, а вводные уроки дали понимание того, как изменить применение CSS к чему-либо в любой точке документа — или назначить CSS элементу, или создать класс. Теперь рассмотрим подробнее то, как каскад определяет выбор CSS-правил, применяемых в случае влияния на стиль элемента нескольких объектов.

Вот три фактора, перечисленные в порядке возрастания важности. Следующий отменяет предыдущий.

1. Порядок следования
2. Специфичность
3. Важность

Мы внимательно изучим их, чтобы увидеть, как именно браузеры определяют, какой CSS следует применить.

Порядок следования

Мы уже видели, какое значение для каскада имеет порядок следования. Если у вас несколько правил, которые имеют одинаковую важность, то побеждает правило, которое идёт последним в CSS. Другими словами, правила, более близкие к самому элементу, переписывают более ранние, пока последнее не победит, оно и стилизует элемент.

Специфичность

Понимая, что порядок следования правил имеет значение, в какой-то момент вы окажетесь в ситуации, когда вы знаете, что правило появляется позже в таблице стилей, но применяется более раннее, конфликтующее правило. Это связано с тем, что более раннее правило имеет более высокую специфичность — оно более специфично и поэтому выбирается браузером как правило, которое должно стилизовать элемент.

Как мы видели ранее в этом уроке, селектор класса имеет больший вес, чем селектор элемента, поэтому свойства, определённые в классе, будут переопределять свойства, применённые непосредственно к элементу.

Здесь следует отметить, что, хотя мы думаем о селекторах и правилах, применяемых к объекту, который они выбирают, переписывается не всё правило, а только свойства, которые являются одинаковыми.

Такое поведение помогает избежать повторения в вашем CSS. Обычной практикой является определение общих стилей для базовых элементов, а затем создание классов для тех, которые отличаются. Например, в таблице стилей ниже мы определяем общие стили для заголовков второго уровня, а затем создаём несколько классов, которые изменяют только некоторые свойства и значения. Определённые вначале значения применяются ко всем заголовкам, затем к заголовкам с классами применяются более конкретные значения.

Heading with no class

Heading with class of small

Heading with class of bright

```
h2 {
  font-size: 2em;
  color: #000;
  font-family: Georgia, 'Times New Roman', Times, serif;
}

.small {
  font-size: 1em;
}

.bright {
  color: rebeccapurple;
}
<h2>Heading with no class</h2>
<h2 class="small">Heading with class of small</h2>
```

<h2 class="bright">Heading with class of bright</h2>

Давайте теперь посмотрим, как браузер будет вычислять специфичность. Мы уже знаем, что селектор элемента имеет низкую специфичность и может быть перезаписан классом. По существу, значение в баллах присуждается различным типам селекторов, и их сложение даёт вам вес этого конкретного селектора, который затем может быть оценён в сравнении с другими потенциальными соперниками.

Степень специфичности, которой обладает селектор, измеряется с использованием четырёх различных значений (или компонентов), которые можно представить как тысячи, сотни, десятки и единицы — четыре однозначные цифры в четырёх столбцах:

1. Тысячи: поставьте единицу в эту колонку, если объявление стиля находится внутри атрибута style (встроенные стили). Такие объявления не имеют селекторов, поэтому их специфичность всегда просто 1000.
2. Сотни: поставьте единицу в эту колонку за каждый селектор ID, содержащийся в общем селекторе.
3. Десятки: поставьте единицу в эту колонку за каждый селектор класса, селектор атрибута или псевдокласс, содержащийся в общем селекторе.
4. Единицы: поставьте общее число единиц в эту колонку за каждый селектор элемента или псевдоэлемент, содержащийся в общем селекторе.

Следующая таблица показывает несколько несвязанных примеров, которые помогут вам разобраться. Посмотрите их все и убедитесь, что вы понимаете, почему они обладают той специфичностью, которую мы им дали. Мы ещё не рассмотрели селекторы детально, но вы можете найти подробную информацию о каждом селекторе в справочнике селекторов MDN.

Селектор	Тысячи	Сотни	Десятки	Единицы	Специфичность
h1	0	0	0	1	0001
h1 + p::first-letter	0	0	0	3	0003
li > a[href*="en-US"] > .inline-warning	0	0	2	2	0022
#identifier	0	1	0	0	0100
Без селектора, с правилом внутри атрибута style элемента.	1	0	0	0	1000

Прежде чем мы продолжим, давайте посмотрим на пример в действии.

One

Two

```
/* 1. specificity: 1-0-1 */

#outer a {

  background-color: red;

}



/* 2. specificity: 2-0-1 */

#outer #inner a {

  background-color: blue;

}



/* 3. specificity: 1-0-4 */

#outer div ul li a {

  color: yellow;

}
```

```
/* 4. specificity: 1-1-3 */

#outer div ul .nav a {

  color: white;

}



/* 5. specificity: 0-2-4 */

div div li:nth-child(2) a:hover {

  border: 10px solid black;

}



/* 6. specificity: 0-2-3 */

div li:nth-child(2) a:hover {

  border: 10px dashed black;

}
```

```
/* 7. specificity: 0-3-3 */

div div .nav:nth-child(2) a:hover {

  border: 10px double black;

}



a {

  display: inline-block;

  line-height: 40px;

  font-size: 20px;

  text-decoration: none;

  text-align: center;

  width: 200px;

  margin-bottom: 10px;

}
```

```
ul {
  padding: 0;
}

li {
  list-style-type: none;
}
<div id="outer" class="container">
  <div id="inner" class="container">
    <ul>
      <li class="nav"><a href="#">One</a></li>
      <li class="nav"><a href="#">Two</a></li>
    </ul>
  </div>
```

Так что здесь происходит? Прежде всего, нас интересуют только первые семь правил этого примера, и, как вы заметите, мы включили их значения специфичности в комментарий перед каждым правилом.

- Первые два правила конкурируют за стилизацию цвета фона ссылки — второе выигрывает и делает фоновый цвет синим, потому что у него есть дополнительный селектор ID в цепочке: его специфичность 201 против 101.
- Третье и четвёртое правило конкурируют за стилизацию цвета текста ссылки — второе выигрывает и делает текст белым, потому что, хотя у него на один селектор элемента меньше, отсутствующий селектор заменяется на селектор класса, который оценивается в десять вместо единицы. Таким образом, приоритетная специфичность составляет 113 против 104.
- Правила 5–7 соревнуются за определение стиля границы ссылки при наведении курсора. Шестой селектор со специфичностью 23 явно проигрывает пятому со специфичностью 24 — у него в цепочке на один селектор элемента меньше. Седьмой селектор, однако, превосходит как пятый, так и шестой — он имеет то же количество подселекторов в цепочке, что и пятый, но один элемент заменён селектором класса. Таким образом, приоритетная специфичность 33 против 23 и 24.

Примечание: Это был условный пример для более простого усвоения. В действительности, каждый тип селектора имеет собственный уровень

специфичности, который не может быть замещён селекторами с более низким уровнем специфичности. Например, *миллион* соединённых селекторов класса не способны переписать правила *одного* селектора id. Более правильный способ вычисления специфичности состоит в индивидуальной оценке уровней специфичности, начиная с наивысшего и продвигаясь к самому нижнему, когда это необходимо. Только когда оценки уровня специфичности совпадают, следует вычислять следующий нижний уровень; в противном случае, вы можете пренебречь селекторами с меньшим уровнем специфичности, поскольку они никогда не смогут преодолеть уровни более высокой специфичности.

!important

Существует специальный элемент CSS, который вы можете использовать для отмены всех вышеперечисленных вычислений, однако вы должны быть очень осторожны с его использованием — !important. Он используется, чтобы сделать конкретное свойство и значение самыми специфичными, таким образом переопределяя нормальные правила каскада.

Взгляните на этот пример, где у нас есть два абзаца, один из которых имеет ID.

This is a paragraph.

One selector to rule them all!

```
#winning {
  background-color: red;
  border: 1px solid black;
}

.better {
  background-color: gray;
  border: none !important;
}

p {
  background-color: blue;
  color: white;
  padding: 5px;
}
```

```
<p class="better">This is a paragraph.</p>

<p class="better" id="winning">One selector to rule them all!</p>
```

Давайте пройдёмся по этому примеру, чтобы увидеть, что происходит — попробуйте удалить некоторые свойства, чтобы увидеть, что получится, если вам трудно понять:

1. Вы увидите, что применены значения color (en-US) и padding третьего правила, но background-color — нет. Почему? Действительно, все три безусловно должны применяться, потому что правила, более поздние в порядке следования, обычно переопределяют более ранние правила.
2. Однако вышеприведённые правила выигрывают, потому что селекторы классов имеют более высокую специфичность, чем селекторы элементов.
3. Оба элемента имеют class с названием better, но у второго также есть id с названием winning. Поскольку ID имеют *ещё более высокую* специфичность, чем классы (у вас может быть только один элемент с каждым уникальным ID на странице, но много элементов с одним и тем же классом — селекторы ID *очень специфичны*, на что они и нацелены), красный цвет фона и однопиксельная чёрная граница должны быть применены ко 2-му элементу, причём первый элемент получает серый фоновый цвет и отсутствие границы, как определено классом.

4. 2-й элемент получил красный цвет фона и отсутствие границы. Почему? Из-за объявления !important во втором правиле — размещение которого после border: none означает, что это объявление перевесит значение границы в предыдущем правиле, даже если ID имеет более высокую специфичность.

Примечание: Единственный способ переопределить объявление !important – это включить другое объявление !important в правило *с такой же специфичностью* позже или в правило с более высокой специфичностью.

Полезно знать о существовании !important, чтобы вы понимали, что это такое, когда встретите в чужом коде. Тем не менее, мы настоятельно рекомендуем вам никогда не использовать его, если в этом нет острой необходимости. !important меняет обычный порядок работы каскада, поэтому он может серьёзно затруднить отладку проблем CSS, особенно в большой таблице стилей.

Одна из ситуаций, в которой вам, возможно, придётся это использовать, — это когда вы работаете с CMS, где вы не можете редактировать модули CSS ядра, и вы действительно хотите переопределить стиль, который нельзя переопределить другим способом. Но, вообще говоря, не стоит использовать этот элемент, если можно этого избежать.

Влияние расположения CSS

Наконец, также полезно отметить, что важность объявления CSS зависит от того, в какой таблице стилей оно указано — у пользователя есть возможность установить индивидуальные таблицы стилей для переопределения стилей разработчика, например, пользователь может

иметь проблемы со зрением и захочет установить размер шрифта на всех посещаемых им веб-страницах в два раза больше нормального размера, чтобы облегчить чтение.

Подведение итогов

Конфликтующие объявления будут применяться в следующем порядке, с учётом замены более ранних более поздними:

1. Объявления в таблицах стилей клиентского приложения (например, стили браузера по умолчанию, используемые, когда не заданы другие стили).
2. Обычные объявления в пользовательских таблицах стилей (индивидуальные стили устанавливаются пользователем).
3. Обычные объявления в авторских таблицах стилей (это стили, установленные нами, веб-разработчиками).
4. Важные объявления в авторских таблицах стилей.
5. Важные объявления в пользовательских таблицах стилей.

Для таблиц стилей веб-разработчиков имеет смысл переопределить пользовательские таблицы стилей так, чтобы можно было сохранить запланированный дизайн, но иногда у пользователей есть веские причины для переопределения стилей веб-разработчика, как упомянуто выше — это может быть достигнуто с помощью использования !important в их правилах.

Использование контентной и декоративной графики в проекте

Давайте разберёмся, что такое контентное изображение, что такое декоративное и как их отличить друг от друга. От типа изображения зависит то, как оно будет добавлено на страницу.

Теория

Контентное изображение

Слово контент происходит от английского слова «content» — содержимое. С помощью таких изображений мы можем донести до пользователей полезную информацию. Чтобы добавить контентное изображение к себе на страницу, используйте в разметке <img>. Для изображений такого типа необходимо заполнять атрибут alt, который описывает то, что изображено на картинке.

Декоративное изображение

Из названия понятно, что этот тип изображений используется исключительно для оформления. Такие изображения не несут для пользователя полезной информации. Декоративные изображения следует реализовывать с помощью CSS.

Практика

В теории всё просто, но на деле мы можем столкнуться с неоднозначными ситуациями, в которых не так-то просто определить тип изображения. Давайте вместе рассмотрим различные ситуации на примерах.

Очевидные случаи

Изображение товара в карточке товара, контентное

Такие изображения содержат информацию о внешнем виде товара. Если они по какой-либо причине не отобразятся, то пользователь потеряет часть информации о товаре.

Реализация: <img>

Логотип, контентное

Логотипы могут быть реализованы как текстом, так и изображением, поэтому нужно ориентироваться на конкретный макет. В приведённых выше примерах логотип представлен изображением, которое содержит важную информацию о названии сайта.

И также часто встречаются блоки с партнёрскими логотипами. В таких ситуациях логотипы являются контентными изображениями.

Реализация: <img>

Изображения и текст, контентное

Изображения в статье. Данный случай ничем не отличается от карточки товара — у нас есть и описание, и изображение, несущие важную для пользователя информацию. Таким образом, изображение является контентным и должно быть сделано тегом <img>.

Реализация: <img>

Вспомогательная иконка, декоративное

Иконки в меню имеют декоративный характер. Если вдруг они пропадут, информация не потеряется, и пользователь всё равно сможет понять в какой раздел ему нужно. Для реализации лучше всего использовать background-image для псевдоэлемента.

Реализация: background-image для псевдоэлемента

Фоновое изображение, декоративное

Для реализации лучше всего подойдёт background-image для всего блока.

Реализация: background-image для всего блока

Спорные моменты

Изображение в промослайдере

В элементах слайдера нам часто встречаются изображения товаров. Они несут в себе важную визуальную информацию о товаре, поэтому в таком случае нам следует отнести их к контентному типу изображений.

Реализация: <img>

Однако, бывает и такое, что в слайдере изображения никак не относятся к тексту слайдов и служат только для декоративных целей, следовательно, относятся к декоративному типу изображений.

Реализация: background-image для всего блока

Карта с изображением

Под интерактивную карту принято добавлять изображение с картой, на случай если интерактивная карта не загрузилась. Каким же образом её добавлять? Карта несёт информацию об адресе. В блоке «Контакты» только по карте можно понять адрес компании, ведь в тексте адрес не продублирован, а значит если изображение потеряется, пользователь потеряет информацию о местонахождении компании. Таким образом, изображения карт относятся к контентным изображениям.

Реализация: <img> (атрибут alt должен описывать изображение, в данном случае — Карта офиса по адресу)

Иконки соцсетей

Кнопки с социальными сетями представляют собой более интересный случай. С одной стороны, они являются декоративными, так как являются частью интерфейса сайта, однако, если их картинки не загрузятся — информация всё же пропадёт (пользователь будет не способен понять к какой социальной сети относится каждая ссылка). Таким образом, здесь применяется комплексный подход: внутри ссылки обязательно прописывается поясняющий текст. Изображение на этих кнопках-ссылках декоративные. Для доступного скрытия текста ссылки необходимо добавить класс .visually-hidden. Таким образом, при потере CSS-файла, изображение пропадёт, а текст ссылки появится. При нормальной работе сайта пользователь увидит лишь изображение. Также необходимо не забывать про доступность — так как в разметке мы прописываем текст ссылки, то при чтении сайта скринридером, ссылки будут озвучены.

Реализация: background-image для ссылки

Вывод

Вопрос определения типа изображения способен запутать неокрепший ум начинающего верстальщика, поэтому советуем вам каждый раз, когда вы сталкиваетесь с необходимостью определить тип изображения, обращать внимание на все детали, а не идти самым простым путём.