UPD Просьба для всех тружеников и терминаторов, которые помогают писать данный документ, по возможности в теории писать текстом, картинки добавлять поменьше.

АТЕНШН!!!

Тот кто будет ломать документ или портить его, получит по жопе

РАЗРАБОТКА КЛИЕНТСКИХ ЧАСТЕЙ ИНТЕРНЕТ РЕСУРСОВ КАФЕДРА ИНСТРУМЕНТАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ/ ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

1. Базовая структура HTML-документа

```
<!DOCTYPE html>                                          DTD(описание типа документа)

<html lang="ru">

<head>

        <meta charset="UTF-8">                           позволяет установить кодировку докумnента

        <title>Моя первая страница</title>

</head>

<body>

</body>

</html>
```

2. Понятие атрибутов в HTML-документе

Атрибут — используется для определения характеристик html-элемента и помещается внутри открытого тега элемента. Все атрибуты состоят из двух частей — это имя и значение:

- Имя — это свойство, которое Вы хотите установить. Например, элемент абзаца <p>, в примере ниже, содержит атрибут align, который Вы можете использовать для указания выравнивания абзаца на странице.
- Значение — это значение, которое Вы хотите установить для свойства. Значение атрибута всегда помещается в кавычки. В приведенном ниже примере показаны три возможных значения атрибута align: left, center и right.

The <a> tag defines a hyperlink. The **href attribute** specifies the URL of the page the link goes to:

3. Понятие тегов в HTML-документе

Тег — это элемент языка HTML, с помощью которого выполняется разметка исходного текста веб-страницы. Теги представляют из себя сокращения или аббревиатуры английских слов заключенные в угловые скобки <>

Кроме того теги подразделяют на несколько типов, которое различаются по выполняем функциям:

теги верхнего уровня; теги заголовка документа; блочные элементы; строчные элементы; универсальные элементы; списки; таблицы; фреймы.

4. Применение элемента DOCTYPE в HTML-документе

Элемент <!DOCTYPE> предназначен для указания типа текущего документа — DTD (document type definition, описание типа документа). Это необходимо, чтобы браузер понимал, как следует интерпретировать текущую веб-страницу, ведь HTML существует в нескольких версиях, кроме того, имеется XHTML (Extensible HyperText Markup Language, расширяемый язык разметки гипертекста), похожий на HTML, но различающийся с ним по синтаксису. Чтобы браузер «не путался» и понимал, согласно какому стандарту отображать веб-страницу и необходимо в первой строке кода задавать <!DOCTYPE>.

Существует несколько видов <!DOCTYPE>, они различаются в зависимости от версии HTML

5. Понятие документной объектной модели в HTML

Все элементы являются документами. Все документы являются объектами. Все объекты являются узлами.**DOM** (от англ. *Document Object Model* — «объектная модель документа») — это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Объектная модель документа это программный интерфейс (что-то вроде API) для HTML документов. DOM это структурное представление документа, которая определяет то, как с этим документом будут взаимодействовать программы на JavaScript, файлы стилей и т.п.

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями «родительский-дочерний».

6. Применение favicon при HTML-верстке

Фавиконка (favicon) — это иконка, которая отображается во вкладке браузера перед названием страницы. Также эта иконка отображается в закладках и на рабочем столе для веб-приложений.

```
<head>
 <link rel="shortcut icon" href="/images/favicon.png" type="image/png">

</head>
```

<link rel="icon" type="image/x-icon" href="favicon.ico" />

7. Использование CSS стилей в HTML-документе

```
<head>
  ...
  <link rel="stylesheet" href="mysite.css">
  <link rel="stylesheet" href="http://www.htmlbook.ru/main.css">
 </head>
```

Каскадные таблицы стилей CSS (Cascading Style Sheets) нужны для оформления страниц вашего сайта в соответствии с разработанным стилем, дизайном.

8. Использование якорей для навигации в HTML-документе

HTML якоря используются для создания постраничной навигации HTML документа.

HTML якоря удобно применять в том случае, если ваш HTML документ очень большого размера и посетителю бывает непросто сориентироваться внутри данного документа. Обратите внимание: у HTML якоря обычно нет содержимого, это делается намеренно.

Чтобы осуществить переход к HTML якорю на другой странице, вам нужно сперва указать путь к странице, затем написать символ «#» и затем имя якоря. Если вы создали ссылку на другую страницу с якорем, а якоря нет, то браузер осуществит переход к началу указанного HTML документа и при этом это не будет считаться ошибкой.

<!— other.html#smth —>

...

9. Использование технологии CSS Flexbox Layout при построении сайта

```
.container {

  display: flex;

}
```

CSS flexbox (Flexible Box Layout Module) — модуль макета гибкого контейнера — представляет собой способ компоновки элементов, в основе лежит идея оси.

Flexbox состоит из гибкого контейнера (flex container) и гибких элементов (flex items).

Свойства flex-box:

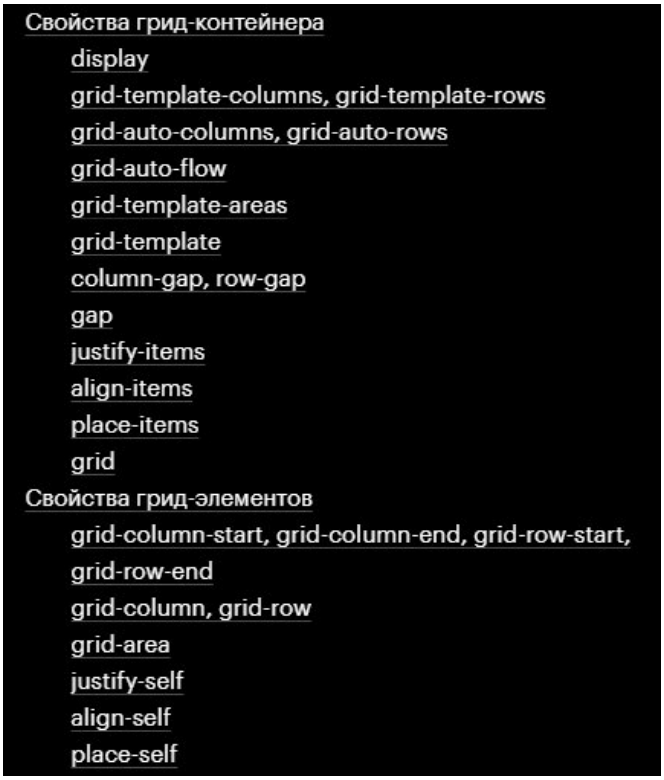
- Направление главной оси: flex-direction
- Управление многострочностью flex-контейнера: flex-wrap
- Краткая запись направления и многострочности: flex-flow
- Порядок отображения flex-элементов: order
- Гибкость flex-элементов
- Задание гибких размеров одним свойством: flex
- Коэффициент роста: flex-grow
- Коэффициент сжатия: flex-shrink
- Базовый размер: flex-basis
- Выравнивание
- Выравнивание по главной оси: justify-content
- Выравнивание по поперечной оси: align-items и align-self
- Выравнивание строк flex-контейнера: align-content

10. Использование технологии CSS Grid Layout при построении сайта

Свойство CSS grid является сокращённой формой записи, которая устанавливает значения для всех явных свойств сетки (grid) (grid-template-rows, grid-template-columns, и grid-template-areas), всех неявных свойств сетки (grid) (grid-auto-rows, grid-auto-columns, и grid-auto-flow), и свойств для промежутков между рядами и столбцами сетки (grid-column-gap и grid-row-gap) в одной строчке.

Начальные значения свойств грид-контенера

- grid-template-rows: none
- grid-template-columns: none
- grid-template-areas: none
- grid-auto-rows: auto
- grid-auto-columns: auto
- grid-auto-flow: row
- grid-column-gap: 0
- grid-row-gap: 0
- column-gap: normal
- row-gap: normal



11. Создание растрового двухмерного изображения, используя тег canvas

Элемент <canvas>, добавленный в HTML5, предназначен для создания графики с помощью JavaScript. Например, его используют для рисования графиков, создания фотокомпозиций, анимаций и даже обработки и рендеринга видео в реальном времени.

Предварительная «настройка» нашего холста

У нашего подопытного тега есть всего два атрибута — height и width, высота и ширина соответственно, по умолчанию размер холста 150x300 пикселей.

<**canvas** height='320' width='480' id='example'>Обновите браузер</**canvas**>

```
<script>
        var example = document.getElementById("example"),
                                ctx     = example.getContext('2d');
        example.width  = 640;
        example.height = 480;
        ctx.strokeRect(15, 15, 266, 266);
        ctx.strokeRect(18, 18, 260, 260);
        ctx.fillRect(20, 20, 256, 256);
        for (i = 0; i < 8; i += 2)
           for (j = 0; j < 8; j += 2) {
              ctx.clearRect(20 + i * 32, 20 + j * 32, 32, 32);
              ctx.clearRect(20 + (i + 1) * 32, 20 + (j + 1) * 32, 32, 32);
           }
      </script>
```

Рисование фигур составленных из линий выполняется последовательно в несколько шагов:

```
beginPath()
closePath()
stroke()
fill()
```

```
<script>var example = document.getElementById("example"),
                                ctx     = example.getContext('2d');
                            example.height = 480;
                            example.width  = 640;
                            ctx.beginPath();
                            ctx.arc(80, 100, 56, 3/4 * Math.PI, 1/4 * Math.PI, true);
                            ctx.fill(); // *14
                            ctx.moveTo(40, 140);
                            ctx.lineTo(20, 40);
                            ctx.lineTo(60, 100);
                            ctx.lineTo(80, 20);
                            ctx.lineTo(100, 100);
                            ctx.lineTo(140, 40);
                            ctx.lineTo(120, 140);
                            ctx.stroke(); // *22</script>
```

12. Правила построения CSS-стиля

- если все подобные элементы (например, все заголовки h1) должны иметь один стиль, то селектор состоит только из этого элемента (например, p{color:black;})

- если элемент (любой: абзац, заголовок...) должен отличаться от всех остальных, то ему присваивается идентификатор (id) и разделителем в таблице стилей является знак решетки (#), например, p#pink{color:pink;} .

- если же на странице будет несколько элементов с одинаковым стилем, то им присваивается класс (class) и разделителем в таблице стилей является знак точки (.), например, p.pink{color:pink;}.

- идентификатор имеет более высокий приоритет, чем класс. Поэтому, если для какого-либо элемента будет указан и класс, и идентификатор (что не противоречит принципам CSS), то применен будет стиль идентификатора.

ДОДЕЛАТЬ

13. Правило @import в CSS-стиле

Правило @import позволяет импортировать содержимое CSS-файла в текущую стилевую таблицу. @import не разрешается вставлять после любых объявлений кроме @charset или другого @import.

```
@import url("имя файла") [типы носителей];
@import "имя файла" [типы носителей];
```

Тип	Описание
all	Все типы. Это значение используется по умолчанию.
aural	Речевые синтезаторы, а также программы для воспроизведения текста вслух. Сюда, например, можно отнести речевые браузеры.
braille	Устройства, основанные на системе Брайля, которые предназначены для слепых людей.
handheld	Наладонные компьютеры и аналогичные им аппараты.
print	Печатающие устройства вроде принтера.
projection	Проектор.
screen	Экран монитора.
tv	Телевизор.

14. Использование универсального селектора * {}

Используется для назначения определенных атрибутов сразу всем элементам страницы.

Может использоваться не только для управления всеми компонентами документа, но и для контроля всеми дочерними элементами определенного блока, например, div *{}.

Достоинства — можно управлять большим количеством элементов одновременно.

Недостатки — недостаточно гибкий вариант. Вдобавок, использование этого селектора в некоторых случаях замедляет работу страницы.

15. Использование селектора классов

Предназначен для управления тегами с атрибутом class.

Несколько правил написания классов:

- в CSS перед названием селектора класса обязательно ставится точка (но при присвоении класса в HTML-документе эта точка не нужна);
- в названии классов можно использовать только буквы латинского алфавита, дефис, символ подчеркивания, цифры;
- название класса всегда должно начинаться с буквы (правильные варианты названий: .intro, .img-border, .nav_menu_01; неправильные: .2color, .-link, ._divider);
- названия классов CSS чувствительны к регистру, поэтому классы вроде .review и .Review будут восприниматься как два отдельных.

16. Использование селектора идентификаторов

- Выбор элемента по уникальному идентификатору (значению атрибута id).
- Обозначение — название идентификатора с предшествующим ему значком #.
- Идентификатор должен быть уникальным в пределах HTML-страницы. То есть, предполагается, что селектор идентификатора выберет только один элемент. Надеюсь, помнишь, что один и тот же класс можно присваивать любому количеству элементов. Существенная оговорка — следить, чтобы id был действительно единственным придется тебе.
- Так как в атрибуте id не допустимо использовать несколько значений, то в отличие от множественных классов, множественных идентификаторов не бывает.
- При определении того, какие стили должны применяться к данному элементу идентификаторы имеют больший вес, чем классы. Подробнее этот вопрос будет рассмотрен в разделе, посвященном наследованию.

17. Использование селектора атрибутов

Селектор [attribute*="value"] - Позволяет выбрать элементы, у которых значение атрибута содержит указанный текст. При этом текст может быть в начале, в

середине или в конце. К примеру, для выбора всех элементов, у которых класс содержит view, используется следующая запись.

```
[class*="view"] { color: red; }
```

При этом цвет изменится для элементов с классом view, view-block, viewer, overview или block-view.

18. Использование вложенных селекторов

При создании веб-страницы часто приходится вкладывать одни теги внутрь других. Чтобы стили для этих тегов использовались корректно, помогут вложенные селекторы. Например, задать стиль для тега **<b>** только когда он располагается внутри контейнера **<p>**. Таким образом можно одновременно установить стиль для отдельного тега, а также для тега, который находится внутри другого.

Пример: `E F { Описание правил стиля }`

Здесь E это родительский тег, а F — дочерний тег, расположенный в контейнере <E>. В этом случае стиль будет применяться к тегу <F>, когда соблюдается следующий код <E><F></F></E>. Не обязательно должно быть два тега, допускается произвольный уровень вложения.

19. Использование селекторов одного уровня

Пример: `E+F { Описание правил стиля }`

Здесь E это родительский тег, а F — самый первый дочерний тег первого уровня, расположенный в контейнере <E>. В этом случае стиль будет применяться к тегу <F>, когда соблюдается следующий код <E><F></F></E>.

Пример: `E~F { Описание правил стиля }`

Здесь E это родительский тег, а F — каждый дочерний тег первого уровня, расположенный в контейнере <E>. В этом случае стиль будет применяться к тегу <F>, когда соблюдается следующий код <E><F></F></E>.

20. Использование псевдоклассов

Псевдоклассы дают возможность стилизовать элемент на основе не только отношений в DOM-дереве, но и основываясь на внешних факторах, таких как история посещений (например, :visited), состояние содержимого (вроде :checked у некоторых элементов формы) или позиции курсора мыши (например, :hover определяет, находится ли курсор мыши над элементом).

Псевдокласс в CSS — *это ключевое слово, добавленное к селектору,* которое определяет его особое состояние. Например, :hover может быть использован для изменения цвета кнопки при наведении курсора на неё.

21. Использование псевдоэлементов

Псевдоэлементы позволяют задать стиль элементов не определённых в дереве элементов документа, а также генерировать содержимое, которого нет в исходном коде текста.

Синтаксис использования псевдоэлементов следующий.

```
Селектор::Псевдоэлемент { Описание правил стиля }
```

Вначале следует имя селектора, затем пишется двоеточие, после которого идёт имя псевдоэлемента. Каждый псевдоэлемент может применяться только к одному селектору, если требуется установить сразу несколько псевдоэлементов для одного селектора, правила стиля должны добавляться к ним по отдельности, как показано ниже.

22. Спецификация ECMA-262

Этот стандарт ECMA основан на нескольких технологиях, самые известные из которых - JavaScript (Netscape) и JScript (Microsoft). Язык был изобретен Brendan Eich в Netscape и впервые появился в браузере этой компании Navigator 2. В дальнейшем он присутствовал во всех браузерах от Netscape и всех - от Microsoft, начиная с Internet Explorer 3.0, и так - до наших дней.

Разработка этого стандарта началась в ноябре 1996г. Первая редакция стандарта ECMA была принята общим собранием ECMA в июне 1997г.

Стандарт ECMA был отослан в ISO/IEC JTC 1 для быстрого согласования и одобрен как международный стандарт ISO/IEC 16262 в апреле 1998г. В июне 1998 года общее собрание ECMA одобрило вторую редакцию ECMA-262, чтобы поддерживать соответствие с ISO/IEC 16262. Изменения между первой и второй редакцией - по сути редакторские правки.

Настоящий документ описывает третью редакцию стандарта и включает в себя мощные регулярные выражения, улучшенную поддержку строк, новые управляющие конструкции, обработку исключений try/catch, конкретизированное определение ошибок, форматирование при численном выводе и небольшие изменения для приближающейся интернационализации и будущего роста языка.

Работа над языком не завершена. Технический комитет работает над значительными улучшениями, включая механизмы для создания и использования скриптов в интернет и улучшенную координацию с другими разработчиками стандартов, такими как группы в консорциуме W3C и Wireless Application Protocol Forum.

Этот стандарт был принят как 3-я редакция ECMA-262 общим собранием ECMA в декабре 1999 года.

23. Объектная модель браузера

Объектная модель браузера (анг. Browser Object Model (BOM)) — это часть JavaScript, которая позволяет скрипту взаимодействовать с программой просмотра веб-страниц. BOM представляет объекты, через свойства и методы которых можно управлять внешним видом и поведением обозревателя.

Например:

- Объект navigator даёт информацию о самом браузере и операционной системе. Среди множества его свойств самыми известными являются: navigator.userAgent – информация о текущем браузере, и navigator.platform – информация о платформе (может помочь в понимании того, в какой ОС открыт браузер – Windows/Linux/Mac и так далее).
- Объект location позволяет получить текущий URL и перенаправить браузер по новому адресу.

BOM является частью общей спецификации HTML.

Да, вы всё верно услышали. Спецификация HTML по адресу https://html.spec.whatwg.org не только про «язык HTML» (теги, атрибуты), она также покрывает целое множество объектов, методов и специфичных для каждого браузера расширений DOM. Это всё «HTML в широком смысле».

24. Основные языки на базе JavaScript

TypeScript — это расширенный набор JavaScript. Допустимая программа JavaScript также допустима для TypeScript, но с добавленной статической типизацией. Компилятор также может работать как переносчик с ES2015 + на текущие реализации, так что вы всегда получаете самые последние функции.

Dart — это классический объектно-ориентированный язык, в котором все является объектом, а любой объект является экземпляром класса (объекты также могут выполнять функции функций). Он специально создан для создания приложений для браузеров, серверов и мобильных устройств.

25. Использование консоли разработчика в браузерах

В большинстве **браузеров**, работающих под Windows, инструменты **разработчика** можно открыть, нажав F12. В Chrome для Mac **используйте** комбинацию Cmd+Opt+J, Safari: Cmd+Opt+C (необходимо предварительное включение «Меню **разработчика**»).

26. Подключение внешнего скрипта JavaScript

<script src="/scripts/script.js"></script>

<script src=" http://www.somesite.com/script.js"></script>

27. Подключение нескольких скриптов JavaScript

<script src="/scripts/script.js"></script>

<script src="/scripts/script.js"></script>

28. Подключение CSS к странице

```
<link rel="stylesheet" type="text/css" href="mysite.css">
```

29. Использование директивы use strict в JavaScript

На протяжении долгого времени JavaScript развивался без проблем с обратной совместимостью. Новые функции добавлялись в язык, в то время как старая функциональность не менялась.

Преимуществом данного подхода было то, что существующий код продолжал работать. А недостатком – что любая ошибка или несовершенное решение, принятое создателями JavaScript, застревали в языке навсегда.

Так было до 2009 года, когда появился ECMAScript 5 (ES5). Он добавил новые возможности в язык и изменил некоторые из существующих. Чтобы устаревший код работал, как и раньше, по умолчанию подобные изменения не применяются. Поэтому нам нужно явно их активировать с помощью специальной директивы: "use strict".

Директива выглядит как строка: "use strict" или 'use strict'. Когда она находится в начале скрипта, весь сценарий работает в «современном» режиме.

30. Типы данных в JavaScript

В языке **JavaScript** существует несколько **типов данных**. Это числа, текстовые строки, логические **данные**, объекты, **данные** неопределенного **типа**, а также специальный **тип** null.

Стандарт ECMAScript определяет 9 типов:

- 6 типов данных являющихся примитивами:
  - Undefined (Неопределённый тип) : typeof instance === "undefined"
  - Boolean (Булев, Логический тип) : typeof instance === "boolean"
  - Number (Число) : typeof instance === "number"
  - String (Строка) : typeof instance === "string"
  - BigInt : typeof instance === "bigint"
  - Symbol (в ECMAScript 6) : typeof instance === "symbol"
- Null (Null тип) : typeof instance === "object". Специальный примитив, используемый не только для данных но и в качестве указателя на финальную точку в Цепочке Прототипов;
- Object (Объект) : typeof instance === "object". Простая структура, используемая не только для хранения данных, но и для создания других структур, где любая структура создаётся с использованием

ключевого слова new: new Object, new Array, new Map (en-US), new Set, new WeakMap, new WeakSet, new Date и множество других структур;
- и Function : typeof instance === "function". Специальный случай, упрощающий определение типа для Функций, несмотря на то, что все функции конструктивно унаследованы от Object.

31. Работа со строками в JavaScript

Строки

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков.

Внутренний формат для строк — всегда UTF-16, вне зависимости от кодировки страницы.

Кавычки

В JavaScript есть разные типы кавычек.

Строку можно создать с помощью одинарных, двойных либо обратных кавычек:

let single = 'single-quoted';let double = "double-quoted";let backticks = `backticks`;
Одинарные и двойные кавычки работают, по сути, одинаково, а если использовать обратные кавычки, то в такую строку мы сможем вставлять произвольные выражения, обернув их в ${...}:

function sum(a, b) {
  return a + b;}alert(`1 + 2 = ${sum(1, 2)}.`); // 1 + 2 = 3.
Ещё одно преимущество обратных кавычек — они могут занимать более одной строки, вот так:

let guestList = `Guests:
 * John
 * Pete
 * Mary
`;alert(guestList); // список гостей, состоящий из нескольких строк
Выглядит вполне естественно, не правда ли? Что тут такого? Но если попытаться использовать точно так же одинарные или двойные кавычки, то будет ошибка:

```
let guestList = "Guests: // Error: Unexpected token ILLEGAL
  * John";
```

Одинарные и двойные кавычки в языке с незапамятных времён: тогда потребность в многострочных строках не учитывалась. Что касается обратных кавычек, они появились существенно позже, и поэтому они гибче.

Обратные кавычки также позволяют задавать «шаблонную функцию» перед первой обратной кавычкой. Используемый синтаксис: func`string`. Автоматически вызываемая функция func получает строку и встроенные в неё выражения и может их обработать. Подробнее об этом можно прочитать в документации. Если перед строкой есть выражение, то шаблонная строка называется «теговым шаблоном». Это позволяет использовать свою шаблонизацию для строк, но на практике теговые шаблоны применяются редко.

32. Работа с логическим типом в JavaScript

Тип boolean. **В JavaScript** есть пара зарезервированных слов, использующихся при **работе с логическими** значениями — это true (истина), и false (ложь). Операции сравнения, например, такие, как == ... >, возвращают true или false. **Логические** выражения используются в конструкциях наподобие if и while, помогая управлять ходом выполнения программы.

33. Использование функций alert(), prompt(), confirm() в JavaScript

Итого

Мы рассмотрели 3 функции браузера для взаимодействия с пользователем:

alert

показывает сообщение.

prompt

показывает сообщение и запрашивает ввод текста от пользователя. Возвращает напечатанный в поле ввода текст или null, если была нажата кнопка «Отмена» или Esc с клавиатуры.

confirm

показывает сообщение и ждёт, пока пользователь нажмёт OK или Отмена. Возвращает true, если нажата OK, и false, если нажата кнопка «Отмена» или Esc с клавиатуры.

Все эти методы являются модальными: останавливают выполнение скриптов и не позволяют пользователю взаимодействовать с остальной частью страницы до тех пор, пока окно не будет закрыто.

На все указанные методы распространяются два ограничения:

1. Расположение окон определяется браузером. Обычно окна находятся в центре.
2. Визуальное отображение окон зависит от браузера, и мы не можем изменить их вид.

Такова цена простоты. Есть другие способы показать более приятные глазу окна с богатой функциональностью для взаимодействия с пользователем, но если «навороты» не имеют значения, то данные методы работают отлично.

34. Преобразование типов в JavaScript

Чаще всего операторы и функции автоматически приводят переданные им значения к нужному типу.

Например, alert автоматически преобразует любое значение к строке. Математические операторы преобразуют значения к числам.

Есть также случаи, когда нам нужно явно преобразовать значение в ожидаемый тип.

Явное преобразование часто применяется, когда мы ожидаем получить число из строкового контекста, например из текстовых полей форм.

Если строка не может быть явно приведена к числу, то результатом преобразования будет NaN. Например:

let age = Number("Любая строка вместо числа");alert(age); // NaN, преобразование не удалось
let value = true;alert(typeof value); // booleanvalue = String(value); // теперь value это строка "true"alert(typeof value); // string

35. Правила преобразования string при численном преобразовании

Правила численного преобразования:

Значение	**Преобразуется в…**
undefined	NaN
null	0
true / false	1 / 0

string	Пробельные символы по краям обрезаются. Далее, если остаётся пустая строка, то получаем 0, иначе из непустой строки «считывается» число. При ошибке результат NaN.

36. Конкатенация строк в JavaScript

Конкатенация строк – это процесс объединения двух или нескольких **строк в** одну новую **строку**. **Конкатенация** выполняется с помощью оператора +. Символ + также является оператором сложения в математических операциях. Для примера попробуйте объединить две короткие **строки**: "Sea" + "horse"; Seahorse.

```
button.onclick = function() {

  var name = prompt('What is your name?');

  alert('Hello ' + name + ', nice to see you!');}
```

37. Присваивание переменных, присваивание по цепочке в JavaScript

Простой оператор присваивания (=) используется для присваивания значения переменной. Операция присваивания вычисляется в присваиваемую величину. Присваивание по цепочке (*chaining*) используется для назначения нескольким переменным одинакового значения.

```
x = 5

y = 10

z = 25

x = y     // x будет присвоено значение 10

x = y = z // x, y и z будут равны 25
```

38. Побитовые операторы в JavaScript

Побитовые операторы интерпретируют операнды как последовательность из 32 битов (нулей и единиц). Они производят операции, используя двоичное представление числа, и возвращают новую последовательность из 32 бит (число) в качестве результата.

Список операторов:

- Побитовое И - a & b - Ставит 1 на бит результата, для которого соответствующие биты операндов равны 1.
- Побитовое ИЛИ - a | b - Ставит 1 на бит результата, для которого хотя бы один из соответствующих битов операндов равен 1.

- Побитовое исключающее ИЛИ - a ^ b - Ставит 1 на бит результата, для которого только один из соответствующих битов операндов равен 1 (но не оба).
- Побитовое НЕ - ~a - Заменяет каждый бит операнда на противоположный.
- Левый сдвиг - a << b - Сдвигает двоичное представление a на b битов влево, добавляя справа нули.
- Правый сдвиг с переносом знака - a >> b - Сдвигает двоичное представление a на b битов вправо, отбрасывая сдвигаемые биты.
- Правый сдвиг с заполнением нулями - a >>> b - Сдвигает двоичное представление a на b битов вправо, отбрасывая сдвигаемые биты и добавляя нули слева.

Побитовые операторы работают следующим образом:

1. Операнды преобразуются в 32-битные целые числа, представленные последовательностью битов. Дробная часть, если она есть, отбрасывается.
2. Для бинарных операторов – каждый бит в первом операнде рассматривается вместе с соответствующим битом второго операнда: первый бит с первым, второй со вторым и т.п. Оператор применяется к каждой паре бит, давая соответствующий бит результата.
3. Получившаяся в результате последовательность бит интерпретируется как обычное число.

39. Операторы сравнения в JavaScript

Многие операторы сравнения известны нам из математики.

В JavaScript они записываются так:

- Больше/меньше: a > b, a < b.
- Больше/меньше или равно: a >= b, a <= b.
- Равно: a == b. Обратите внимание, для сравнения используется двойной знак равенства ==. Один знак равенства a = b означал бы присваивание.
- Не равно. В математике обозначается символом ≠, но в JavaScript записывается как a != b.

В этом разделе мы больше узнаем про то, какие бывают сравнения, как язык с ними работает и к каким неожиданностям мы должны быть готовы.

В конце вы найдёте хороший рецепт того, как избегать «причуд» сравнения в JavaScript.

40. Сравнение строк в JavaScript

Алгоритм сравнения двух строк довольно прост:

1. Сначала сравниваются первые символы строк.

2. Если первый символ первой строки больше (меньше), чем первый символ второй, то первая строка больше (меньше) второй. Сравнение завершено.
3. Если первые символы равны, то таким же образом сравниваются уже вторые символы строк.
4. Сравнение продолжается, пока не закончится одна из строк.
5. Если обе строки заканчиваются одновременно, то они равны. Иначе, большей считается более длинная строка.

В примерах выше сравнение 'Я' > 'А' завершится на первом шаге, тогда как строки 'Коты' и 'Кода' будут сравниваться посимвольно:

1. К равна К.
2. о равна о.
3. т больше, чем д. На этом сравнение заканчивается. Первая строка больше.

41. Сравнение разных типов в JavaScript

При сравнении значений разных типов JavaScript приводит каждое из них к числу.

alert('2' > 1); // true, строка '2' становится числом 2
alert('01' == 1); // true, строка '01' становится числом 1
Логическое значение true становится 1, а false – 0.

42. Строгое сравнение в JavaScript

Использование обычного сравнения == может вызывать проблемы. Например, оно не отличает 0 от false. Это происходит из-за того, что операнды разных типов преобразуются оператором == к числу. В итоге, и пустая строка, и false становятся нулём.

Оператор строгого равенства === проверяет равенство без приведения типов.

Другими словами, если a и b имеют разные типы, то проверка a === b немедленно возвращает false без попытки их преобразования.

43. Сравнение null и 0 в JavaScript

Причина в том, что нестрогое равенство и сравнения > < >= <= работают по-разному. Сравнения преобразуют null в число, рассматривая его как 0. Поэтому выражение null >= 0 истинно, а null > 0 ложно.

С другой стороны, для нестрогого
равенства == значений undefined и null действует особое правило: эти значения ни к чему не приводятся, они равны друг другу и не равны ничему другому.
Поэтому null == 0 ложно.

44. Условные операторы: if и '?' в JavaScript

Инструкция if(...) вычисляет условие в скобках и, если результат true, то выполняет блок кода.

Оператор представлен знаком вопроса ?. Его также называют «тернарный», так как этот оператор, единственный в своём роде, имеет три аргумента.

Синтаксис: *let result = условие ? значение1 : значение2;*

Сначала вычисляется условие: если оно истинно, тогда возвращается значение1, в противном случае – значение2.

45. Преобразование значений к логическому типу

Происходит в логических операциях. Может быть вызвано с помощью `Boolean(value)`.

Подчиняется правилам:

Значение	**Становится…**
0, null, undefined, NaN, ""	false
любое другое значение	true

Пример:

```
alert( Boolean("Привет!")); // true

alert( Boolean("") ); // false
```

46. Использование цикла «do…while» в JavaScript для перебора элементов внутри сетки

47. Локальные и внешние переменные в функциях JavaScript

Глобальная переменная - это такая, которая объявлена вне тела какой-либо функции. Все глобальные переменные являются свойствами глобального объекта (в браузере – это window).

```
var car = 'Audi';

console.log(window.car); // "Audi"
```

Кроме этого если переменную в функции не объявить, а сразу ей присвоить значение, то она тоже будет глобальной. Но это только справедливо не для строго режима. В строгом режиме необходимо обязательно объявлять переменные.

Локальные переменные – это такие, которые определены с помощью ключевого слова var внутри тела какой-либо функции. Локальные переменные существуют только внутри тела функции, в которой они объявлены, а также доступны внутри её дочерних функций.

```
function getViews() {
  // локальная переменная, созданная внутри функции getViews и не доступная за её пределами
  var numberViews = 37;
  // вернём в качестве результата данной функции значение локальной переменной numberViews
  return numberViews;
}
```

48. Возврат значения функцией в JavaScript

С помощью инструкции return функция может возвратить некоторое значение (результат работы функции) программе, которая её вызвала. Возвращаемое значение передаётся в точку вызова функции.

Инструкция return имеет следующий синтаксис: return выражение;

В программу возвращается не само выражение, а результат его вычисления.

Инструкция return может быть расположена в любом месте функции. Как только будет достигнута инструкция return, функция возвращает значение и немедленно завершает своё выполнение. Код, расположенный после инструкции return, будет проигнорирован

Если инструкция return не указана или не указано возвращаемое значение, то функция вернёт значение undefined.

49. Особенности использования function expression в JavaScript

Синтаксис, который мы использовали до этого, называется *Function Declaration* (Объявление Функции):

```
function sayHi() {
  alert( "Привет" );
}
```

Существует ещё один синтаксис создания функций, который называется ***Function Expression* (Функциональное Выражение)**.

Оно выглядит вот так:

```
let sayHi = function() { // () => {
  alert( "Привет" );
};
```

В коде выше функция создаётся и явно присваивается переменной, как любое другое значение. По сути без разницы, как мы определили функцию, это просто значение, хранимое в переменной sayHi.

50. Копирование функции в другую переменную в JavaScript

51. Проверка существования свойства, оператор in в JavaScript

Оператор in возвращает true, если свойство содержится в указанном объекте или в его цепочке прототипов.

prop in *object*

prop - Строка или symbol, представляющий название свойства или индекс массива
object - Объект, в котором нужно проверить содержание свойства с заданным именем.
52. Использование цикл «for…in» в JavaScript для обхода словаря

```
for (var [key, value] of map) {
  console.log(key + " goes " + value);
}
```

53. Копирование по значению и копирование по ссылке в JavaScript

Одним из фундаментальных отличий объектов от примитивных типов данных является то, что они хранятся и копируются «по ссылке».

Примитивные типы: строки, числа, логические значения – **присваиваются и копируются «по значению».**

Объекты ведут себя иначе.

Переменная хранит не сам объект, а его «адрес в памяти», другими словами «ссылку» на него.

Сам объект хранится где-то в памяти. А в переменной user лежит «ссылка» на эту область памяти.

Когда переменная объекта копируется – копируется ссылка, сам же объект не дублируется.

54. Сравнение значений по ссылке в JavaScript ¶

Операторы равенства == и строгого равенства === для объектов работают одинаково.

Два объекта равны только в том случае, если это один и тот же объект.

В примере ниже две переменные ссылаются на один и тот же объект, поэтому они равны друг другу:

```
let a = {};let b = a; // копирование по ссылке
alert( a == b ); // true, т.к. обе переменные ссылаются на один и тот же объект
```

alert(a === b); // true
В другом примере два разных объекта не равны, хотя оба пусты:

let a = {};let b = {}; // два независимых объекта
alert(a == b); // false
Для сравнений типа obj1 > obj2 или для сравнения с примитивом obj == 5 объекты преобразуются в примитивы. Мы скоро изучим, как работают такие преобразования объектов, но, по правде говоря, сравнения такого рода необходимы очень редко и обычно являются результатом ошибки программиста.

55. Поэлементное копирование объектов в JavaScript

циклом там while, for

56. Копирование свойств объектов с использованием object.assign

Метод **Object.assign()** используется для копирования значений всех собственных перечисляемых свойств из одного или более исходных объектов в целевой объект. После копирования он возвращает целевой объект.

Object.assign(*target*, ...*sources*)

57. Перезапись свойств с использованием object.assign

Обращаю Ваше внимание, что метод **.assign()** перезаписывает значение свойства или метода в целевом объекте, если они имеют один и тот же ключ, что и в переданном источнике. Если источники имеют один и тот же ключ (свойство), то более поздние источники будут перезаписывать более ранние.

58. Доступ к вложенным объектам в JavaScript

Одни объекты могут содержать в качестве свойств другие объекты.

Для доступа к свойствам таких вложенных объектов мы можем использовать стандартную нотацию точки: country.capital.name

Либо обращаться к ним как к элементам массивов: country["capital"]["population"]

Также допустим смешанный вид обращения: country.capital["year"]

59. Использование оператора «this» в методах в JavaScript

Когда функция вызывается как метод объекта, используемое в этой функции ключевое слово this принимает значение объекта, по отношению к которому вызван метод.

60. Создание объектов через "new" в JavaScript

Оператор (операторная функция) **new** создаёт экземпляр объекта, встроенного или определённого пользователем, имеющего конструктор.

`new constructor[([arguments])]`

61. Использование типа данных Symbol

Символ (анг. Symbol) — это уникальный и неизменяемый тип данных, который может быть использован как идентификатор для свойств объектов. *Символьный объект (анг. symbol object)* — это объект-обёртка (англ. wrapper) для примитивного символьного типа.

Чтобы создать новый символьный примитив, достаточно написать Symbol(), указав по желанию строку в качестве описания этого символа:

var sym1 = Symbol();

var sym2 = Symbol("foo");

Код выше создаёт три новых символа. Заметьте, что Symbol("foo") не выполняет приведение (англ. coercion) строки "foo" к символу. Это выражение создаёт каждый раз новый символ:

```
Symbol("foo") === Symbol("foo"); // false
```

62. Понятие объектной модели браузера

Объектная модель браузера (анг. Browser Object Model (BOM)) — это часть JavaScript, которая позволяет скрипту взаимодействовать с программой просмотра веб-страниц. BOM представляет объекты, через свойства и методы которых можно управлять внешним видом и поведением обозревателя.

Например:

- Объект navigator даёт информацию о самом браузере и операционной системе. Среди множества его свойств самыми известными являются: navigator.userAgent – информация о текущем браузере, и navigator.platform – информация о платформе (может помочь в понимании того, в какой ОС открыт браузер – Windows/Linux/Mac и так далее).
- Объект location позволяет получить текущий URL и перенаправить браузер по новому адресу.

BOM является частью общей спецификации HTML.

Да, вы всё верно услышали. Спецификация HTML по адресу https://html.spec.whatwg.org не только про «язык HTML» (теги, атрибуты), она также покрывает целое множество объектов, методов и специфичных для каждого браузера расширений DOM. Это всё «HTML в широком смысле».

63. Интерпретация комментария в документной объектной модели

64. Типы узлов в документной объектной модели

Ниже таблица с кратким описанием этих типов данных.

document	Когда член возвращает объект типа document (например, свойство элемента ownerDocument возвращает документ к которому он относится), этот объект document является собственным корневым объектом. В DOM document Reference разделе описан объект document. element
element	обозначает элемент или узел типа element, возвращаемый членом DOM API. Вместо того, чтобы говорить, что метод document.createElement() возвращает ссылку на node, мы просто скажем, что этот элемент возвращает element, который просто был создан в DOM. Объекты element реализуют DOM element интерфейс и также более общий Node интерфейс. Оба интерфейса включены в эту справку. nodeList
NodeList	массив элементов, как тот, что возвращается методом Document.getElementsByTagName(). Конкретные элементы в массиве доступны по индексу двумя способами: • list.item(1) • list[1] Эти способы эквивалентны. В первом способе item() - единственный метод объекта NodeList. Последний использует обычный синтаксис массивов, чтобы получить второе значение в списке.
attribute	Когда attribute возвращается членом API (например, метод createAttribute()) - это будет ссылка на объект, который предоставляет специальный (хоть и небольшой) интерфейс для атрибутов. Атрибуты - это узлы в DOM, как и элементы, хотя вы можете редко использовать их в таком виде.
namedNodeMap	namedNodeMap подобна массиву, но элементы доступны по имени или индексу. Доступ по индексу - это лишь для удобства перечисления, т.к. элементы не имеют определённого порядка в списке. Этот тип данных имеет метод item() для этих целей и вы можете также добавлять и удалять элементы из namedNodeMap

65. Дочерние узлы, потомки в документной объектной модели

66. Коллекция childNodes в документной объектной модели

Доступный для чтения аттрибут **Node.childNodes** возвращает *коллекцию* дочерних элементов данного элемента.

```
var ndList = elementNodeReference.childNodes;
```

Элементы в коллекции -- объекты, а не строки. Чтобы получить данные из этих объектов, вы должны использовать их свойства (например, elementNodeReference.childNodes[1].nodeName чтобы получить имя, и т. д.).

DOM-коллекции, такие как childNodes и другие, которые мы увидим далее, не являются JavaScript-массивами.

childNodes также включают, например, текстовые узлы и комментарии. Чтобы пропустить их, используйте ParentNode.children (en-US) взамен.

67. Свойства firstChild и lastChild в документной объектной модели

Свойство **Node.firstChild** только для чтения, возвращающее первый потомок узла в древе или null, если узел является бездетным. Если узел это документ, он возвращает первый узел в списке своих прямых детей.

lastChild возвращает последнего потомка в узле. Возвращаемый элемент last_child является узлом. Если его родитель является элементом, то возвращаемый узел является узлом типа Элемент, Текст, или же узлом комментария. Если в опрашиваемом узле нет дочерних элементов, **lastChild** возвращает null .

68. Цикл for..of для перебора элементов DOM-коллекции

Оператор for...of выполняет цикл обхода итерируемых объектов (включая Array, Map, Set, объект аргументов и подобных), вызывая на каждом шаге итерации операторы для каждого значения из различных свойств объекта.

Синтаксис

```
for (variable of iterable) {
  statement
}
```

variable

На каждом шаге итерации *variable* присваивается значение нового свойства объекта *iterable*. Переменная *variable* может быть также объявлена с помощью const, let или var.

iterable

Объект, перечисляемые свойства которого обходятся во время выполнения цикла.

Примеры

Обход Array

```
let iterable = [10, 20, 30];


for (let value of iterable) {
  value += 1;
  console.log(value);
}
// 11
// 21
// 31
```

Можно также использовать const вместо let, если не нужно переназначать переменные внутри блока.

```
let iterable = [10, 20, 30];


for (const value of iterable) {
  console.log(value);
}
// 10
// 20
// 30
```

69. Перебор узлов DOM-коллекции свойствами nextSibling и previousSibling

Node.previousSibling

Свойство **Node.previousSibling** используется только для чтения, оно возвращает узел предшествующий указанному в родительском элементе childNodes, или null, если указанный узел первый в своём родителе.

```
// <a><b1 id="b1"/><b2
id="b2"/></a>alert(document.getElementById("b1").previousSibling); //
nullalert(document.getElementById("b2").previousSibling.id); // "b1"
```

Свойство **Node.nextSibling** используется только для чтения и возвращает узел, непосредственно следующий за данным узлом в списке childNodes его родительского элемента, или null если данный узел последний в этом списке.

Браузеры, основанные на Gecko, вставляют текстовые узлы в документ для представления пробелов в разметке. Поэтому узел, полученный, например, при использовании Node.firstChild или Node.previousSibling может относиться к пробелу, а не к тому элементу, который автор хотел получить.

Смотрите Пробел в DOM (en-US) и W3C DOM 3 FAQ: Почему некоторые текстовые узлы пустые? для дополнительной информации.

Пример

```
<div id="div-01">Вот div-01</div><div id="div-02">Вот div-02</div><script
type="text/javascript">var el = document.getElementById('div-01').nextSibling,
    i = 1;

console.log('Потомки div-01:');while (el) {
  console.log(i + '. ' + el.nodeName);
  el = el.nextSibling;
  i++;}</script>

/**********************************************
  Следующий пример напишет в консоль:

    Потомки div-01:

      1. #text
      2. DIV
      3. #text
      4. SCRIPT

**********************************************/
```

70. Поиск CSS-селекторов через метод elem.querySelectorAll(css)

querySelectorAll

Самый универсальный метод поиска – это elem.querySelectorAll(css), он возвращает все элементы внутри elem, удовлетворяющие данному CSS-селектору.

71. Сравнение CSS-селекторов через метод elem.matches(css)

Метод matches позволяет проверить, удовлетворяет ли элемент указанному CSS селектору.

Синтаксис

элемент.matches('селектор');

Пример

Проверим, является ли наш элемент абзацем с классом www:

let elem = document.querySelector('#elem');console.log(elem.matches('p.www'));

Результат выполнения кода:

true

72. Поиск ближайшего предка через метод elem.closest(css)

Метод **Element.closest()** возвращает ближайший родительский элемент (или сам элемент), который соответствует заданному CSS-селектору или null, если таковых элементов вообще нет.

Синтаксис

```
var elt = element.closest(selectors);
```

- **selectors -** строка, а точнее DOMString, содержащая CSS-селектор, к примеру: ***"#id", ".class", "div"***...
- Результат - элемент DOM (Element), либо null.

Пример

```
<div id="block" title="Я - блок">
    <a href="#">Я ссылка в никуда</a>
    <a href="http://site.ru">Я ссылка на сайт</a>
    <div>
        <div id="too"></div>
    </div>
</div>
```

```
var div = document.querySelector("#too"); //Это элемент от которого мы начнём поиск

div.closest("#block"); //Результат - самый первый блок древа выше
div.closest("div"); //Сам блок #too и будет результатом, так как он подходит под селектор "div"
div.closest("a"); //null - В предках #too нет ни одного тега "a"!
div.closest("div[title]") //#block - так как ближе нет блоков с атрибутом title.
```

73. Использование методов getElementsBy*

document.getElementById или просто id

Если у элемента есть атрибут id, то мы можем получить его вызовом document.getElementById(id), где бы он ни находился.

querySelectorAll

Самый универсальный метод поиска – это elem.querySelectorAll(css), он возвращает все элементы внутри elem, удовлетворяющие данному CSS-селектору.

querySelector

Метод elem.querySelector(css) возвращает первый элемент, соответствующий данному CSS-селектору.

getElementsBy*

Существуют также другие методы поиска элементов по тегу, классу и так далее.

На данный момент, они скорее исторические, так как querySelector более чем эффективен.

Здесь мы рассмотрим их для полноты картины, также вы можете встретить их в старом коде.

- elem.getElementsByTagName(tag) ищет элементы с данным тегом и возвращает их коллекцию. Передав "*" вместо тега, можно получить всех потомков.
- elem.getElementsByClassName(className) возвращает элементы, которые имеют данный CSS-класс.
- document.getElementsByName(name) возвращает элементы с заданным атрибутом name. Очень редко используется.

74. Основные значения свойства «nodeType»

Свойство «nodeType»

Свойство nodeType предоставляет ещё один, «старомодный» способ узнать «тип» DOM-узла.

Его значением является цифра:

- elem.nodeType == 1 для узлов-элементов,
- elem.nodeType == 3 для текстовых узлов,
- elem.nodeType == 9 для объектов документа,
- В спецификации можно посмотреть остальные значения.

75. Использование свойства innerHTML

(смореть ниже)

76. Использование свойства outerHTML

Атрибут outerHTML DOM-интерфейса element получает сериализованный HTML-фрагмент, описывающий элемент, включая его потомков. Можно установить замену элемента узлами, полученными из заданной строки.

```
// HTML:// <div id="d"><p>Content</p><p>Further Elaborated</p></div>



d = document.getElementById("d");
```

```
console.log(d.outerHTML);// строка '<div id="d"><p>Content</p><p>Further Elaborated</p></div>'// выведена в окно консоли
```

77. Использование свойства textContent

Свойство Element.**textContent** позволяет считывать или задавать текстовое содержимое элемента. Обращение к **свойству** вернёт строку, которая будет состоять из текстового содержимого всех вложенных элементов, даже если они скрыты с помощью CSS и не видны на экране. Аналогичной функциональностью, но с некоторыми ограничениями обладает **свойство** Element.innerText.

```
var text = element.textContent;

element.textContent = "Это просто текст";
```

78. Создание мигающего элемента на базе свойства «hidden»

HTMLElement Свойство **hidden** является Boolean типом данных , который принимает значение true , если содержимое спрятано, в противном случае значение будет false. Это свойство совершенно отличается от использования CSS-свойства display , чтобы контролировать отображение элемента.Свойство hidden применимо ко всем режимам представления и не должно использоваться для скрытия содержимого предназначенного для прямого доступа к пользователю. Соответствующие варианты использования включают:

```
document.getElementById("okButton")
        .addEventListener("click", function() {
  document.getElementById("welcome").hidden = true;
  document.getElementById("awesome").hidden = false;}, false);Copy to Clipboard
```

Этот код устанавливает обработчика для кнопки "OK", которая скрывает панель приветствия и делает The follow-up panel панель с необычным именем "awesome" - видимой в этом месте.

79. Изменение HTML документа через свойство innerHTML

Свойство интерфейса Element **innerHTML** устанавливает или получает HTML или XML разметку дочерних элементов.

Примечание: Если узлы , , или <noembed> (en-US) имеют дочерние текстовые узлы, содержащие символы (&), (<), или (>), innerHTML вернёт эти символы как &, &lt и &gt соответственно. Используйте Node.textContent для получения правильной копии содержимого этих текстовых узлов.

Чтобы вставить HTML в документ, не меняя содержимое элемента, используйте insertAdjacentHTML().

Value

Строка DOMString, которая содержит части HTML разметки. Установка значения innerHTML удаляет всё содержимое элемента и заменяет его на узлы, которые были разобраны как HTML, указанными в строке *htmlString.*

80. Методы для вставки узлов или строк в HTML документ

(ниже указано)

81. Использование метода insertAdjacentHTML для вставки HTML

Метод insertAdjacentHTML позволяет вставлять произвольный HTML в любое место документа, в том числе *и между узлами*!

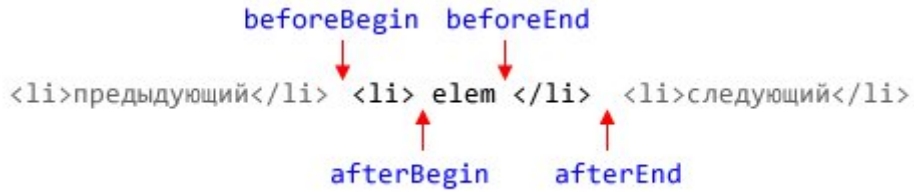
Синтаксис:

elem.insertAdjacentHTML(where, html);

html
Строка HTML, которую нужно вставить

where
Куда по отношению к elem вставлять строку. Всего четыре варианта:

1. beforeBegin – перед elem.
2. afterBegin – внутрь elem, в самое начало.
3. beforeEnd – внутрь elem, в конец.
4. afterEnd – после elem.



Например, вставим пропущенные элементы списка *перед* 5:

82. Использование методов для создания узлов

DOM-узел можно создать двумя методами:

document.createElement(tag)
Создаёт новый *элемент* с заданным тегом:

document.createTextNode(text)
Создаёт новый *текстовый узел* с заданным текстом:

let textNode = document.createTextNode('А вот и я');

83. Использование методов для вставки и удаления узлов

Вот методы для различных вариантов вставки:

- node.append(...nodes or strings) – добавляет узлы или строки в конец node,
- node.prepend(...nodes or strings) – вставляет узлы или строки в начало node,
- node.before(...nodes or strings) –- вставляет узлы или строки до node,
- node.after(...nodes or strings) –- вставляет узлы или строки после node,
- node.replaceWith(...nodes or strings) –- заменяет node заданными узлами или строками.

Для удаления узла есть методы node.remove().

84. Использование свойства className

Свойство **className** отвечает за значение атрибута **class** элемента.

```
var cName = elem.className;

elem.className = cName;
```

- ***cName*** - строка. Если нужно указать несколько классов, они указываются через пробел.

```
<div class="booble example"> </div>
```

```
var elm = document.querySelector("div");alert(elm.className); //"booble example"

elm.className = "class1 class2 class3";alert(elm.className); //"class1 class2 class3"
```

85. Использование методов classList

Свойство **classList** возвращает псевдомассив DOMTokenList, содержащий все классы элемента.

ClassList является геттером. Возвращаемый им объект имеет несколько методов:

add(String [,String])
Добавляет элементу указанные классы

remove(String [,String])

Удаляет у элемента указанные классы

item (Number)
Результат аналогичен вызову **classList[Number]**

toggle (String [, Boolean])
Если класс у элемента отсутствует - добавляет, иначе - убирает. Когда вторым параметром передано false - удаляет указанный класс, а если true - добавляет.

Если вторым параметром передан undefined или переменная с typeof == 'undefined', поведение будет аналогичным передаче только первого параметра при вызове toggle.

contains (String)
Проверяет, есть ли данный класс у элемента (вернёт true или false)

```
<div id="clock" class="example for you"> </div>Copy to Clipboard
var elem = document.querySelector("#clock")//Выведем классы
console.log(elem.classList); //DOMTokenList ["example", "for", "you"]//Добавим классы
elem.classList.add("ok", "understand");
console.log(elem.classList); //DOMTokenList ["example", "for", "you", "ok",
"understand"]//Переключим классы
elem.classList.toggle("you");
elem.classList.toggle("he");
console.log(elem.classList); //DOMTokenList ["example", "for", "ok", "understand",
"he"]//Проверим класс
console.log(elem.classList.contains("example")); //true
console.log(elem.classList.contains("lol")); //false//И удалим классы
elem.classList.remove("example", "for", "understand", "he");
console.log(elem.classList); //DOMTokenList ["ok"]
```

86. Использование elem.style для применения свойств CSS

До того, как начнёте изучать способы работы со стилями и классами в JavaScript, есть одно важное правило. Надеемся, это достаточно очевидно, но мы все равно должны об этом упомянуть.

Как правило, существует два способа задания стилей для элемента:

1. Создать класс в CSS и использовать его:
2. Писать стили непосредственно в атрибуте style: .

JavaScript может менять и классы, и свойство style.

Классы – всегда предпочтительный вариант по сравнению со style. Мы должны манипулировать свойством style только в том случае, если классы «не могут справиться».

Например, использование style является приемлемым, если мы вычисляем координаты элемента динамически и хотим установить их из JavaScript:

let top = /* сложные расчёты */;let left = /* сложные расчёты */;

elem.style.left = left; // например, '123px', значение вычисляется во время работы скрипта
elem.style.top = top; // например, '456px'

87. Обработка браузерных событий мыши и клавиатуры

(ниже вопросы)

88. Использование addEventListener

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM).

89. Обработка события onclick при нажатии на кнопку

Событие onclick возникает при щелчке левой кнопкой мыши на элементе, к которому добавлен атрибут onclick.

90. Использование основных событий мыши

Типы событий мыши

Мы можем разделить события мыши на две категории: «простые» и «комплексные».

Простые события

Самые часто используемые простые события:

mousedown/mouseup

Кнопка мыши нажата/отпущена над элементом.

mouseover/mouseout // mouseenter/mouseleave

Курсор мыши появляется над элементом и уходит с него.

mousemove

Каждое движение мыши над элементом генерирует это событие.

contextmenu

Вызывается при попытке открытия контекстного меню, как правило, нажатием правой кнопки мыши. Но, заметим, это не совсем событие мыши, оно может вызываться и специальной клавишей клавиатуры.

…Есть также несколько иных типов событий, которые мы рассмотрим позже.

Комплексные события

click

Вызывается при mousedown , а затем mouseup над одним и тем же элементом, если использовалась левая кнопка мыши.

dblclick

Вызывается двойным кликом на элементе.

Комплексные события состоят из простых, поэтому в теории мы могли бы без них обойтись. Но хорошо, что они существуют, потому что работать с ними очень удобно.

91. Получение координат мыши и обработка движения мыши

Координаты: clientX/Y, pageX/Y

Все события мыши имеют координаты двух видов:

1. Относительно окна: clientX и clientY.
2. Относительно документа: pageX и pageY.

Например, если у нас есть окно размером 500x500, и курсор мыши находится в левом верхнем углу, то значения clientX и clientY равны 0. А если мышь находится в центре окна, то значения clientX и clientY равны 250 независимо от того, в каком месте документа она находится и до какого места документ прокручен. В этом они похожи на position:fixed.

Наведите курсор мыши на поле ввода, чтобы увидеть clientX/clientY (пример находится в iframe, поэтому координаты определяются относительно этого iframe):

92. События change, input, submit для обработки нажатия на тег

Событие change срабатывает по окончании изменения элемента.

Для текстовых <input> это означает, что событие происходит при потере фокуса.

Пока мы печатаем в текстовом поле в примере ниже, событие не происходит. Но когда мы перемещаем фокус в другое место, например, нажимая на кнопку, то произойдёт событие change:

Для других элементов: select, input type=checkbox/radio событие запускается сразу после изменения значения

Событие input срабатывает каждый раз при изменении значения.

В отличие от событий клавиатуры, оно работает при любых изменениях значений, даже если они не связаны с клавиатурными действиями: вставка с помощью мыши или распознавание речи при диктовке текста.

Эти события происходят при вырезании/копировании/вставке данных.

Они относятся к классу ClipboardEvent и обеспечивают доступ к копируемым/вставляемым данным.

Мы также можем использовать event.preventDefault() для предотвращения действия по умолчанию, и в итоге ничего не скопируется/не вставится.

Например, код, приведённый ниже, предотвращает все подобные события и показывает, что мы пытаемся вырезать/копировать/вставить:

```
<input type="text" id="input"><script>
  input.oncut = input.oncopy = input.onpaste = function(event) {
    alert(event.type + ' - ' + event.clipboardData.getData('text/plain'));
    return false;
  };</script>
```

Событие: submit

Есть два основных способа отправить форму:

1. Первый – нажать кнопку <input type="submit"> или <input type="image">.
2. Второй – нажать Enter, находясь на каком-нибудь поле.

Оба действия сгенерируют событие submit на форме. Обработчик может проверить данные, и если есть ошибки, показать их и вызвать event.preventDefault(), тогда форма не будет отправлена на сервер.

В примере ниже:

1. Перейдите в текстовое поле и нажмите Enter.
2. Нажмите <input type="submit">.

93. Событие window.onload для работы после загрузки страницы

Событие загрузки возникают в конце процесса загрузки документа. В этот момент все объекты документа находятся в DOM, и все картинки, скрипты, фреймы, ссылки загружены.

Обработчик загрузки окна

94. Использование атрибутов defer и async в JavaScript

Атрибут async
Поддерживается всеми браузерами, кроме IE9-. Скрипт выполняется полностью асинхронно. То есть, при обнаружении <script async src="..."> браузер не останавливает обработку страницы, а спокойно работает дальше. Когда скрипт будет загружен – он выполнится.

Атрибут defer
Поддерживается всеми браузерами, включая самые старые IE. Скрипт также выполняется асинхронно, не заставляет ждать страницу, но есть два отличия от async.

Первое – браузер гарантирует, что относительный порядок скриптов с defer будет сохранён.

То есть, в таком коде (с async) первым сработает тот скрипт, который раньше загрузится:

<script src="1.js" async></script><script src="2.js" async></script>
А в таком коде (с defer) первым сработает всегда 1.js, а скрипт 2.js, даже если загрузился раньше, будет его ждать.

<script src="1.js" defer></script><script src="2.js" defer></script>
Поэтому атрибут defer используют в тех случаях, когда второй скрипт 2.js зависит от первого 1.js, к примеру – использует что-то, описанное первым скриптом.

Второе отличие – скрипт с defer сработает, когда весь HTML-документ будет обработан браузером.

Например, если документ достаточно большой…

<script src="async.js" async></script><script src="defer.js" defer></script>

Много много много букв
…То скрипт async.js выполнится, как только загрузится – возможно, до того, как весь документ готов. А defer.js подождёт готовности всего документа.

95. Использование window.open, window.close для перехода на новую страницу

Всплывающее окно («попап» – от англ. Popup window) – один из старейших способов показать пользователю ещё один документ.

В этой статье мы рассмотрим открытие окон и ряд тонких моментов, которые с этим связаны.

Простейший пример:

```
window.open("http://ya.ru");
```
…При запуске откроется новое окно с указанным URL.

Большинство браузеров по умолчанию создают новую вкладку вместо отдельного окна, но чуть далее мы увидим, что можно и «заказать» именно окно.

Полный синтаксис:

```
win = window.open(url, name, params)
```
Функция возвращает ссылку на объект window нового окна, либо null, если окно было заблокировано браузером.

Параметры:

url

URL для загрузки в новое окно.

name

Имя нового окна. Может быть использовано в параметре target в формах. Если позднее вызвать window.open() с тем же именем, то браузеры (кроме IE) заменяют существующее окно на новое.

params

Строка с конфигурацией для нового окна. Состоит из параметров, перечисленных через запятую. Пробелов в ней быть не должно.

window.closed

Свойство window.closed равно true, если окно закрыто. Может быть использовано, чтобы проверить, закрыл ли посетитель попап.

window.close()

Закрывает попап без предупреждений и уведомлений. Вообще, метод close() можно вызвать для любого окна, в том числе, текущего. Но если окно открыто не с помощью window.open(), то браузер может проигнорировать вызов close или запросить подтверждение.

96. Создание CSS-анимации

CSS позволяет создавать простые анимации без использования JavaScript.

JavaScript может быть использован для управления такими CSS-анимациями. Это позволяет делать более сложные анимации, используя небольшие кусочки кода.

Существует 4 свойства для описания CSS-переходов:

- transition-property – свойство перехода
- transition-duration – продолжительность перехода
- transition-timing-function – временная функция перехода
- transition-delay – задержка начала перехода

97. Свойства CSS-переходов анимации transition-duration, transition-delay

transition-duration
Определяет время происхождения перехода. Можно указать время анимирования всех свойств перехода сразу или для каждого свойства в отдельности.

transition-delay (en-US)
Определяет как много должно пройти времени, перед тем как начнётся переход.

98. Использование свойства transition-property в CSS-анимации

transition-property (en-US)
Указывает имя или имена свойств, чьи переходы должны анимироваться. Только свойства, указанные здесь, анимируются в переходах; изменение других свойств будет происходить обычным образом.

```
.animated {
  transition-property: background-color;
      transition-duration: 3s;}
```

99. Использование CSS-правила @keyframes

CSS правило @**keyframes** позволяет контролировать промежуточные этапы анимации путем создания ключевых кадров в процессе анимации. Это дает возможность точнее контролировать процесс анимации вручную, задавая для нее промежуточные шаги. Чтобы **использовать** ключевые кадры, создается **правило** @**keyframes** с произвольным именем, которое впоследствии используется в свойстве animation-name, либо в универсальном свойстве animation для добавления списка ключевых кадров в анимацию.

@keyframes animationName {from | % {css-styles} // начало циклаto | % {css-styles} // конец цикла}

100. Создание JavaScript-анимации

С помощью JavaScript-анимаций можно делать вещи, которые нельзя реализовать на CSS.

Например, движение по сложному пути с временной функцией, отличной от кривой Безье, или canvas-анимации.

Использование setInterval

Анимация реализуется через последовательность кадров, каждый из которых немного меняет HTML/CSS-свойства.

Использование requestAnimationFrame

Теперь давайте представим, что у нас есть несколько анимаций, работающих одновременно.

Если мы запустим их независимо с помощью setInterval(..., 20), тогда браузеру будет необходимо выполнять отрисовку гораздо чаще, чем раз в 20ms.

Это происходит из-за того, что каждая анимация имеет своё собственное время старта и «каждые 20 миллисекунд» для разных анимаций – разные. Интервалы не выравнены и у нас будет несколько независимых срабатываний в течение 20ms.

ЗАДАНИЯ

1. Подключить файлы CSS, JavaScript к странице сайта

```
<link rel="stylesheet" href="style.css">
<script type="text/javascript" src="script.js"></script>
```

2. Сделать выделение блока другим цветом и появление тени при наведении на него

```
<div id="block1">блок</div>
<style>
    #block1::selection {
        background: cyan;
    }
    #block1:hover {
        text-shadow: 2px 2px 2px red;
    }
</style>
```

3. Нарисуйте любую фигуру из блока используя css. С помощью flexbox расположите его по центру страницы

```
<div id="block2"></div>
<style>
    #block2 {
        background-color: red;
        width: 40px;
        height: 40px;
    }
    body {
        height: 100vh;
        display: flex;
        align-items: center;
        justify-content: center;
    }
</style>
```

4. Создать анимацию плавного исчезновения блока и плавного появления его обратно при наведении на него курсора

```
<div id="block3"></div>
   <style>
      #block3 {
         background-color: red;
         width: 40px;
         height: 40px;
         transition: opacity 0.5s ease-in-out;
      }
      #block3:hover {
         opacity: 0;
      }
   </style>
```

5. Создать вертикальное меню для сайта, которое появляется при нажатии на кнопку «Меню»

```
   <div class="dropdown">
      <button id="btn" onclick="toggleMenu()">Меню</button>
      <div id="menu">
         <a href="#">Элемент 1</a>
         <a href="#">Элемент 2</a>
      </div>
   </div>
   <script>
      function toggleMenu() {
         const menu = document.getElementById("menu");
         if (menu.style.display === "none") {
            menu.style.display = "grid";
         } else {
            menu.style.display = "none";
         }
      }
   </script>
   <style>
```

```
    #btn {
        background-color: red;
        width: 90px;
        height: 40px;
        color: white;
    }

    #menu {
        background-color: red;
        display: none;
        position: absolute;
    }
</style>
```

6. Создать несколько вариантов отображения блока текста, для разных размеров экрана использую медиа-запросы

```
<p> Текст </p>
<style>
    p {
        color: blue;
        font-size: 40px;
    }
    @media screen and (max-width: 1600px) {
        p {
            color: green;
        }
    }
    @media screen and (max-width: 600px) {
        p {
            color: red;
        }
    }
</style>
```

7. Реализовать вывод на странице текущего даты и времени используя JavaScript

```
<p id="date"></p>
<script>
    window.timerId = window.setInterval(currentTime, 500);
    function currentTime() {
        var date = new Date();

        var datetxt = date.getDate() + '-' + (date.getMonth() + 1) + '-' +
date.getFullYear();

        var clock = date.getHours() + ":" + addZero(date.getMinutes()) + ":" +
addZero(date.getSeconds());
        document.getElementById("date").innerHTML = (datetxt + " " +
clock);
    }
    function addZero(num) {
        if (num <= 9) return "0" + num;
        else return num;
    }
</script>
```

8. Реализовать страницу, на которой есть тег , в котором есть 5 цветов на выбор. При выборе цвета фон страницы менять на соответствующий цвет

```
<div>
    <button id="green" onclick="makeGreen()"> </button>
    <button id="red" onclick="makeRed()"> </button>
</div>
<script>
    function makeGreen() {
        document.body.style.backgroundColor = "green";
    }
    function makeRed() {
        document.body.style.backgroundColor = "red";
    }
</script>
<style>
    button {
```

```
        width: 40px;
        height: 40px;
    }
    #green {
        background-color: green;
    }
    #red {
        background-color: red;
    }
</style>
```

9. Реализовать увеличение размера блока при наведении на него, текст, который внутри блока, должен увеличить шрифт

```
<p> Текст </p>
<style>
p {
    font-size: 50px;
    border: 5px solid red;
    position: absolute;
    margin: 0;
}
p:hover {
    height: 500px;
    width: 500px;
    font-size: 200px;
}
</style>
```

10. Создать тег и заполнить его произвольным текстом, также создать 2 кнопки «меньше» и «больше», по нажатию на которые будет меняться шрифт текста

```
<button onclick="changeSize(1)">Увеличить</button>
<button onclick="changeSize(-1)">Уменьшить</button>
<p id="text"> Текст </p>
<style>
    p {
        font-size: 50px;
```

```
    }
  </style>
  <script>
    function changeSize(param) {
      var textElem = document.getElementById("text")

      style = window.getComputedStyle(textElem,
null).getPropertyValue('font-size');

      currentSize = parseFloat(style);

      textElem.style.fontSize = currentSize + (5 * param) + 'px';
    }
  </script>
```

11. Создать header сайта с привязкой к верхней части экрана

```
 <header>
      <h1>Header</h1>
  </header>
<style>
header{
  position:fixed;
  top:0;
  width:100%;
  height:auto;
  background:green;
}
h1{
  text-align: center;
  color:white;
}
</style>
```

12. Создать footer сайта с привязкой к нижней части экрана

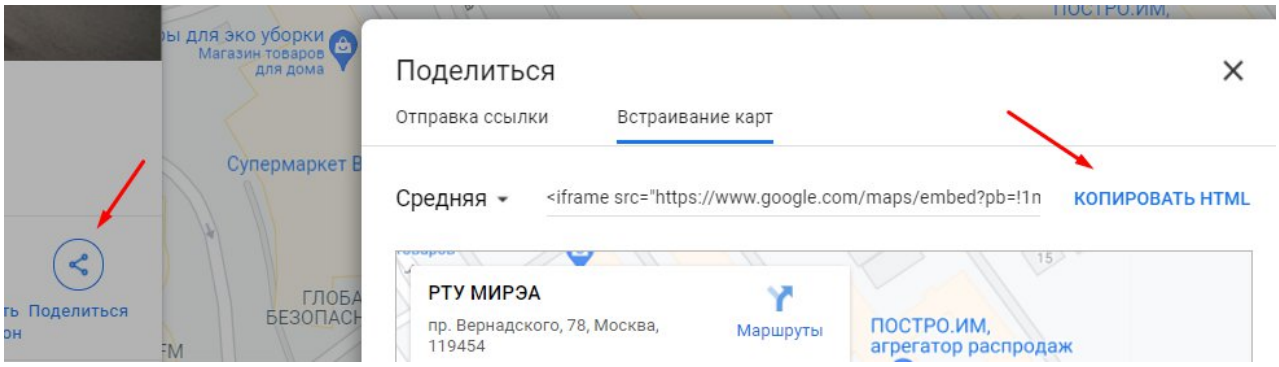
```
<footer>
      <h1>Footer</h1>
```

```
    </footer>
<style>
footer{
    position:fixed;
    bottom:0;
    width:100%;
    height:auto;
    background:green;
}
h1{
    text-align: center;
    color:white;
}
</style>
```

13. Создать навигационную панель для сайта используйте flexbox для позиционирования

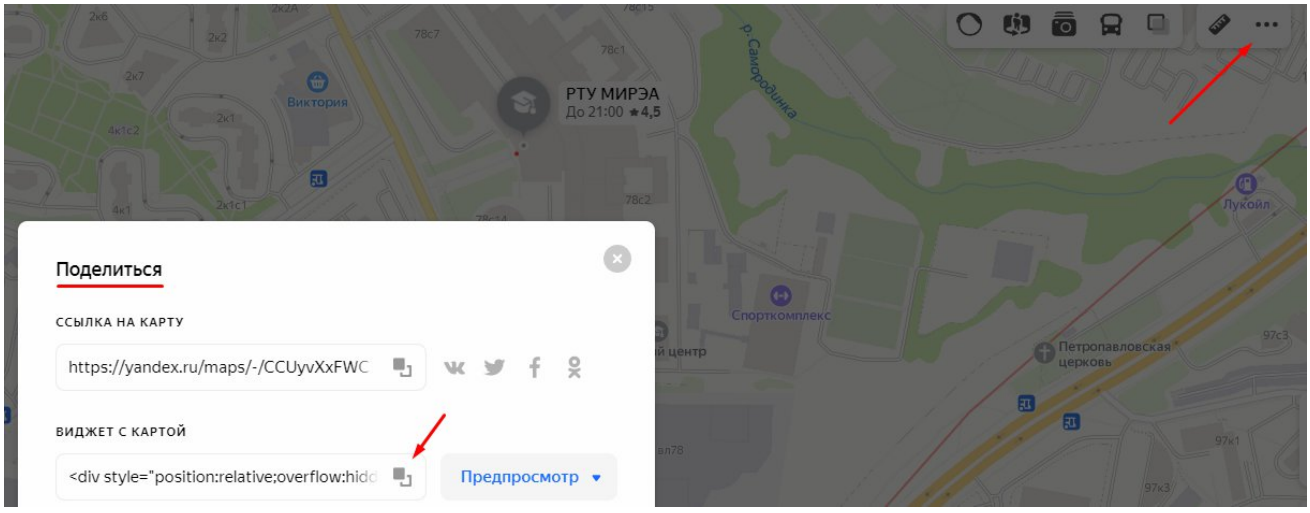
```
<head>
    <style>
        #a {
            position: fixed;
            top: 0;
            width: 100%;
            height: 50px;
            background-color: red;
            display: flex;
            flex-direction: row;
            justify-content: space-between;
        }
        #c {
            width: 100%;
            height: 1024px;
            background-color: grey;
        }
    </style>
</head>
<body>
    <div id="a">
        <button id="b">Smth</button>
    </div>
    <div id="c"></div>

</body>
```

14. Встроить карту Google карту на страницу своего сайта




```
<iframe
src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d1637
.7019516137052!2d37.47946427263383!3d55.66939887980949!2m3!1f0!2
f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x46b54dc1d23b51c3%3A0x7
4763ed59c81ccb6!2z0KDQotCjINCc0JjQoNCt0JA!5e0!3m2!1sru!2sru!4v1642
603665932!5m2!1sru!2sru" width="600" height="450" style="border:0;"
allowfullscreen="" loading="lazy"></iframe>
```

15. Встроить карту Яндекс карту на страницу своего сайта




```
<div style="position:relative;overflow:hidden;"><a
href="https://yandex.ru/maps/org/rtu_mirea/1084832794/?utm_medium=
mapframe&utm_source=maps" style="color:#eee;font-
size:12px;position:absolute;top:0px;">РТУ МИРЭА</a><a
href="https://yandex.ru/maps/213/moscow/category/university_college/1
84106140/?utm_medium=mapframe&utm_source=maps"
style="color:#eee;font-size:12px;position:absolute;top:14px;">ВУЗ в
Москве</a><iframe src="https://yandex.ru/map-widget/v1/-
```

```
/CCUyvXXdSB" width="560" height="400" frameborder="1" allowfullscreen="true" style="position:relative;"></iframe></div>
```

16. Установить изображение на фон сайта и сделать фавикон гифкой

1. `<body style="background-image: url('__IMAGE__')">`
2. While it's currently only supported by Firefox other browsers will hopefully support it i n the future. To achieve the effect, you need to upload the gif to your server and then add the line below to head section of your page:

   ```
   <link rel="icon" href="animated_favicon.gif" type="image/gif" >
   ```

3. In other browsers you can animate a favicon using JavaScript. You just have to extract single frames from the GIF and change <link rel="favicon"> src every time the GIF frame changes. Requires additional gif-processing library - not an option.

17. Изменить внешний вид скроллбара и внешний вид курсора

Для Chrome/Edge/Safari

```
body::-webkit-scrollbar {
  width: 12px;              /* width of the entire scrollbar */
}

body::-webkit-scrollbar-track {
  background: orange;       /* color of the tracking area */
}

body::-webkit-scrollbar-thumb {
  background-color: blue;    /* color of the scroll thumb */
  border-radius: 20px;       /* roundness of the scroll thumb */
  border: 3px solid orange;  /* creates padding around scroll thumb */
}
```

18. Создать анимацию, при нажатии на картинку она должна увеличиваться в размере, когда размер дойдёт до максимального (по вашему выбору), она начнёт уменьшаться

```
<head>
    <style>
        #a {
            width: 100px;
            height: 100px;
```

```
            object-fit: cover;
            transition: all 0.5s;
        }
    </style>
</head>
<body>
    <img id="a" src="__IMAGE__" alt=""/>
    <script>
        const a = document.getElementById('a'),
              c = 3 // max_step
        let b = 1 // step
        a.addEventListener('click', () => {
            if (b > 0 && b < c)
                b++
            else if (b < 0 && b > -c)
                b--
            else if (b === c)
                b = -1
            else
                b = 1

            a.style.width =
                a.style.height = `${b * 10 + a.clientWidth}px`
        })
    </script>

</body>
```

19. Создать кнопку «Смена» и блок, который при нажатии становится поочередно то невидимым, то снова видимым. При этом позиция, которую он занимает, не освобождается

```
<head>
    <style>
        #b {
            width: 100px;
            height: 100px;
            background-color: red;
        }
    </style>
</head>
<body>
    <button id="a">Switch</button>
    <div id="b"></div>
    <script>
        const a = document.getElementById('a'), // button
              b = document.getElementById('b') // block
        let c = false // flag

        a.addEventListener('click', () => {
```

```
            b.style.visibility = c ? 'visible' : 'hidden'
            c = !c
        })
    </script>

</body>
```

20. Создать сетку из 7 блоков разного размера и расположить их используя flexbox

NOTICE: there several elements with the SAME id are generated, and each of them obeys to the second CSS rule. So there MIGHT be SEVERAL elements with the SAME id, but the only way to do that is to clone the original element. Interesting, isn't it?

```
<head>
    <style>
        #a { /*container*/
            display: flex;
            flex-direction: row;
            justify-content: space-between;
            align-items: center;
        }
        #b { /*.b element template*/
            margin: 5px;
        }
    </style>
</head>
<body>
    <div id="a">
        <div id="b"></div><!- class="b" –>
    </div>
    <script>
        const a = document.getElementById('a'); let b
        for (let i = 0; i < 7; i++) {
            b = i > 0 ? b.cloneNode(true) : a.querySelector('div')

            const c = (Math.random() * 100).toString() + 'px'
            b.style.width = b.style.height = c
            b.style.backgroundColor = b.style.backgroundColor = 'red'

            i > 0 ? a.append(b) : undefined // noinspect
        }
    </script>

</body>
```

21. Взять изображение и разметить на нём область — квадрат. К области добавить возможность перехода на внешние сайты по смыслу самого изображения

```
<img
```

```
    src="__IMAGE__"
    usemap="#map"
    alt="Usermap">
<map name="map">
    <area
        shape="poly"
        coords="74,0,113,29,98,72,52,72,38,27"
        href="__HREF__"
        alt="Another page"
        target="_blank">
    <area
        shape="rect"
        coords="22,83,126,125"
        alt="Another page"
        href="__HREF__"
        target="_blank">
    <area
        shape="circle"
        coords="73,168,32"
        alt="Another page"
        href="__HREF__"
        target="_blank">

</map>
```

22. Вывести свои ФИО и рядом нарисовать окружность используя <canvas>/<svg>

```
<body>
    <canvas id="cns" width="200px" height="200px"></canvas>
    <script>
        const cns = document.getElementById('cns'),
              ctx = cns.getContext('2d')

        ctx.beginPath()
        ctx.arc(50, 50, 40, 0, 2 * Math.PI)
        ctx.fill()
        ctx.closePath()
```

```
        ctx.font = "30px Arial"
        ctx.fillText("__NAME__", 0, 125)
    </script>

</body>
```

23. Реализовать два блока, накладывающихся друг на друга так, чтобы содержимое одного из них перекрывалось содержимым другого

```
<head>
    <style>
        :root {
            --width: 100px;
            --height: var(--width);
            --pos: absolute;
        }
        #a {
            width: var(--width);
            height: var(--height);
            background-color: red;
            position: var(--pos);
            top: 0;
            left: 0;
        }
        #b {
            width: var(--width);
            height: var(--height);
            background-color: blue;
            position: var(--pos);
            top: 50px;
            left: 50px;
        }
    </style>
</head>
<body>
    <div id="a"></div>
    <div id="b"></div>

</body>
```

УСЛОЖНЕННЫЕ ЗАДАНИЯ

1. Создать обработчик событий, для нажатия на блок с помощью мыши. После каждого нажатия фон блока меняется на случайный

```
<head>
```

```
    <style>
        #a {
            width: 100px;
            height: 100px;
            background-color: grey;
        }
    </style>
</head>
<body>
    <div id="a"></div>
    <script>
        const b = [
            null, // noinspect
            "grey",
            "red",
            "blue",
            "yellow",
            "green",
            "orange"
        ],
              a = document.getElementById('a')

        a.addEventListener('click', () =>
            a.style.backgroundColor =
                b[Math.floor(Math.random() * 6)]
        )
    </script>

</body>
```

Вариант с реально рандомным цветом // pseudo-random

```
  <style>
    #a {
      width: 100px;
      height: 100px;
      background-color: grey;
    }
  </style>
</head>
<body>
  <div id="a"></div>
  <script>
    a = document.getElementById('a')
    a.addEventListener('click', () =>
      a.style.backgroundColor =
'#'+Math.floor(Math.random()*255*255*255).toString(16)
    )
  </script>
</body>
```

2. На странице создайте таблицу 5x5, используя тег , заполнив их случайными числами, добавим кнопку «Сортировка» по нажатию на которую все числа будут сортироваться в порядке возрастания

```
<!--

sorts every row’s content i.e.

8 6 1 9 0

9 0 2 1 0

...

becames

0 1 6 8 9

0 0 1 2 9

...

but doesn’t sort rows themselfes

By the way do not ever use the following construction

smth ?

    smth2 :

    smth3

in production, it’s bad. Consider use of if-else operators.

-->

    <table id="tbl">
        <tr>
             <!--noinspect-->
            <th>__TEMPLATE__</th>
        </tr>
    </table>
    <button id="bt">Sort</button>
    <script>
        // initialization
        const tbl = document.getElementById('tbl'),
              ln = 5, // it’s not a natural logarithm, just the LeNgth
              bt = document.getElementById('bt')
        let tr = tbl.querySelector('tr'),
            th = tr.querySelector('th')

        // generate and fill the table
        for (let i = 0; i < ln; i++) {
```

```
            tr = i > 0 ? tr.cloneNode(false) : tr

            for (let j = 0; j < ln; j++) {
                th = j > 0 || i > 0 ? th.cloneNode(true) : th
                th.textContent =
                     Math.floor(Math.random() * 100).toString()
                j > 0 || i > 0 ?
                     tr.append(th) :
                     undefined // noinspect
            }

            i > 0 ? tbl.append(tr) : undefined // noinspect
            tr = tbl.querySelector('tr')
            th = tr.querySelector('th')
        }

        // sorting (buble sort)
        function srt(ar /*array*/, ac /*predicate*/) {
            for (let i = 0; i < ar.length; i++){
                for (let j = 0; j < ar.length - i - 1; j++){
                    if (ac(ar[j], ar[j + 1])) {
                        const t = ar[j]
                        ar[j] = ar[j + 1]
                        ar[j + 1] = t
                    }
                }
            }
        }

        // button event handling
        bt.addEventListener('click', () => {
            let rs = tbl.querySelectorAll('tr') // querying rows
            for (let i = 0; i < ln; i++) {

                // querying next row's cells
                const cs = rs[i].querySelectorAll('th')
                // creating buffer array for storing cells' numbers
                let ns = new Array(ln)

                // converting each cell to number
                for (let j = 0; j < ln; j++)
                    ns[j] = parseInt(cs[j].textContent)

                // sorting them
                srt(ns, (a, b) => a > b)

                // replacing unsorted cells with sorted ones in each
row
                for (let j = 0; j < ln; j++)
                    cs[j].textContent = ns[j]
```

```
            }
        })
    </script>
```

3. С помощью тега и любой картинки, сделать изменение размера изображения, с помощью перемещения «ползунка»

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Ползунок</title>
   <script>
  function sizePic() {
   size = document.getElementById("size").value;
   img = document.getElementById("pic");
   img.width = 60 + 20*size;
  }
  </script>
 </head>
 <body>
  <p>Размер рисунка: <input type="range" min="10" max="50"
id="size"
    oninput="sizePic()" value="3"></p>
  <p><img src="/pict.jpg" alt="" id="pic"></p>
 </body>
</html>
```

4. Создать форму ввода логина и пароля, добавьте проверку, чтобы длина пароля была не менее 8 и не более 16 символов. В случае, если пароль не соответствует требованиям, поле должно очищаться и поменять цвет фона для поля на красный

```
    <input id="a" type="email">
    <input id="b" type="password">
    <button id="c">Authorise (or Authenticate, which is
right?)</button>
    <script>
        const a = document.getElementById('a'),
              b = document.getElementById('b'),
              c = document.getElementById('c')

        c.addEventListener('click', () => {
```

```
            const d = b.value.length
            if (d < 8 || d > 16) {
                b.style.backgroundColor = 'red'
                b.value = ''
                setTimeout(() =>
                    b.style.backgroundColor = 'initial', 2000)
            }
        })
    </script>
```

5. Реализовать таймер, с возможностью выбора времени с помощью тега и кнопки «Пуск», по истечении времени вывести уведомление «Время истекло»

```
<!— time format: hh:mm i.e. 01:30 is 1 hour and 30 seconds -->
<!-- will countdown in this way: 01:29, 01:28..., 01:00..., 00:00 -->
    <h1></h1>
    <input><!--noinspect-->
    <button>Start</button>
    <script>
        const $0x1 = document
                .getElementsByTagName('body')[0]
                .querySelectorAll('*'),
              $0x2 = $0x1[0],
              $0x3 = $0x1[1],
              $0x4 = $0x1[2]
        let $0x11 = undefined

        $0x4.addEventListener('click', () => {
            const
                $0x6 = new Date(),
                $0x7 = $0x3.value

            $0x6.setHours($0x6.getHours() + parseInt($0x7.substr(0, 2)))
            $0x6.setMinutes(
                 $0x6.getMinutes() + parseInt($0x7.substr(3, 2)))

            clearInterval($0x11)
            $0x11 = setInterval(() => {
                let $0x5 = new Date()
                const
                    $0x8 = new Date($0x6.getTime() - $0x5.getTime() +
                        new Date().getTimezoneOffset() * 60 * 1000),
                    $0x9 = $0x8.getHours(),
                    $0x10 = $0x8.getMinutes(),
                    $0x12 = $0x8.getSeconds()

                $0x2.textContent = $0x9 + ' : ' + $0x10 + ' : ' + $0x12

                if ($0x9 === 0 && $0x10 === 0 && $0x12 === 0) {
                    clearInterval($0x11)
```

```
                    $0x2.textContent = 'Countdown is over'
                }
            }, 1000)
        })
    </script>
```

6. Создать две кнопки (min и max) и полосу загрузки (тег). При нажатии на кнопку max срабатывает функция, которая плавно заполняет полосу загрузки до максимума. При нажатии на кнопку min срабатывает функция, которая плавно убирает полосу загрузки до минимума

```
    <button>Min</button>
    <button>Max</button>
    <progress value="0" max="100"></progress>
    <script>
        const _a = document.querySelectorAll('body > *'),
            a = _a[0], b = _a[1], c = _a[2]

        function a_(b_) {
            const d = c.getAttributeNode('value')
            d.value = b_.toString()
            c.setAttributeNode(d)

// there's gotta be another way to change it, cuz this isn't
// convinient. 3 lines only to change one thing. That’s why it isn’t
// widely used and is emulated with divs
        }

// can be replaced with ‘generator-function’ i.e. function* b_(...)
// - js feature the following code perfectly emulates
        function b_(b_) {
            if (e === (b_ ? 100 : 0))
            { clearInterval(f); return }

            // with replacement e becames ‘yeild’

            b_ ? e += 10 : e -= 10
            a_(e)
        }

        function c_(_a_) {
            clearInterval(f)
            f = setInterval(() => b_(_a_), 100)
        }

        var e = 0, f
        a.addEventListener('click', () => c_(false))
        b.addEventListener('click', () => c_(true))
    </script>
```

7. Создать тег который содержит текст «Нажмите», с помощью JavaScript, обработать нажатие на текст, после каждого нажатия цвет текста должен меняться на случайный

```
<body>
    <h1>Click</h1>
    <script>
        const a = document.querySelector('h1')
        a.addEventListener('click', () =>
            a.style.color =
                '#' + Math.floor(
                    Math.random() * (1 << 8 - 1) ** 3)
                                // ((2^8=256)-1)^3
                .toString(16))
    </script>

</body>
```

8. Написать простой калькулятор, где 2 формы для ввода первого и второго чисел, «checkbox», кнопка «Вычислить» для выполнения операции сложения двух чисел и всплывающее уведомление — результат сложения

```
    <input type="number">
    <input type="number">
    <input type="checkbox">
    <input type="submit">
    <!--noinspect^-->
    <script>
        const a = document.querySelectorAll('input'),
            b = a[0], c = a[1], e = a[2], f = a[3]
        f.addEventListener('click', () =>
            alert(parseFloat(b.value) + parseFloat(c.value)))
    </script>
```

9. Создать тест из 5 вопросов и поля для ввода ответа, кнопку «Проверить» и поле для вывода процента правильных ответов (неверные ответы должны быть отмечены)

10. Реализовать функцию, которая каждые 5 секунд будет менять цвет фона на случайный

```
<head>
    <script>
        window.onload = () =>
            setInterval(() =>
                document
                    .getElementsByTagName('body')[0]
                    .style.backgroundColor
```

```
                    = '#' + Math.floor(
                        Math.random() * (1 << 8 - 1) ** 3
                    ).toString(16), 5000)
    </script>

</head>
```

11. Создать поле для ввода символов с ограничением в 16 символов (потом он не должен вносить символы)

Подвоха нет?

```
 <style>
    input{
       width: 150px;
    }
  </style>
</head>
<body>
  <input type="text" maxlength="16"">
</body>
```

12. Создать генератор капчи из 6 символов. Текст нельзя выделять, а фон за ним трудно читаемый

```
    <img
         src="__IMAGE__"
         style="
            position: absolute;
            left: 0;
            width: 100px;
            height: 100px;
            object-fit: contain">
    <canvas
        style="
            width: 100px;
            height: 100px;
            border: 2px solid black;
            position: absolute;
            left: 0">
    </canvas>
    <script>
        const a = document.querySelector('canvas'),
            b = a.getContext('2d')

        let c = 'a' /*a-z*/, d = ''
        for (let i = 0; i < 25; i++)
            c += String.fromCharCode(c.charCodeAt(0) + i)
```

```
        for (let i = 0; i < 6; i++)
            d += c[Math.floor(Math.random() * 25)]

        b.font = "30px Arial"
        b.fillText(d, 75, 75)
    </script>
```

13. С помощью псевдоэлементов и ключевых кадров создать анимацию заполнения цветом слова "Loading…"

Вот. Чем не решение?

https://disk.yandex.ru/d/JE9cnEyqtL4SHQ

14. Создать анимацию мячика, летающего в разные стороны. В анимации использовать минимум 3 типа проигрывания (ease, ease-in, ease-out, linear и т.д.), и обязательным условием - сделать такой же мячик, у которого идёт задержка анимации

https://mnogoblog.ru/letayushhie-shariki-dlya-sajta-s-pomoshhyu-css

15. Создать объект произвольной формы и заставить его следовать за курсором.

https://codepen.io/wiez/pen/JjrqwQv

16. Сделать упрощённую версию часов - круг с секундной стрелкой. Добавить анимацию вращения стрелки как в часах.

https://codepen.io/wiez/pen/KKXLJKN

17. Создать блок с нестандартными краями. Ограничение области видимости для создания необычной формы реализовать при помощи clip-path.

```
  <style>
    div{
      background-color:rgb(0, 128, 0);
      width: 50%;
      height: 25%;

      clip-path: polygon(0% 0%, 100% 0%, 100% 75%, 75% 75%, 75% 100%, 50%
75%, 0% 75%);
    }
  </style>
</head>
<body>
```

```
  <div></div>
</body>
```

18. Создать минимум 5 блоков с одинаковым классом и стилизовать каждый из них при помощи псевдокласса :nth-child().

```
<head>
    <style>
        div {
            width: 100px;
            height: 100px;
            background-color: red;
            margin: 5px;
        }
        div:first-child
        { background-color: deepskyblue; }
        div:nth-child(2)
        { background-color: grey; }
        div:nth-child(3)
        { background-color: yellow; }
        div:nth-child(4)
        { background-color: orange; }
        div:last-child
        { background-color: cyan; }
    </style>
</head>
<body>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>

</body>
```

19. При помощи CSS счётчика и псевдоэлементов пронумеровать элементы в HTML документе (в качестве элементов можно использовать div, заголовки h1-h6 и т.д.).

Проверьте!!! Не факт, что правильно понял условие.

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
```

```
<style type="text/css">
 body {
    counter-reset:number 0;
    font-size:24px;
    color:green;
 }
 body div:after {
 counter-increment: number 1;
 content: counter(number) ".";

 }
 </style>
 </head>
 <body>
 <div>Блок </div>
 <div>Блок </div>
 <div>Блок </div>
 <div>Блок </div>
 <div>Блок </div>
 </body>
 </html>
```