Памятка по пройденному материалу раздела JavaScript (Алгоритмика)

1) М3У1.Введение

- Создание переменной
- Вывод результата (console.log, alert)
- Строку в число
- DOM дерево
- querySelector (получить объект дерева)
- Изменение объекта (innerHTML, style)

2) М3У2.Функции и события

- Создание функции
- SetTimeout
- Анонимная функция
- События
- AddEventListener

3) М3У3.Управляющие конструкции

- true, false
- И, ИЛИ
- Условный оператор (if else)
- Команда для получения значения элемента из тега input
- Массив (список)
- Добавление элемента в массив (push)
- Удаление элемента из массива (splice)
- QuerySelectorAll (для элементов с одинаковым классом)
- Цикл for

4) МЗУ4. ООП

- Создание объекта
- Получение значения свойства
- Изменение, создание, удаление св-ва
- Создание объекта с методом
- Ключевое слово this
- Цикл for in для перебора элементов
- Создание класса и его экземпляра (объекта)
- Форматирование строк

5) МЗУ5. Анимация

- Свойство transition (CSS)
- Пример со сменой цвета кнопки
- Свойство transform
- Значения свойства transform
- Пример со свойством transform
- Ключевые кадры (keyframes)
- Подключение keyframes
- Библиотека anime.js
- Ключевые кадры в anime.js

6) МЗУ6. Асинхронность и BOM

- Синхронный код
- Объект Promise
- Пример с анимациями
- BOM (для создания карты см презентацию)

7) МЗУ7. Работа со строками и cookie

- Базовые операции со строками
- Получение символа и подстроки из строки (slice)
- Замена символов (replaceAll)
- Проверка наличия подстроки в строке (include)
- Строку в массив (split)
- Массив в строку (join)
- Cookie см в презентации

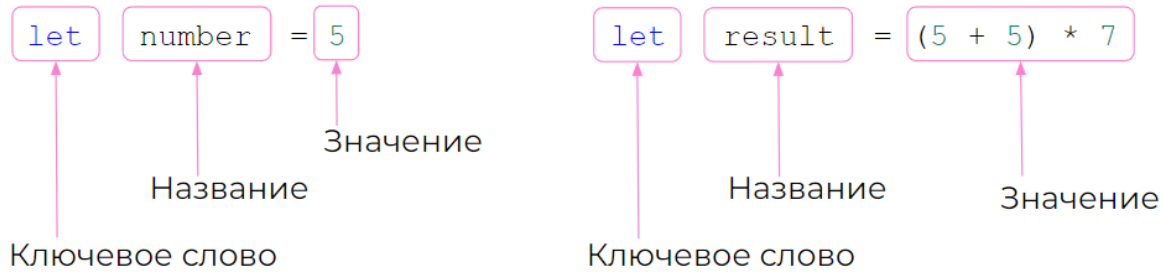
МЗУ1.Введение

Переменные

Переменные

Для создания переменной необходимо:

- Использовать ключевое слово **let**.
- Написать название переменной.
- Присвоить переменной значение.



Вывод результата

console.log()

Для вывода на экран служебной информации существует команда **console.log()**. В скобках нужно указать значение, которое мы хотим вывести. Оно окажется в консоли — там же, где вы видели результаты тестов в прошлых заданиях:

```
let a = 10
let b = 20
let result = a * b

console.log(result)
```

```
200
```

alert()

Команда alert() — ещё один способ посмотреть на данные. Работает точно также, как и console.log(), но результат оказывается не в консоли, а во **всплывающем окне**.

```
let a = 10
let b = 20
let result = a * b

alert(result)
```

Унарный плюс (строку в число)

Типы данных

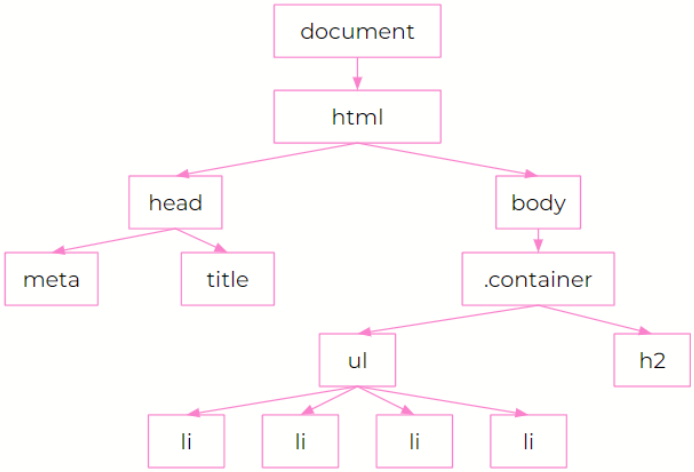
JavaScript решил всё за нас. А может, мы хотели получить число? В случаях, когда строку нужно интерпретировать как число, используется **унарный плюс**. Ставим его перед строковым значением — получается число:

```
let a = 10
let b = '4'
console.log(a + +b)
```

DOM - дерево

DOM-дерево, пример

```
<!DOCTYPE html>
<html>
<head>
   <meta charset='UTF-8'>
   <title>DOM-дерево</title>
</head>
<body>
   <div class='container'>
       <h2>Список товаров</h2>
       <ul>
           <li>Кактус комнатный</li>
           <li>Кактус уличный</li>
           <li>Маленький кактус</li>
           <li>Розовый кактус</li>
       </ul>
   </div>
</body>
</html>
```



Получение объекта в переменную (querySelector)

Поиск по селектору

Команда, которая позволяет получить объект со страницы, называется **querySelector**. Поиск нужно начинать с корневого элемента **document**:

```
let item = document.querySelector(<селектор>)
```

В результате этой команды переменная **item** будет содержать объект со страницы, найденный по заданному селектору, с которым мы сможем работать.

```
<h1>Как вам querySelector?</h1>
<p class="red">Тут красный текст</p>



let heading = document.querySelector('h1')
let item = document.querySelector('.red')
```

Изменение элементов (innerHTML, style)

Изменяем элементы

Теперь, когда переменные содержат изменяемые объекты, попробуем сделать с этими объектами что-нибудь интересное!

```
let first = document.querySelector('p')
let container =
document.querySelector('.container')
let second = container.querySelector('p')

first.innerHTML = 'Теперь тут другой текст'
second.style.color = 'red'
```

МЗУ2. Функции и события в JS

Создание функции

Функции

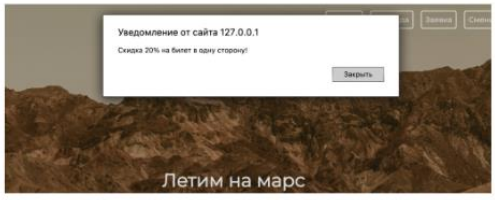
В таком случае результат можно сохранить в переменную:

```
function summa(number1, number2) {
   let result = number1 + number2
   return result
}
let sum = summa(2, 3)
```

Значение, которое возвращает функция,
после вызова окажется в переменной.

Функция SetTimeout

```
function remind() {
   alert('Скидка 20% на билет в одну сторону!')
}
setTimeout(2*60*1000, remind )
```



Анонимная функция

Анонимные функции

В таких случаях используют **анонимные функции**. Им не нужно придумывать название, т. к. вся конструкция целиком передаётся в setTimeout:

А уже из неё будет вызвана функция remind с переданным аргументом.

```
function remind(message) {
   alert(message)
}
setTimeout(2*60*1000, function(){ remind('Сообщение') } )
```

События

Событие —

это сигнал браузера о том, что пользователь совершил действие в рамках страницы. Вот список основных действий, которые распознаёт браузер:

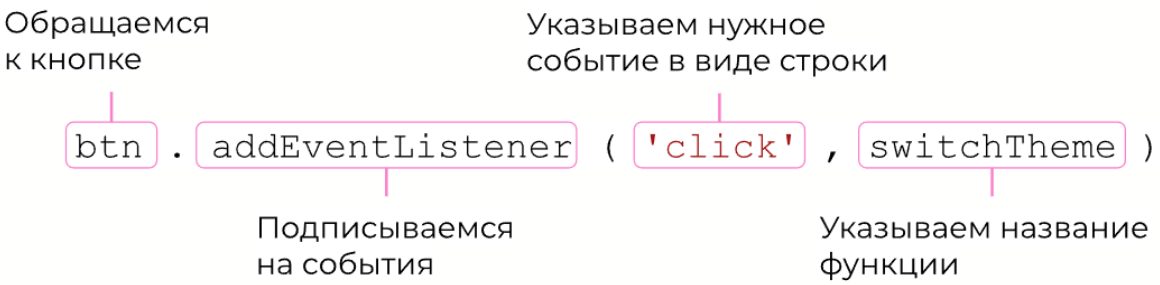
- **click** — нажатие на элемент
- **mouseenter** — перемещение мыши на блок
- **mouseleave** — перемещение мыши за пределы блока
- **mousemove** — движение мыши

[Обработка события (слушатель событий AddEventListener)]

Обработка событий

Последний шаг — запрограммировать реакцию:

```
function switchTheme() {
    let header = document.querySelector('header')
    header.style.background = 'url(dark4.jpg)'
    header.style.backgroundSize = 'cover'
}
let btn = document.querySelector('.btn-change-img')
btn.addEventListener('click', switchTheme)
```



М3У3. Управляющие конструкции

Логический тип данных (true, false)

Логический тип данных

true и **false** — значения **логического** типа данных.

Тип данных	*Целочисленный*	*Логический*
Величины	`-100, 5, 512`	`true, false`
Переменные	`let days = 31`	`let is_correct = true`
Простые выражения	`daily_money * days` `price - sale`	`5 > 2` `name != 'Иван'`

Составные выражения (ИЛИ, И)

Составные выражения

Для сложных проверок используются операторы:

- **&&** — логическое «и» — тогда оба условия должны выполниться, чтобы результатом проверки оказалось **true**;
- **||** — логическое «или» — тогда хотя бы одно из условий должно выполниться, чтобы результатом проверки оказалось **true**.

Условный оператор (if else)

```
if (Логическое выражение) {
    Выполнить блок кода
} else {
    Выполнить блок кода
}
```

Если логическое выражение **истинно** — выполнится первый блок кода. **Иначе** — второй.

Для получения данных из поля ввода (input - value)

```
let correct_pass = 'qwerty123'
let btn = document.querySelector('.btn')
let pass_field = document.querySelector('.pass-input')
let message = document.querySelector('.message')

given_pass = pass_field.value
```

Массив (список)

```
let items = [
    'Тарелка',
    'Кружка',
    'Ракета',
    'Марсоход'
    ]
```

```
console.log(items[2]) // Ракета


items[2] = 'Шаттл'
console.log(items[2]) // Шаттл
```

Добавление элемента в массив (push)

```
let new_item = 'Камень'
```

Название нового товара можно добавить в список с помощью команды **push**:

```
items .push(new_item)
```

Массив, в который добавляем.

Удаление элемента (splice)

```
items.splice( 2 , 2 )
```

Начиная с какого элемента удаляем

Сколько штук удаляем

```
let items = [
    'Тарелка',
    'Кружка',
    'Шаттл',
    'Марсоход',
    'Камень'
    ]
```

Для множества объектов с одинаковыми классами (QuerySelectorAll)

1. Список с названиями уже есть.
2. Получаем массив объектов **.merch-item-name**.
3. К каждому из объектов по очереди добавляем название.

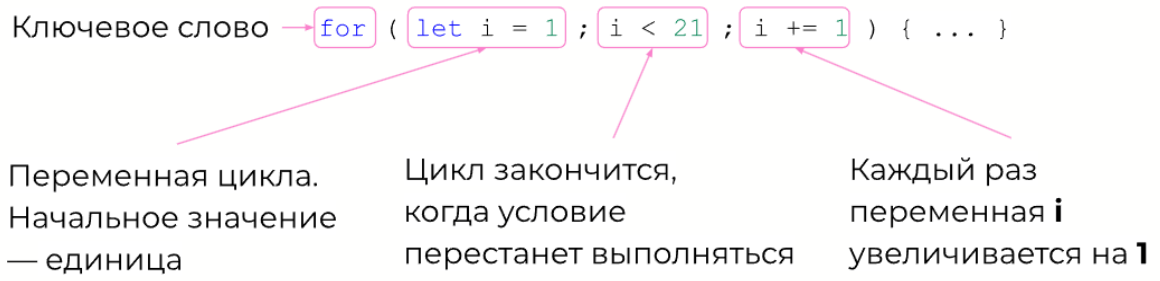
```
let items = [
    'Кружка',
    'Шаттл',
    'Марсоход',
    'Камень'
    ]
```

```
let item_objects =
document.querySelectorAll('.merch-item-name')
item_objects[0].innerHTML = items[0]
item_objects[1].innerHTML = items[1]
item_objects[2].innerHTML = items[2]
item_objects[3].innerHTML = items[3]
```

Цикл for

```
let items = [
    'Тарелка',
    'Кружка',
    'Ракета',
    'Марсоход'
    ]
```

```
for (let i = 0; i < item_objects.length; i += 1)
{
    item_objects[i].innerHTML = items[i]
}
```



М3У4.ООП

Создание объекта (перечисление свойств)

Название свойства можно записывать без кавычек, если они не содержат пробелов. Если содержат — то в кавычках:

```
let user = {
   name: 'Артем',
   surname: 'Хакимов',
   'phone number': 123456543
   ...
}
```

Получение значения свойства

Если название содержит пробелы:

```
user['phone number']
```

Если название не содержит пробелов:

```
user['phoneNumber']
```

или

```
user.phoneNumber
```

Изменение, создание, удаление свойства

Значение свойства уже созданного объекта можно менять:

```
user.id = 56
```

Можно добавить новое свойство:

```
user.login = 'algo'
```

И удалить существующее:

```
delete user.surname
```

Создание объекта с методом

greetings — это метод. Иными словами — функция, описанная для конкретного объекта. Просто так её вызвать не получится — только через точку.

```
let user = {
    name: 'Артем',
    number: 123456543,
    greetings() {
        console.log('Привет!')
    }
}
```

```
user.greetings() // Сработает

greetings() // Будет ошибка
```

Ключевое слово this (аналог self)

this

Ключевое слово **this** — способ обратиться к тому объекту, которому метод принадлежит:

```
let user = {
    name: 'Артем',
    number: 123456543,
    greetings() {
        console.log('Привет! Меня зовут ' + this.name)
    }
}
```

Цикл for in (для перебора элементов)

Цикл for(in)

```
let user = {
   name: 'Артем',
   number: 123456543,
   greetings() { ... }
}
```

```
for (let field in user) {
   console.log(field)
}
```

В консоль выведется:

```
name
number
greetings
```

Создание класса и объекта на основе класса

Конструктор

Поскольку класс — это всего лишь **описание**, то конкретные значения полям задать не получится — нужно просто уточнить, что они есть. Для этого используется **конструктор** — метод, который будет вызван при создании объекта:

```
class User {
   constructor(name, surname, id) {
       this.name = name
       this.surname = surname
       this.id = id
   }
}
```

```
let user1 = new User(
    'Иван',
    'Иванов',
    32453
)
```

Форматирование строк

```
`<div class="product"><img src="${this.pic}"><h1>${this.name}</h1>`
```

` — Символ двойной кавычки делает магию. Всё, что находится внутри этих кавычек, будет восприниматься, как текст.

${...} Всё, что находится внутри этих фигурных скобок, будет восприниматься как значение, которое нужно получить и «склеить» с основной строкой.

А ещё — при использовании таких кавычек можно переносить строку в рамках кода, оставляя её целой:

```
toHtml() {
    return `<div class="product">
                <img src="${this.pic}">
                <h1>${this.name}</h1>
                <p>${this.descr}</p>
                <p>${this.price}</p>
            </div>`
}
```

М3У5.Анимация

Свойство transition (CSS)

transition

Вот какие свойства помогут создать анимацию:

1. **transition-property** — какое свойство должно анимироваться. Если свойств несколько, перечисляем их через запятую;
2. **transition-duration** — продолжительность анимации в секундах;
3. **transition-delay** — задержка перед выполнением в секундах.

Пример со сменой цвета кнопки

```
.button {
  padding: 1em;
  background-color: rgb(78, 0, 213);
  color: white;
  border-radius: 5px;
}

.button:hover {
  transition-property: background-color;
  background-color: rgb(213, 0, 188);
  transition-duration: 0.5s;
}
```

Свойство transform (CSS)

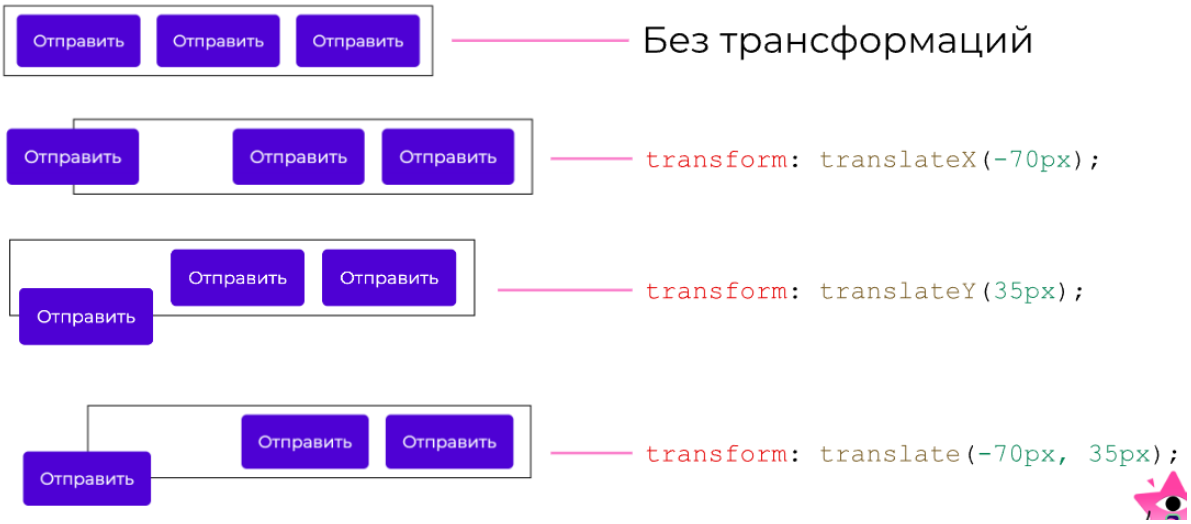
transform

В таких случаях может помочь свойство **transform**. Это свойство позволяет трансформировать элемент так, чтобы соседние элементы при этом изменений «не замечали».

[Значения свойства transform]

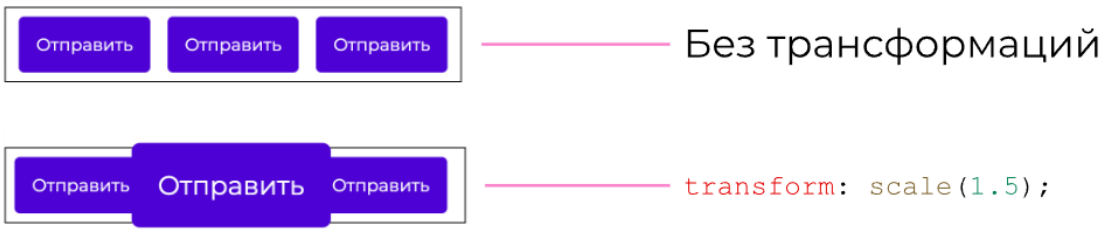
- **scale** — трансформация размера;
- **translate** — трансформация сдвига;
- **rotate** — трансформация поворота;

translate — пример



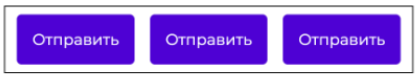
scale

Для изменения видимого размера элемента в качестве значения указываем **scale()**:



rotate

Пожалуй, самая интересная трансформация — поворот. Для этого в качестве значения нужно указать **rotate()**:



Без трансформаций


```
transform: rotate(45deg);
```

[Пример transform]

```
.button1:hover {
   transition-property:
transform;
   transition-duration: .5s;
   transform: scale(1.1)
}
```

```
.button1 {
   transition-property:
transform;
   transition-duration: .5s;
   transform: scale(0);
}
```

[Ключевые кадры (для сложных анимаций)]

Ключевые кадры

Проценты позволяют описать состояние объекта после выполнения указанной части анимации.

```
@keyframes lemon1_onload {
   0% {
      transform: none;
   } 25% {
      transform: translateX(50px);
   } 50% {
      transform: translate(50px,
50px);
   } 75% {
      transform: translateY(50px);
   }
}
```

Эта анимация продолжается 2 секунды. Значит, за 0.5 секунды (25% всего времени анимации), лимончик перейдёт из первого состояния во второе.

[Свойство animation (для подключения keyframes)]

Свойства анимации

Пока что анимация не связана с самим блоком. Это срочно нужно исправить! Для этого поможет свойство **animation** и его друзья:

- **animation-name** — название ключевых кадров;
- **animation-duration** — продолжительность анимации;
- **animation-delay** — задержка при выполнении анимации;
- **animation-iteration-count** — количество выполнений;
- **animation-fill-mode** — состояние блока после анимации;
- **animation-timing-function** — «траектория анимации».

animation-iteration-count определяет, сколько раз анимация выполнится от начала до конца. В качестве значения можно указать число или ключевое слово **infinite**, чтобы анимация выполнялась бесконечно.

animation-fill-mode — с помощью этого свойства можно указать, в каком состоянии **останется** объект по завершении анимации. Есть несколько возможных значений:

- **forwards** — тогда объект останется таким, каким был, когда анимация завершилась;
- **backwards** — тогда объект вернётся в исходное состояние.

animation-timing-function — с помощью этого свойства можно указать, как анимация будет менять свою **скорость**. Значения могут быть такими:

- **ease-in** — начинается медленно, заканчивается быстро;
- **ease-out** — начинается быстро, заканчивается медленно;
- **ease-in-out** — начинается медленно, к середине достигает пика скорости и замедляется к концу;
- **linear** — начинается и заканчивается с одинаковой скоростью.

Попробуем привязать ключевые кадры к одному из лимончиков. Помимо основных стилей, укажем:

```
.lemon1 {
    animation-name: lemon1_onload;
    animation-duration: 1s;
    animation-iteration-count: infinite;
}
```

Количество оборотов

Anime.js

Работа с библиотекой заключается в использовании функции **anime**. Эта функция принимает на вход **объект**, свойства которого — параметры анимации. Обязательно должны быть указаны свойства:

- **targets** — селекторы элементов, которые будут анимироваться;
- **duration** — продолжительность анимации в **миллисекундах**;
- **<свойство>** — CSS-свойство, которое будет меняться.
- **delay** — задержка перед выполнением анимации в **миллисекундах**;
- **easing** — то же самое, что и **animation-timing-function**;
- **loop** — будет анимация выполняться бесконечно или нет (**true**/**false**).

Пример

```
anime({
    targets: '.block',
    scale: 7,
    duration: 1000,
    delay: 1000,
    easing: 'linear',
})
```

Ключевые кадры в anime.js

Ключевые кадры

Очевидно, в anime.js также можно указывать ключевые кадры, чтобы анимация прошла через несколько состояний. Здесь ключевые кадры указываются для каждого свойства по отдельности.

Вот так:

```
anime({
 targets: '.shape',
 scale: [
   { value: 1, duration: 400},
   { value: 2, duration: 800},
   { value: 1, duration: 400},
 ],
 rotate: 360,
 loop: true,
 duration: 1600,
 easing: 'linear'
})
```

М3У6.Асинхронность и BOM

Синхронный код

Посмотрим на пример **синхронного** кода:

```
console.log('I')
console.log('Love')
console.log('Algorithmics'
)
```

```
>>> I
>>> Love
>>> Algorithmics
```

Все команды выполняются одна за другой. Второй console.log сработает только тогда, когда закончил работать первый.

Объект Promise (для выполнения двух функций по очереди и для анимаций)

Promise

Что если задача поставлена так: «Выполнить одну функцию только после того, как завершится первая». С синхронным кодом всё ясно:

```
console.log('I')
console.log('Love')
console.log('Algorithmics'
)
```

```
>>> I
>>> Love
>>> Algorithmics
```

Promise

В **result** попадает значение, переданное в resolve, поэтому в консоль выведется **done** спустя 5 секунд после запуска.

```
let promise = new Promise(function(resolve, reject)
{
    setTimeout(function() {
        resolve('done')
    }, 5000)
})

promise.then(function(result) {console.log(result)})
```

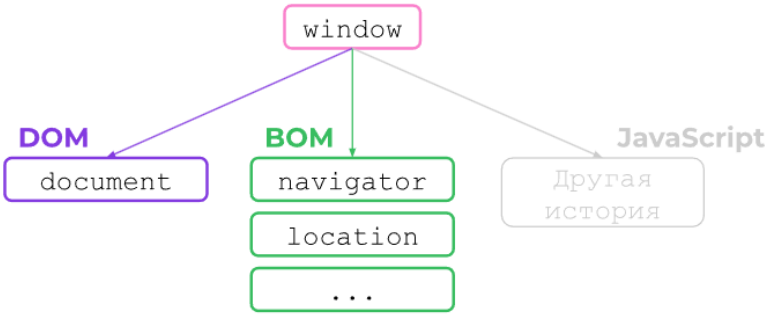
Пример с анимациями

```
anime({
    targets: '.block1',
    width: '200px',
    height: '200px',
    duration: 2000,
    easing: 'linear'
}).finished.then(function() {
    anime({
        targets: '.block2',
        width: '200px',
        height: '200px',
        duration: 2000,
        easing: 'linear'
    })
})
```

BOM (тут создавали карту, см в презентации)

BOM

Окружение в браузере кроме DOM и alert предоставляет ещё и другие объекты. Одним словом — **BOM**.
BOM — **Browser Object Model** — объектная модель **браузера**.
BOM — это тоже дерево объектов, одним из узлов которого является уже знакомый **document**. А корневой элемента дерева вообще все объектов — объект **window**.



МЗУ7. Работа со строками и cookie

Базовые операции со строками

Базовые операции

Строки можно складывать друг с другом:

```
'Привет, ' + 'Василий!' // Получится строка "Привет, Василий!"
```

Строку можно сложить с числом:

```
'abc' + 2 // Получится строка "abc2"
```

С помощью **унарного плюса** строку можно превратить в число:

```
+'327' // Получится число 327
```

Получение символа и подстроки (метод slice)

```
let name = 'Василий'
console.log(name[0]) // В консоль выведется буква "В"
```

```
                    8 … 12
let names = 'Василий;Артём;Мария;Дарья'
console.log(names.slice(8, 13)) // Выведется строка "Артём"
```

Метод slice принимает на вход два аргумента — индекс первого элемента и индекс элемента, **следующего за последним**.

Замена символов (replaceAll)

```
let names = 'Вас ил ий;Артё м;Ма рия;Да р ья'
console.log(names.replaceAll(' ', ''))
```

В результате получится строка со всеми пробелами, заменёнными на пустую строку. То есть, без пробелов.

Проверка наличия подстроки в строке

```
let names = 'Василий;Артём;Мария;Дарья'

if (names.includes('Мария')) {
   console.log('Мария есть :)')
} else {
   console.log('Марии нет :(')
}
```

Строку в массив (split)

```
let names = 'Василий;Артём;Мария;Дарья'
```

В качестве разделителя укажем **точку с запятой**:

```
console.log(names.split(';'))
```

И на выходе получим готовый массив:

```
['Василий', 'Артём', 'Мария', 'Дарья']
```

Массив в строку (join)

Существует и метод, который работает в обратную сторону. Метод **join** позволяет **объединить** элементы массива в одну строку, разделяя их указанным разделителем:

```
let names = ['Василий', 'Артём', 'Мария', 'Дарья']
console.log(names.join(';')) // Получим строку "Василий;Артём;Мария;Дарья"
```