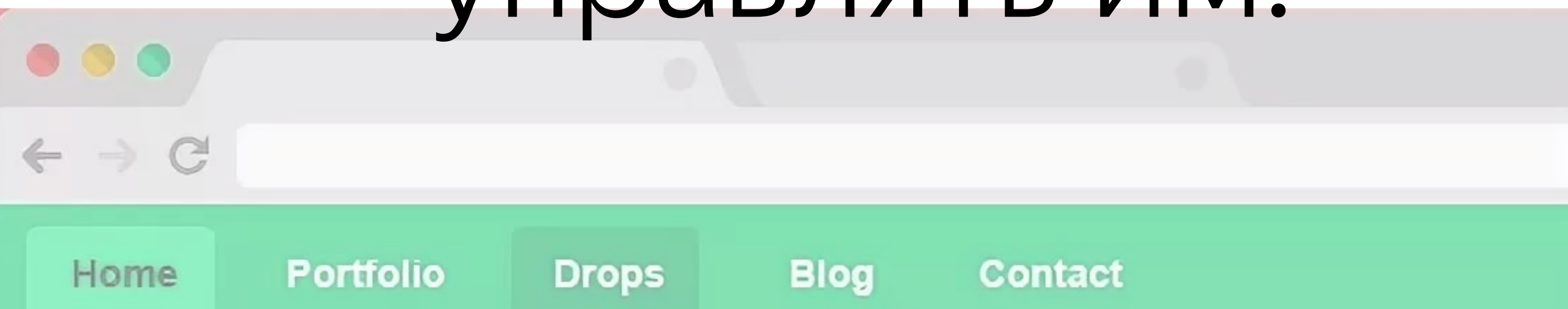



Яндекс

Яндекс

Browser Object Model

| Browser Object Model -
объектная модель браузера -
объект window, его
подобъекты и методы,
позволяющие получать
информацию от браузера и
управлять им.





ВОМ - это объект window, его методы и поля-объекты, кроме тех, что нужны для работы с документом

| window по совместительству
еще и глобальный объект



Все переменные объявленные в глобальном пространстве становятся полями `window`

`var someVariable = 'я переменная';`

`console.log(window.someVariable); // я переменная`

Модальные окна



Модальные окна

Мы уже знаем модальные окна, которые браузер показывает на методы `alert`, `confirm`, `prompt`. Так как эти «функции» - тоже поля объекта `window`, то вызовы

`alert('я модалко');`

`window.alert('я модалко');`

эквивалентны. Аналогично для `confirm` и `prompt`.

location

http://w

window.location

location - объект, позволяющий работать с адресной строкой браузера

location.href // текущий адрес открытой страницы

location.href = 'https://yandex.ru' // перейти на указанную страницу

location.reload(); // работает аналогично клику на обновление страницы, со всеми вытекающими из этого последствиями

Также, работает как поле window -> window.location

Получение значение GET параметра из JS

Очень часто данные для страницы передаются в GET параметрах, например

[https://yandex.ru/yandsearch?](https://yandex.ru/yandsearch?clid=2320519&text=%D0%BA%D1%80%D1%8B%D0%BC&lr=213)

[clid=2320519&text=%D0%BA%D1%80%D1%8B%D0%BC&lr=213.](https://yandex.ru/yandsearch?clid=2320519&text=%D0%BA%D1%80%D1%8B%D0%BC&lr=213)

Получим текст запроса

```
(location.search.slice(1).split('&').find(function(pair) {  
    return pair.indexOf('text=') === 0;  
})) || 'text=').split('=').pop();
```

Получение значение GET параметра из JS

Очень часто данные для страницы передаются в GET параметрах, например <https://yandex.ru/yandsearch?clid=2320519&text=%D0%BA%D1%80%D1%8B%D0%BC&lr=213>.

Получим текст запроса

```
decodeURIComponent (  
    (location.search.slice(1).split('&').find(function(pair) {  
        return pair.indexOf('text=') === 0;  
    }) || 'text=').split('=').pop()  
);
```

history



History

window.history

history дает нам возможность перемещать браузер по истории посещенных страниц

history.forward(); // на шаг вперед

history.back(); // на шаг назад

history.go(-2); // переместит на произвольное количество шагов вперед/назад

Работает аналогично клику по стрелкам вперед/назад в браузере

navigator



window.navigator

Чаще всего говоря про navigator на курсах для начинающих упоминают поле navigator.userAgent

navigator.userAgent содержит исчерпывающую (и самую полную) информацию о браузере клиента, его версии, версии движка и т.д.

Впрочем, начинающему такая информация обычно примерно никогда не нужна, и складывается ощущение, что navigator - скука скучная.

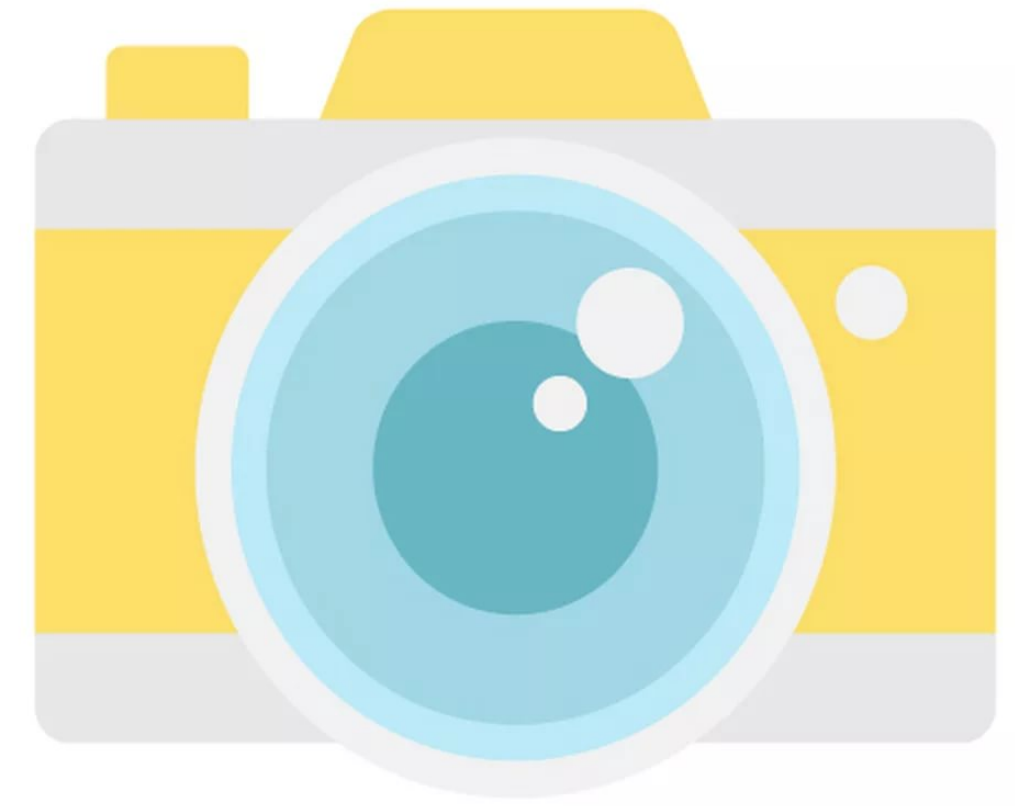
А это, между тем, самый интересный и богатый функциональностью объект из всего BOM

| navigator.geolocation





navigator.mediaDevices



Яндекс

Document Object Model



DOM - это все, что лежит в объекте `window.document`

Вспоминаем изученное

`document.getElementById('login-button')` - позволяет получить объект-узел, у которого атрибут `id` равен `login-button` (если такого тега нет, вернет `null`)

`.innerText` позволяет работать с текстом тега. Также, работает на чтение

`.value` позволяют получить значение из текстового поля. При записи в поле, мы поменяем текст в поле.

в свойства, начинающиеся на `.on...` записываем обработчики событий

На всякий случай...

Объект-узел, который мы получаем, например, с помощью `document.getElementById`, конечно, чем-то необычен, позволяет работать с документом и все такое, но это тем не менее «обычный» объект, и вполне можно:

- Передавать объекты-узлы в качестве аргументов функции

- Возвращать объект-узел в качестве результата функции

- Дописывать в объект какие-то поля

Особые поля объекта-узла



| .innerHTML



```
01 <HTML>  
02 <BODY>  
03 <HEADER>  
04 </HEADER>
```


innerHTML

innerHTML - поле, доступное на чтение и запись. При чтении браузер вычисляет строку, которая описывает содержимое тега. При записи в поле строка будет разобрана как HTML разметка, старое содержимое тега удалено, новые элементы созданы и добавлены в тег

Смотрим пример...

Использование этого поля значительно дороже использования innerText

src, href, placeholder

Поля объекта-узла, позволяющие, аналогично value, читать/перезаписывать, соответственно, адрес загруженной картинки (буквально - поменять картинку), url ссылки, текст подсказки в текстовом поле

 Смотрим пример...



Работа с атрибутами тега

Свойства `src`, `href`, `value` не равны значениям верстки

Все дело в том, что в объект узел эти свойства попадают уже в «актуальном» состоянии - относительные пути раскрываются в полный URL, `value` будет соответствовать актуальному тексту в поле ввода и т.п.

Иногда же нам нужно повзаимодействовать с актуальными значениями из верстки

Работа с атрибутами тега


`var link = document.getElementById('yandex');`

`link.getAttribute('href');` // получим значение href таким, каким оно было в верстке

`link.setAttribute('href');` // устанавливаем атрибут в верстку, свойства объекта узла также пересчитываются

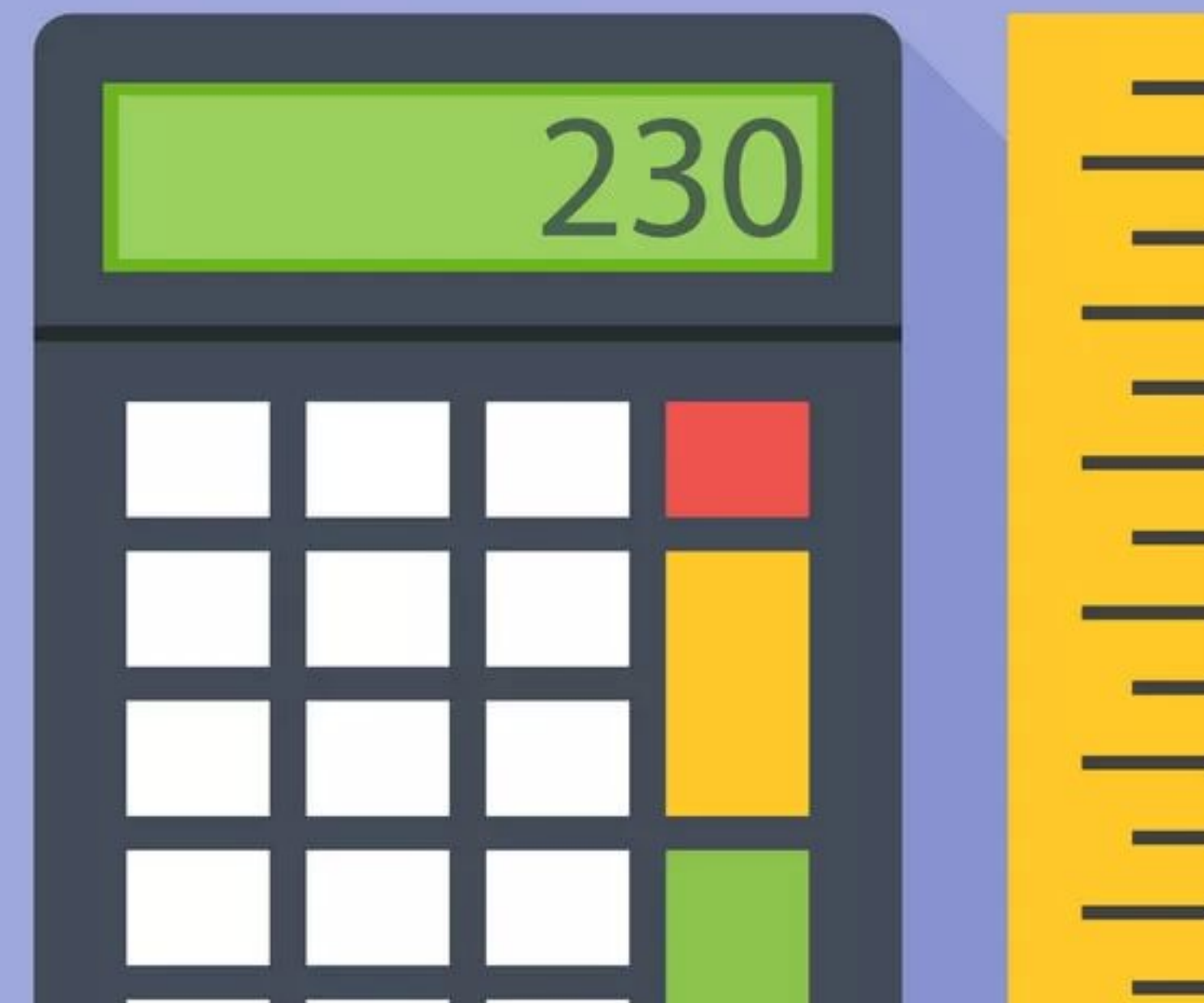
`link.removeAttribute('href');` // удаляет атрибут из верстки, со всеми последствиями

`link.hasAttribute('href');` // отвечает на вопрос, есть ли такой атрибут у тега



Можно взаимодействовать как со стандартными html атрибутами, так и создавать свои

| data-атрибуты



Свойство dataset

Помимо стандартных методов, для работы с data-атрибутами есть специальное свойство dataset, которое позволяет удобно и быстро читать-записывать данные.

link.dataset.id = 1; // значение будет преобразовано в строку

link.dataset.userId = 1; // в html такой camelCase будет автоматически вытянут в kebab-case, а при чтении будет обратное преобразование




| Работа с внешним видом узла

Свойство `style`

Свойство `.style` у объекта-узла служит для чтения/записей из атрибута `style` соответствующего тега. То есть, по сути это работа с инлайновыми стилями.

- Такие стили будут иметь приоритет над любым неинлайновым CSS

- Применятся только на одном блоке, если нужно поменять внешний вид многим блокам, потребуется всех их пробежать и установить стиль каждому



Обычно, работа с
инлайновыми стилями -
плохая идея*

*Кроме случаев, когда значение стиля вычисляется

className/classList



className

Поле className предоставляет примитивный способ считать/изменить класс элемента. Значение атрибута предоставляется в виде строки, присваивание же новой строки полностью перезапишет старое значение. Поэтому поиск/замена класса превращается в разбивку строки на массив и фильтрацию массива в соответствии с задачей.

classList

Поле `classList` было добавлено чуть позже, и представляет собой объект с массивоподобным списком классов, у которого есть методы для добавления/удаления классов и не только

`link.classList.add('error');`

`link.classList.remove('error');`

`link.classList.toggle('error');`

`link.classList.contains('error');`

Яндекс

Спасибо

Шлейко Александр

Разработчик интерфейсов



dusty@yandex-team.ru



@dustyo_O