Неделя 18

Вопросы 💎

1. Чем отличаются куки, localStorage и sessionStorage? Что стоит использовать, если нам нужно надолго сохранить много пользовательских данных?

Основным различием между Local Storage и Session Storage является время хранения данных. Local Storage, в теории, является бессрочным хранилищем данных. Но браузеры все равно вводят свои ограничения. Session Storage похож на краткосрочные Cookie, потому что данные в этом хранилище хранятся только во время жизни текущей сессии.

Cookie – это не просто хранилище данных в браузере. Данные, хранящиеся в куках, также передаются на сервер в виде HTTP-заголовка и могут быть им изменены. У хранения данных в Cookie есть много ограничений. Они передаются при каждом запросе к серверу, их размер ограничен 4096 байтами, а содержимое должно быть закодировано и быть безопасным, чтобы не сломать отправляемый запрос. Cookie не являются постоянным хранилищем, срок хранения данных по умолчанию ограничен длинной сессии, и для продления хранения кук используется дополнительный параметр. Как правило, этим способом пользуются для хранения авторизационных данных или когда доступ к записанным данным нужен на сервере. Ещё куки используются для отслеживания поведения пользователя на сайте, но браузеры активно с этим борются.

Если нужно сохранить надолго и много данных – вариант использовать Local Storage. Однако, нужно сказать, что при использовании LocalStorage, следует избегать обработки слишком больших объемов данных, так как это может привести к снижению производительности, так как его функции (`JSON.stringify()` и `JSON.parse()`) являются синхронными. Они будут блокировать выполнение JavaScript, пока не будут завершены. Кроме этого, нельзя хранить конфиденциальную информацию в LocalStorage (пароли, ключи API, токены аутентификации, финансовую информацию и пр.).

Есть еще один API для хранения данных - IndexedDB API, который обеспечивает браузер полной базой данных для хранения сложных данных. Этот способ может быть использован для хранения полных наборов записей клиентов и даже для сложных типов данных, таких как аудио или видео файлы.

2. Как добавить и получить значение из веб-хранилища?

```
// Запись в поле
window.localStorage.setItem('name', 'value')
window.sessionStorage.setItem('name', 'value')

// Чтение из поля
const nameFromLocalStorage = window.localStorage.getItem('name')
const nameFromSessionStorage = window.sessionStorage.getItem('name')
```

3. Придумайте еще минимум 3 ситуации помимо предложенных в уроке, для чего может быть нужно сохранять данные пользователя и какие?

Пример из урока: запомнить, что пользователь уже залогинился, что у него лежит в корзине или в каком разделе сайта он сейчас находится

Например, в куках могут храниться индивидуальные настройки пользователя (регион, дизайн оформления и прочее) – и автоматически загружаться при загрузке страницы.

Многие формы сохраняют введенные пользователем данные в LocalStorage, пока они не будут отправлены, например, в сервисах бронирования билетов.

Также хранилище может быть использовано для сохранения созданных веб-приложением документов локально для использования в автономном режиме.

4. Как сделать валидацию номера кредитной карты?

Только с помощью pattern, обычно кредитную карту не валидируют, есть дополнительные способы проверки. Однако регулярка тоже может использоваться. Например, `/(\d{4}[-. ]?){4}|\d{4}[-. ]?\d{6}[-. ]?\d{5}/g` - простая регулярка для Visa, MasterCard, American Express, Discover.

Также нашла информацию, что популярным методом проверки номера кредитной карты является *Luhn algorithm* или *Luhn formula, написание кода может быть разным, пример:*

```
function luhn(array) {
    return function (number) {
        let len = number ? number.length : 0,
            bit = 1,
            sum = 0;

        while (len--) {
            sum += !(bit ^= 1) ? parseInt(number[len], 10) : array[number[len]];
        }
        return sum % 10 === 0 && sum > 0;
    };
}([0, 2, 4, 6, 8, 1, 3, 5, 7, 9]);
```

Или, сначала проверить номер на 16 цифр, и другой вариант Luhn algorithm:

```
function validateCardNumber(number) {
    var regex = new RegExp("^[0-9]{16}$");
    if (!regex.test(number))
        return false;

    return luhnCheck(number);
}
```

```
function luhnCheck(val) {
    var sum = 0;
    for (var i = 0; i < val.length; i++) {
        var intVal = parseInt(val.substr(i, 1));
        if (i % 2 == 0) {
            intVal *= 2;
            if (intVal > 9) {
                intVal = 1 + (intVal % 10);
            }
        }
        sum += intVal;
    }
    return (sum % 10) == 0;
}
```

5. Как сделать input, который будет принимать только числа (минимум 2 способа)?

С помощью регулярки:

```
function validateInput(numberField) {

    var numberFormat = /^\d{1,}$/;
    if (numberField.value.match(numberFormat)) {
        return true;
    }
    else {
        alert('Неверный формат заполнения');
        return false;
    }
}
```

Другой способ, использовать встроенную валидацию форм, установив `type="number"` в html коде:

```
    <form action="#" class="form">
        <input type="number" name="growth" placeholder="Ваш рост" />
        <button type="button">Отправить</button>
    </form>
```

Соответственно, если данные, введённые в input, соответствуют правилам перечисленных выше атрибутов, они считаются валидными, если нет — не валидными.

6. Найдите регулярное выражение для ФИО на русском языке

```
let regExp =
/^([А-ЯA-Z]|[А-ЯA-Z][\x27а-яa-z]{1,}|[А-ЯA-Z][\x27а-яa-z]{1,}\-([А-ЯA-Z][\x27а-яa-z
]{1,}|(оглы)|(кызы)))\040[А-ЯA-Z][\x27а-яa-z]{1,}(\040[А-ЯA-Z][\x27а-яa-z]{1,})?$/
```

или такое, более простое:

```
let regExp = /([А-ЯЁ][а-яё]+[\-\s]?){3,}$/
```

7. Как должно выглядеть невалидное поле, чтобы пользователю было понятно, что в него вводить? Как показать пользователю, что не так, если ввод некорректен (теги и атрибуты html, псевдоклассы css) ? вопрос теоретический, код писать не надо

   Невалидный элемент соответствует CSS-псевдоклассу `:invalid` или, в зависимости от ошибки, другим псевдоклассам (например, `:out-of-range`), которые позволяют применять определённые стили к элементам, не являющимся валидными. То есть некорректно заполненное поле может подсвечиваться, иным образом выделяться стилизацией, чтобы обратить внимание пользователя на него (или, например, для атрибута `required` тоже можно прописать стили в `CSS`).

   Также пользователь будет получать встроенные сообщения об ошибке, с использованием метода: `setCustomValidity(`*`message`*`)`.

8. Какие есть недостатки у стандартного способа задания валидации через HTML5?

HTML5-валидация обычно не требует большого количества JavaScript-кода и демонстрирует хорошую производительность, но не настолько настраиваема, как валидация с помощью JavaScript. JavaScript-валидация полностью настраиваема, более гибкая.

Также при валидации форм HTML невозможно управлять внешним видом встроенных сообщений об ошибке. Кроме этого, устаревшие браузеры не поддерживают HTML5-валидацию.