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

**Local Storage (локальное хранилище)**

* Хранит данные бессрочно.
* Очищается только с помощью JavaScript или очистки кэша браузера.

**Session Storage (сессионное хранилище)**

* Хранит данные, пока продолжается текущая сессия. Когда пользователь закрывает браузер, данные становятся недоступными.
* Используется контекст браузера верхнего уровня, поэтому каждая вкладка браузера хранит уникальные данные.

**Cookie (куки)**

* Хранит данные, которые можно передавать на сервер через заголовки.
* Локальное и сессионное хранилище доступны только на клиентской стороне, а cookie можно передавать на сервер.

Что бы на долго сохранить данный лучший способ для этого использовать локальное хранилище.

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

Добавление данных в localStorage очень простой процесс, мы можем использовать setItem() метод с парой ключ (key) - значение (value). Названия ключей мы придумываем сами. Ключ **всегда пишем на английском языке**, а вот значение может быть каким угодно, и русским, и английским.

localStorage.setItem('name', 'Марь Ванна');

Если мы хотите получить значение конкретного ключа, то нам надо будет использовать метод getItem().

localStorage.getItem('name');

1. 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

1. запомнить любимую еду ресторана/доставки

2. книгу и на какой странице остановился пользователь

3.

1. Как сделать валидацию номера кредитной карты?

Вы можете использовать этот фрагмент для проверки 16-значных номеров карт с помощью [алгоритма](http://en.wikipedia.org/wiki/Luhn_algorithm) Lun :

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;

}

1. Как сделать input, который будет принимать только числа (минимум 2 способа)?
2. Найдите регулярное выражение для ФИО на русском языке

let regExp = /^([А-Я]|[А-Я][\x27а-я]{1,}|[А-Я][\x27а-я]{1,}\-([А-Я][\x27а-я]{1,}|(оглы)|(кызы)))\040[А-Я][\x27а-я]{1,}(\040[А-Я][\x27а-я]{1,})?$/

if (regExp.test("Наркат Иван Иванович"))

console.log("true");

else

console.log("false");

1. Напишите самый красивый и понятный вид, как должно выглядеть невалидное поле, чтобы пользователю было понятно, что в него вводить и что не так, если ввод некорректен (теги html+css)
   1. Прописанный placeholder
   2. Перепрыгивающий курсор автоматически на следующие поле ввода
   3. Подсвеченные ошибки и подпись что нужно исправить
2. Какие есть недостатки у стандартного способа задания валидации через HTML5?

один минус этого способа - его ограниченность. Проверять мы можем только факт заполнения, но не корректность введенных данных.

атрибут minlength до сих пор не поддерживается в браузерах IE, EDGE и только с версии 10.3 появился в iOS

Псевдо классы: используя :valid и :invalid, мы можем показать пользователю, правильно ли заполнено поле по мере ввода. Однако стили применяются до того, как пользователь начнёт работу с формой. Это значит, что пользователь, даже не приступив к заполнению формы, может сразу же получить негативную обратную связь. Совсем некрасиво и непонятно для пользователя, что от него хотят.

* 1. Ограниченность, проверка только на пустые поля а не на корректность данных
  2. атрибут minlength до сих пор не поддерживается в браузерах
  3. псевдо классы сразу могут раздражать пользователя