Неделя 18  
  
**1 вопрос**Local Storage (локальное хранилище)\*\*

- Хранит данные бессрочно.

- Очищается только с помощью JavaScript или очистки кэша браузера.

- Хранит данные объёмом до 5 МБ, это самый большой объём из трёх вариантов хранилища.

- Не поддерживается старыми браузерами, например, IE 7 и ниже.

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.

- Объём данных больше чем в Cookie.

- Не поддерживается старыми браузерами, например, IE 7 и ниже.

\*\*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.

- Срок хранения устанавливается при создании cookie.

- Объём данных не превышает 4 Кбайт.

* Cookie могут быть защищёнными, в этом случае их содержимое нельзя получить на стороне клиента. Это важно при хранении пользовательских данных авторизации.  
    
  Если нам нужно надолго сохранить много пользовательских данных - используем Local Storage (локальное хранилище).  
    
  **2 вопрос**  
  Добавление данных в localStorage. Можно 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');  
    
  Чтобы получить значение конкретного ключа, надо использовать метод `getItem()`.
* localStorage.getItem(‘name');
* удалить данные можно с помощью `removeItem()`.
* localStorage.removeItem(‘key');
* А используя `clear()`, вы очистите всё веб-хранилище.

**3 вопрос**

Сохранение истории переписки, сохранение истории просмотров страниц, создание записной книжки, для персонифицированного обращения к пользователю, наверное, для контекстной рекламы  
  
**4 вопрос**  
 <input type = "credit\_card" name = "credit\_card" placeholder="credit card number" pattern="/^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6(?:011|5[0-9][0-9])[0-9]{12}|3[47][0-9]{13}|3(?:0[0-5]|[68][0-9])[0-9]{11}|(?:2131|1800|35\d{3})\d{11})$/">  
Нужно подобрать регулярное выражение под определенный тип кредитной карты. Пример выше - для наиболее используемых видов КК.  
  
**5 вопрос**Указать тип = number:

<input type = "number" placeholder="number" max=100 min=1 required>\*

**Хотя у меня можно вводить не только числа, но и буквы. Но принимает в итоге только числа.**  
  
Использовать регулярное выражение:  
 **<input type="tel" placeholder="number" pattern="/^\\d+$/">**

**<input type="tel" placeholder="number" pattern="/^[ 0-9]+$/«>**

**<input type="tel" placeholder="number" pattern="/^\d{1,}$/">**

!!! **НО здесь у меня непорядок. Я не смогла ввести цифры. Мне выдавало ошибку формата заполнения.**

**Какое регулярное выражение нужно использовать?**

**6 вопрос**

**Эти выражения также не сработали. Им все время не нравится формат заполнения!**

**<input type = "text" id = "fio" placeholder="fio" pattern="/([А-ЯЁ][а-яё]+[\-\s]?){3,}/" required>\***

**<input type = "text" id = "fio" placeholder="fio" pattern="/^[А-Яа-яЁё\s]+$/" required>\***

**7 вопрос**В CSS существует четыре специальных псевдокласса, применимых к полям формы: `:valid` (\*валидное поле\*), `:invalid` (\*невалидное\*), `:required` (\*обязательное\*) и `:optional` (\*необязательное\*). Их можно использовать, чтобы добавлять некоторые — хотя и весьма ограниченные — подсказки пользователям, заполняющим форму.

Используя `:valid` и `:invalid`, мы можем показать пользователю, правильно ли заполнено поле по мере ввода.

input:valid { border-color: green; }

input:invalid { border-color: red; }

Однако с этим способом связана одна проблема: стили применяются до того, как пользователь начнёт работу с формой. Поля, обязательные для заполнения, сразу подсветятся нам как `:invalid`, а необязательные — как `:valid`. 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.

Стилизация состояний `:required` и `:optional` сама по себе не особо полезна, поскольку эта информация обычно указывается в подписях к полям формы. Однако мы можем объединить эти состояния с псевдоклассами `:valid` / `:invalid` и стилизовать их комбинации. Например, мы хотим показывать лишь положительный результат, когда валидно обязательное к заполнению поле.

input:required:valid { border-color: green;}

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

**8 вопрос**

Некоторые атрибуты поддерживаются не во всех браузерах.   
 Мы можем проверить только факт заполнения, но не корректность введенных данных. Ошибки выводятся последовательно одна за одной, в определенном предустановленном виде.