#31 Таймеры и Интервалы

Вы уже и так умеете откладывать на потом. Теперь это возможно и в ваших программах! Уже совсем скоро вы разберетесь с асинхронностью и поймете как работает setTimeout под капотом, а сегодня будет достаточно научиться им пользоваться и применять у себя в коде

* Прочитайте про [Таймеры и Интервалы](https://learn.javascript.ru/settimeout-setinterval#rekursivnyy-settimeout) \*переходить по ссылкам из статьи не рекомендуется (рано)
* Решите задачи в конце статьи

#### #32 Браузер и DOM

Сегодня начинаем покорять браузер и знакомится с таким понятием как DOM (если вы уже верстали - вам это знакомо)

JS имеет огромное кол-во возможностей для работы с DOM - сам язык изначально и был создан именно для этого. Затем, язык менялся, появлялись библиотеки и фреймворки, менялись подходы работы с DOM и прямо сейчас может сложиться такое впечатление что работать с html из js (говоря совсем упрощенно) можно и нужно только с помощью таких библиотек как React. Эта точка зрения навязывается со всех сторон. И в ней есть доля правды.

*Но перед тем как сесть на мопед, лучше научиться крутить педали на велосипеде и держать равновесие*

Эта тема окажется намного важнее, чем любая библиотека или фреймворк. Потому что это основа. Поняв эту основу, вы сможете не изучать библиотеки для работы с UI, а просто брать их и пользоваться. Убедиться в этом можно будет уже скоро, примерно через 3 месяца 🙃

Ваша задача - ознакомиться с двумя темами:

* [**Браузерное окружение**](https://learn.javascript.ru/browser-environment)
* [**DOM-дерево**](https://learn.javascript.ru/dom-nodes)

*Нам важно понять саму суть, а значит, перечитывать все спецификации не стоит.*

#### #33 DOM и Навигация

А теперь пройдемся по DOM дереву вдоль и поперек. Здесь ничего сложного, но много важных вещей. Вы ведь уже умеете верстать? Очень скоро вам понадобится этот навык

Прочитайте про:

* [Навигацию по DOM](https://learn.javascript.ru/dom-navigation)
* [Поиск элементов](https://learn.javascript.ru/searching-elements-dom)

Достаточно понимать возможности, которые у вас есть. Не нужно запоминать все наизусть, когда мы перейдем к практике - можно будет легко вернуться и подсмотреть в справочник

#### #34 Практика Code Wars

**Ваша задача**

Решите эту задачу как минимум двумя разными способами Если за пару часов решить не получится, не переживайте, такие задачи не каждый джун может решить на собеседовании, а вы вообще об этом вчера узнали. Считайте это бонус задачей. Можете подсмотреть решение, спросить в чате и так далее. Но если сами решите - я буду гордиться вами до конца роадмапа

[Задача](https://www.codewars.com/kata/56b71b1dbd06e6d88100092a/train/javascript)

p.s. возможно, вам будет проще копировать код к себе в IDE, писать его там, тестировать и только потом проверять на code wars. Это удобно, особенно для больших задач

**Бонус**

Помните как находить среднее арифметическое? Вспоминайте и решайте простую, но интересную задачу. Здесь вам и математические операторы и методы массивов

[Задача](https://www.codewars.com/kata/563e320cee5dddcf77000158/train/javascript)

#### #36 Калькулятор UI

Сделаем простой интерфейс для калькулятора. Не самый красивый, не самый удобный, но простой в разработке. Позже, в другом потоке, мы сделаем полноценный калькулятор и да, некоторые из вас готовы сделать его уже сегодня, но нашей главной задачей будет вовсе не верстка, а работа с DOM.

В этом задании достаточно расположить элементы как показано на макете, никакой логики добавлять пока не нужно. Будем использовать нативные html тэги, такие как: input, select, button.

Макет попросите отправлю вам.

#### #37 События

Веб-страницы взаимодействуют с пользователями через события браузера. Событие - это сигнал, который браузер отправляет в ответ на действие пользователя, такое как клик мыши или нажатие клавиши. В этой статье мы рассмотрим основы работы с событиями браузера в JavaScript.

**Обработчик событий**

Обработчик событий - это функция, которая вызывается в ответ на возникновение определенного события. Чтобы добавить обработчик событий к элементу, нужно использовать метод addEventListener

<button id="myButton">Нажми меня</button>

<script>

var button = document.getElementById('myButton');

function buttonClickHandler() {

alert('Кнопка нажата');

}d

button.addEventListener('click', buttonClickHandler);

</script>

Попробуйте запустить этот код у себя

Здесь мы создаем кнопку с идентификатором myButton и добавляем к ней обработчик событий click. Когда пользователь нажимает на кнопку, вызывается функция buttonClickHandler, которая выводит сообщение "Кнопка нажата" с помощью функции alert

**Объект события**

При вызове обработчика событий в функцию передается объект события, который содержит информацию о событии, такую как тип события, целевой элемент и координаты мыши. Объект события можно использовать для отмены действия по умолчанию, перенаправления события на другой элемент или передачи дополнительных параметров в обработчик событий.

Например, давайте добавим обработчик событий для ссылки, который отменяет переход по ссылке и выводит сообщение с именем ссылки

<a href="http://example.com" id="myLink">Ссылка</a>

<script>

var link = document.getElementById('myLink');

function linkClickHandler(event) {

event.preventDefault();

alert('Вы нажали на ссылку ' + event.target.innerText);

}

link.addEventListener('click', linkClickHandler);

</script>

Здесь мы добавляем обработчик событий click для ссылки и используем метод preventDefault объекта события, чтобы отменить переход по ссылке по умолчанию. Затем мы выводим сообщение с именем ссылки с помощью свойства innerText объекта события target.

**Всплытие событий**

Событие может "всплывать" от вложенного элемента к его родительским элементам. Всплытие событий позволяет реагировать на события, которые происходят внутри элемента, но обрабатывать их на более высоком уровне документа. Чтобы прекратить всплытие события, нужно использовать метод stopPropagation объекта события.

Создадим несколько вложенных элементов и добавим обработчики событий

<div id="outerDiv">

<div id="innerDiv">

<button id="innerButton">Нажми меня</button>

</div>

</div>

<script>

var outerDiv = document.getElementById('outerDiv');

var innerDiv = document.getElementById('innerDiv');

var innerButton = document.getElementById('innerButton');

function buttonClickHandler() {

alert('Кнопка нажата');

}

function divClickHandler() {

alert('Див нажат');

}

innerButton.addEventListener('click', buttonClickHandler);

innerDiv.addEventListener('click', divClickHandler);

outerDiv.addEventListener('click', divClickHandler);

</script>

Здесь мы создаем три элемента: внешний див outerDiv, внутренний див innerDiv и кнопку innerButton. Мы добавляем обработчик событий click для каждого элемента, который выводит сообщение в окне alert

Если пользователь нажимает на кнопку, вызывается функция buttonClickHandler и функции для внешних дивов (попробуйте добавить в неё event.stopPropagation()). Если пользователь нажимает на внутренний див, вызывается функция divClickHandler для внутреннего и внешнего дивов. Если пользователь нажимает на внешний див, вызывается функция divClickHandler только для внешнего дива

**Делегирование событий**

Делегирование событий - это техника, которая позволяет добавлять один обработчик событий к родительскому элементу, который будет реагировать на события, происходящие в его дочерних элементах. Это позволяет уменьшить количество обработчиков событий на странице и сделать код более эффективным.

Например, давайте создадим список с несколькими элементами и добавим обработчик событий к его родительскому элементу:

<ul id="myList">

<li>Первый элемент</li>

<li>Второй элемент</li>

<li>Третий элемент</li>

</ul>

<script>

var list = document.getElementById('myList');

function listItemHandler(event) {

alert('Вы выбрали элемент: ' + event.target.innerText);

}

list.addEventListener('click', listItemHandler);

</script>

Здесь мы добавляем обработчик событий click к списку и используем объект события target, чтобы определить, на какой элемент был произведен клик. Затем мы выводим сообщение с текстом элемента.

**Заключение**

События браузера играют важную роль во взаимодействии веб-страниц с пользователями. Обработчики событий позволяют реагировать на события, происходящие на странице, а объекты событий предоставляют информацию

Это основные концепции работы с событиями браузера в JavaScript. Есть еще множество других событий, свойств и методов, которые можно изучить по мере необходимости

**Ваша задача**

* На странице есть кнопка “Секундомер”.
* При нажатии на кнопку необходимо запускать “секундомер” и выводить в консоль время каждую секунду.
* Также необходимо предусмотреть возможность паузы/возобновления при нажатии на кнопку еще раз.

*Для решения этой задачи вам понадобится использовать функции setInterval и clearInterval для запуска и остановки таймера, их вы уже знаете*

#### #38 Калькулятор

Логика готова, представление готово, осталось вставить одно в другое.

**Ваша задача**:

* Иметь два файла index.html и main.js.
* Подключите main в ваш index
* Добавьте обработчики для кнопок (event listeners)

Вот вам и готовый калькулятор 💁🏽‍♂️

Только вы ещё не читали про вставку/изменение текста, давайте посмотрим на самые важные моменты которые могут быть использованы для работы с содержимым элементов

**innerHTML: Содержимое элемента**

Свойство innerHTML используется для доступа к содержимому элемента, включая любые дочерние элементы, которые могут находиться внутри него. Оно также может быть использовано для изменения содержимого элемента.

<div id="myDiv">Это div с <strong>жирным текстом</strong>.</div>

<script>

// получаем содержимое div-элемента

const content = document.getElementById('myDiv').innerHTML;

alert(content); // выводит "Это div с <strong>жирным текстом</strong>."

// изменяем содержимое div-элемента

document.getElementById('myDiv').innerHTML = 'Новое содержимое div-элемента.';

</script>

**outerHTML: HTML элемента**

Свойство outerHTML используется для доступа к HTML-коду элемента, включая сам элемент и все его дочерние элементы. Оно также может быть использовано для изменения элемента целиком.

<div id="myDiv">Это div с <strong>жирным текстом</strong>.</div>

<script>

// получаем HTML-код div-элемента

const html = document.getElementById('myDiv').outerHTML;

alert(html); // выводит "<div id="myDiv">Это div с <strong>жирным текстом</strong>.</div>"

// изменяем HTML-код div-элемента

document.getElementById('myDiv').outerHTML = '<p>Новый абзац</p>';

</script>

**nodeValue/data: Содержимое текстового узла**

Свойство nodeValue (или data) используется для доступа к содержимому текстового узла. Оно также может быть использовано для изменения содержимого текстового узла.

<div id="myDiv">Это жирный текст.</div>

<script>

// получаем содержимое текстового узла

const content = document.getElementById('myDiv').firstChild.nodeValue;

alert(content); // выводит "Это жирный текст."

// изменяем содержимое текстового узла

document.getElementById('myDiv').firstChild.nodeValue = 'Новый текст.';

</script>

**textContent: Просто текст**

Свойство textContent используется для доступа к содержимому элемента, но без учета всех дочерних элементов. Оно также может быть использовано для изменения содержимого элемента.

<div id="myDiv">Это <strong>жирный</strong> текст.</div>

<script>

// получаем содержимое элемента

const content = document.getElementById('myDiv').textContent;

alert(content); // выводит "Это жирный текст."

// изменяем содержимое элемента

document.getElementById('myDiv').textContent = 'Новый текст.';

</script>

Здесь свойство textContent используется для получения текстового содержимого элемента "myDiv", который содержит текст и дочерний элемент "strong". Затем свойство textContent используется для изменения содержимого элемента "myDiv" на новый текст "Новый текст.".

Важно отметить, что использование свойств innerHTML и outerHTML может привести к потенциальным уязвимостям безопасности, если введенный пользователем контент не экранируется должным образом перед использованием в HTML-коде. Поэтому рекомендуется использовать свойство textContent для изменения текстового содержимого элементов, особенно если пользователь может вводить данные.

#### #39 Атрибуты и свойства / Стили и классы

Продолжаем изучать работу с DOM.

Теории по этой теме очень много, но не стоит зубрить ее наизусть. Достаточно понимать как это устроено и, в случае необходимости, знать что такие возможности у вас есть

## Атрибуты и свойства

JavaScript позволяет получать доступ к атрибутам и свойствам HTML-элементов на странице. Атрибуты это значения, которые указываются в HTML-тегах, например, **id** или **class**. Свойства - это значения, которые можно получить или установить с помощью JavaScript, например, **value** или **textContent**.

Атрибуты HTML-элементов можно получить с помощью свойства **getAttribute**, а свойства самих элементов - с помощью точечной нотации или квадратных скобок. Вы уже видели это в предыдущей задаче

// Получить значение атрибута

const elem = document.getElementById('myElement');

const value = elem.getAttribute('data-value');

// Получить значение свойства

const text = elem.textContent;

Обратите внимание, что некоторые атрибуты и свойства имеют разные названия. Например, атрибут **class** имеет свойство **className**, а атрибут **for** имеет свойство **htmlFor**.

### DOM-свойства

DOM-свойства - это свойства элементов, которые не указаны в HTML-коде, но создаются браузером автоматически при загрузке страницы. Например, свойство **style** позволяет получить доступ к CSS-свойствам элемента:

const elem = document.getElementById('myElement');

const backgroundColor = elem.style.backgroundColor;

### HTML-атрибуты

HTML-атрибуты можно устанавливать и изменять с помощью свойства **setAttribute** или просто изменяя свойство элемента. Например:

// Установить значение атрибута

const elem = document.getElementById('myElement');

elem.setAttribute('data-value', '10');

Обратите внимание, что изменения HTML-атрибутов не всегда влияют на свойства элементов и наоборот. Например, изменение атрибута **class** не изменяет свойство **className**, и наоборот.

### Ваша задача:

Напишите функцию, которая

* Изменяет фон страницы на заданный цвет по клику на кнопку "Изменить фон"
* Должна получать цвет из атрибута data-color кнопки
* Присваивать его в качестве свойства background-color элементу body на странице.

<button id="change-bg" data-color="#e6e6e6">Изменить фон</button>

пример кнопки

### Классы и className

Классы - это механизм, который позволяет задавать набор стилей для нескольких элементов одновременно. Чтобы задать класс элементу, нужно использовать атрибут **class**

<div class="my-class">Этот элемент имеет класс my-class</div>

Для работы с классами в JavaScript есть свойство **className**. Оно позволяет получать и задавать классы элемента

let element = document.querySelector('.my-class');

element.className = 'new-class';

В этом примере мы выбираем элемент с классом **my-class** и задаем ему новый класс **new-class**.

### classList

Кроме свойства **className**, в JavaScript есть объект **classList**, который предоставляет более удобный способ работы с классами. С помощью методов **add**, **remove**, **toggle** и **contains** можно добавлять, удалять и проверять наличие классов у элемента. Например:

let element = document.querySelector('.my-class');

element.classList.add('new-class');

element.classList.remove('my-class');

console.log(element.classList.contains('new-class')); // true

console.log(element.classList.contains('my-class')); // false

В этом примере мы выбираем элемент с классом **my-class**, добавляем ему класс **new-class**, удаляем класс **my-class** и проверяем наличие классов у элемента.

### Element style

Свойство **style** позволяет задавать стили элемента непосредственно из JavaScript. В этом примере мы выбираем элемент с классом **my-class** и задаем ему красный фон.

let element = document.querySelector('.my-class');

element.style.backgroundColor = 'red';

❗

При задании значений стилей следует учитывать единицы измерения. Например, ширина элемента может быть задана как **100px**, а не просто **100**. В противном случае браузер может не правильно интерпретировать значение.

### Вычисленные стили: getComputedStyle

Для получения вычисленных стилей элемента можно использовать метод **getComputedStyle**. Он возвращает объект, содержащий все текущие стили элемента

let element = document.querySelector('.my-class');

let styles = getComputedStyle(element);

console.log(styles.backgroundColor);

В этом примере мы выбираем элемент с классом **my-class** и получаем его вычисленный стиль для свойства **backgroundColor**. Метод **getComputedStyle** возвращает объект **CSSStyleDeclaration**, который содержит все текущие стили элемента. Этот объект не является массивом, но можно получить значение конкретного свойства, обратившись к нему по имени, как показано в примере выше.

### Ваша задача:

Попробуйте вывести все эти свойства в консоль и посмотреть что в них содержится

#### Бонусная задача (только если вы в темпе)

Создать функцию, которая

* будет менять цвет фона страницы каждые 2 секунды
* на случайный цвет из заранее определенного списка

Подсказки

пример массива цветов

const colors = ['red', 'blue', 'green', 'orange'];

пример получения рандомного цвета

const randomColor = colors[Math.floor(Math.random() \* colors.length)];

#### #40 Модификация документа

В калькуляторе все ваши UI элементы были статичными, но очень часто нужно добавлять или удалять элементы (список задач в TODO, например), для этого понадобятся специальные методы

### Создание элемента

Для создания нового элемента в DOM можно использовать метод **createElement**. Он принимает название тега в качестве аргумента и возвращает созданный элемент. Затем его можно настроить с помощью различных свойств и методов, и вставить в документ.

// Создаем новый элемент <div>

const newDiv = document.createElement('div');

// Устанавливаем текстовый контент

newDiv.textContent = 'Hello, world!';

// Добавляем стиль

newDiv.style.color = 'red';

// Вставляем элемент в тело документа

document.body.appendChild(newDiv);

### Методы вставки

Существует несколько методов для вставки элементов в DOM. Рассмотрим некоторые из них.

#### appendChild

Метод **appendChild** добавляет элемент в конец родительского элемента. Если элемент уже присутствует в DOM, он будет перемещен в новое место.

const parent = document.querySelector('#parent');

const child = document.querySelector('#child');

parent.appendChild(child);

#### insertBefore

Метод **insertBefore** добавляет элемент перед указанным элементом. Если вторым аргументом не указать другой элемент, то элемент будет добавлен в конец родительского элемента.

const parent = document.querySelector('#parent');

const child = document.querySelector('#child');

const referenceElement = document.querySelector('#reference-element');

parent.insertBefore(child, referenceElement);

#### replaceChild

Метод **replaceChild** заменяет указанный элемент другим элементом.

const parent = document.querySelector('#parent');

const newChild = document.createElement('div');

const oldChild = document.querySelector('#old-child');

parent.replaceChild(newChild, oldChild);

### insertAdjacentHTML/Text/Element

Методы **insertAdjacentHTML**, **insertAdjacentText** и **insertAdjacentElement** позволяют вставлять HTML, текст и элементы перед, после, в начало и в конец элемента. В первом аргументе нужно указать позицию вставки, во втором - содержимое.

const parent = document.querySelector('#parent');

parent.insertAdjacentHTML('beforeend', '<div>Hello, world!</div>');

parent.insertAdjacentText('beforeend', 'Hello, world!');

const newChild = document.createElement('div');

parent.insertAdjacentElement('beforeend', newChild);

### Удаление узлов

Существует несколько методов для удаления элементов из DOM

Метод **removeChild** удаляет указанный элемент из родительского элемента.

const parent = document.querySelector('#parent');

const child = document.querySelector('#child');

parent.removeChild(child);

Метод **remove** удаляет элемент из DOM.

const element = document.querySelector('#element');

element.remove();

### Клонирование узлов: cloneNode

Метод **cloneNode** создает копию элемента, включая его атрибуты, дочерние элементы и текстовые узлы. По умолчанию создается поверхностная копия элемента, то есть копируются только атрибуты и структура элемента, но не его содержимое. Метод принимает один булевый аргумент, который указывает, нужно ли клонировать все дочерние элементы.

const originalElement = document.querySelector('#original-element');

// Клонируем элемент со всем его содержимым

const clonedElement = originalElement.cloneNode(true);

// Добавляем клон в тело документа

document.body.appendChild(clonedElement);

## Ваша задача

В вашей только что написанной программе “Калькулятор” добавьте сохранение всех результатов: после каждого расчёта (клика по кнопке “=” ) добавляйте новый div под строкой калькулятора и записывайте в него результат вычислений. После нескольких расчетов у вас получится что-то похожее на:

12 32 43 545 12

По клику на каждое из этих чисел - удаляйте его див из разметки.

У вас будет 2 операции из новой темы: добавление узла в DOM и удаление узла. А еще вам нужно будет “вешать” события на только что созданные “дивы”. Это не самая простая задача, но разве вас уже можно хоть чем-то напугать

#### #41 Модули

Вот вам и супер интересная тема. Вы наверняка заметили что ваши файлы с кодом становятся все объемнее, что хочется вынести некоторые вещи куда-то за пределы основной программы? Как раз для этого есть модули. Для начала нам хватит только самых базовых знаний

Ваша задача:

* прочитать про [Модули](https://learn.javascript.ru/modules-intro)
* создайте для вашего калькулятора хотя бы 1 модуль и вынесите в него часть кода

Чуть позже вы научитесь делить ваши программы на логичные и переиспользуемые модули. А пока просто практикуйте деление

#### #43 TODO UI

Сделаем TODO с интерфейсом! С очень простым интерфейсом, без каких либо оформлений (наша цель пока не в красоте)

**Ваша задача: сделать всю верстку без какой-либо логики** (макет попросите отправлю)- подсказки можно отключить в меню слева (обращайтесь в чат, если не получится найти их)

Пусть вас не пугает этот макет в сравнении с калькулятором, если вы разобьете все на логические блоки - станет ясно, что здесь вообще ничего сложно нет

#### #44 Формы

Сегодня вам предстоит научиться работать с полем input чтобы добавлять новые задачи в список.

Для этого будет не лишнем прочитать эту теорию про [Формы](https://learn.javascript.ru/forms-submit).

**Ваша задача:**

* добавить обработку для поля ввода по нажатию на enter
* после получения текста из формы, создавать новый элемент (тот, где имя задачи, чек бокс и кнопка удалить)
* добавлять этот элемент в DOM

да, поле для добавления новой задачи должно быть именно формой

нет, весь остальной функционал сейчас делать не нужно, только форма - enter - создание дива - добавление в верстку

кажется все это вы уже делали в прошлых задачах

#### #45 TODO

Сегодня нужно просто доделать туду. Так, чтобы можно было удалять, добавлять и отмечать как «готово» все ваши задачи из списка

Все знания для этого у вас уже есть, осталось перенести их в код, но я подскажу вам без деталей, как это лучше сделать:

* Взять за основу тот самый todo написанный на массивах
* Взять недавно сделанную верстку
* Добавить обработку событий для удаления, выполнения и добавления задач (добавление уже должно быть готово)
* По каждому из событий в UI вызывать уже готовые функции для работы с массивом (changeStatus, remove, add)
* После каждого изменения массива запускать функцию render (о ней ниже)

**Функция render:**

Так как теперь ваши задачи есть и в массиве в JS и в верстке (DOM), нужно их как-то синхронизировать и выбрать один источник правды. Этим источником будет массив задач в JS. После каждого изменения этого массива можно удалять все элементы в DOM, а потом создавать и вставлять их в DOM снова. Так, все что есть в UI будет всегда отражением того, что есть в данных (JS).

Да, это не самый элегантный способ, но сейчас важно научиться и понять подобный механизм.

**Extra**: будьте наставником для тех, у кого не получилось или не получилось в самом лучшем виде

#### #46 Ошибки

Пришло время научиться по-взрослому обрабатывать ошибки. Больше никаких строк, типа “Error”, больше никаких «ой, ошибка».

Теперь все по-настоящему! После чтения теории, добавьте обработку ошибок в ваши программы. Это может быть хоть калькулятор, хоть туду, хоть все вместе.

[Обработка ошибок](https://learn.javascript.ru/try-catch)

#### #47 JSON

Сначала прочитайте эту статью про [формат JSON](https://learn.javascript.ru/json) и методы для встроенного класса JSON

Выполнять задачи под этой статьей не нужно. Главное, чтобы вы в теории понимали что такое JSON и чем он может быть полезен

**А теперь немного практики:**

*Пока мы не используюем специальных подходов чтобы импортировать данные из json файла, сделаем это экспериментальным путем (не запоминайте этот подход, он нужен только для практики)*

* Скачайте файл data.json из вложения

[data.json](https://s3.us-west-2.amazonaws.com/secure.notion-static.com/c6378cc3-351e-4460-a2d5-e6c61f759c5a/data.json?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Credential=AKIAT73L2G45EIPT3X45%2F20230310%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Date=20230310T033429Z&X-Amz-Expires=86400&X-Amz-Signature=849377e4d96bf07898129f3b939a46e01a2213ddbb43321b2beaf04b066585ba&X-Amz-SignedHeaders=host&x-id=GetObject" \t "_blank)

[0.6KB](https://s3.us-west-2.amazonaws.com/secure.notion-static.com/c6378cc3-351e-4460-a2d5-e6c61f759c5a/data.json?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Credential=AKIAT73L2G45EIPT3X45%2F20230310%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Date=20230310T033429Z&X-Amz-Expires=86400&X-Amz-Signature=849377e4d96bf07898129f3b939a46e01a2213ddbb43321b2beaf04b066585ba&X-Amz-SignedHeaders=host&x-id=GetObject" \t "_blank)

* Создайте файл main.mjs (внимание на расширение файла)
* В вашем файле main.mjs добавьте импорт файла с данными data.json (файлы должны быть в одной папке) import data from './data.json' assert {type: "json"};
* Откройте терминал в этой папке и запустите ваш код со специальным флагом node --experimental-json-modules main.mjs
* Теперь в переменной data будут данные из файла json
* Можете проявить свою фантазию и вывести эти данные в консоль разным образом. например вывести только имена людей или вывести строку типа Dennis, born at September 9, 1941 and created the C programming language для каждого из участников В этой задаче так же можно попробовать все методы из прочитанной вами статьи

Уже скоро мы будем получать такие данные по сети и делать с ними интересные вещи

#### #48 localstorage

Знаю знаю, в TODO вы очень хотели хранить список дел хоть где-то, чтобы после перезагрузки страницы он не пропадал. Теперь у вас есть такая возможность. Но, добавлять это в старые программы не обязательно. Уже очень скоро мы попробуем это в деле, так что можете ограничится теорией

**Ваша задача:** Прочитать про [LocalStorage](https://learn.javascript.ru/localstorage)

*Пусть вас не пугают упоминания про куки, с ними мы познакомимся чуть позже*

#### #50 Колбэки

Вы уже вовсю их используете, но прочитать эту тему не будет лишним чтобы подготовиться к серьезной и сложной следующей. callback - это функция, которая будет вызвана после завершения текущего действия. Так мы можем “дождаться” завершения кода и только потом вызвать нужную нам функцию

Прочитайте про [Колбэки](https://learn.javascript.ru/callbacks)

#### #51 Промисы

Ух. Промисы - это супер большая и супер мощная тема. Мы разберем только четверть из нее в этой статье и вы сильно не пугайтесь, если поймете не все и сразу, мы будем разбираться с ними еще очень долго. Главное, понять основную идею этого класса и самые простые случаи использования.

[Промисы](https://learn.javascript.ru/promise-basics)

#### #52 Fetch (основы)

Сегодня мы узнаем о способе получения данных по сети. Для начала, представьте для себя Fetch как промис, который умеет что-то извлекать по сети. Этого на данном этапе хватит

Чтобы не запутаться и не забить голову лишним - прочитайте только часть этой статьи, вот до этого примера:

То же самое без await, с использованием промисов

Дальше читать строго не советую. Мы вернемся к этой статье и рассмотрим ее подробнее чуть позже

[Fetch](https://learn.javascript.ru/fetch)

#### #53 Genderize

**Пришло время воспользоваться вашими знаниями на практике:**

Напишите программу, которая передает на сервер имя и в ответ получает "возможный" пол для этого имени. Вот вам готовая строка с URL (только имя успевайте менять)

const firstName = 'artem'; const serverUrl = 'https://api.genderize.io'; const url = `${serverUrl}?name=${firstName}`;

Сделайте самый простой интерфейс для этой программы (форму):

* input - в него введете имя
* кнопку рядом - она будет отправлять это имя на сервер
* в alert или в строке ниже показывайте результат вот такого вида

"Artem is male"

*p.s. помните fetch не работает в nodeJS, а значит вам нужно запускать это в браузере*

#### #54 Цепочки промисов

Их сложно понять, просто забыть, так что лучше повторить 🤪 Кто-то из вас уже понял что из функции можно вернуть другую функцию, то же самое касается и промисов; Промис может вернуть другой промис - так получается цепочка. Теперь пора как следует понять как это работает

Прочитайте про [Цепочки промисов](https://learn.javascript.ru/promise-chaining)

*p.s. пример с Thenable читать строго не рекомендуется*

#### #55 Погода: верстка

Сегодня начнем делать программу побольше, чем до этого. Полноценное приложение, прям настоящее Погодное приложение, с поиском городов, детальной сводкой и главное: сетевыми запросами. Ну еще и с сохранением городов

Начнем с верстки. Предлагаю сделать 2 экрана сегодня - Now и Details Никакой логики, только верстка (самым сложным тут будут табы - разные вкладки для разных экранов)

Макет попросите отправлю

Бонус (для самых продвинутых кул хацкеров):

Сделайте и 3-й экран (forecast). Но не беритесь за него если потратили на задачу выше все 4 часа, это бонусное задание не должно идти вам в ущерб

#### #57 Погода: Запросы

Пора запросить хотя бы один город из строки поиска. Ничего сложного - вы уже делали все это в прошлых практических задачах. (Gendorize)

**Ваша задача:**

1. вводим что-то в строку поиска, по нажатию на Enter - берем имя города из этой строки
2. подставляем это имя в url через шаблонные строки и делаем запрос на сервер

const serverUrl = 'http://api.openweathermap.org/data/2.5/weather';

const cityName = 'boston';

const apiKey = 'f660a2fb1e4bad108d6160b7f58c555f'; (этот ключ имеет ограничение в кол-ве запросов, если будут ошибки - придется сгенерировать новый или спросить в чате)

const url = `${serverUrl}?q=${cityName}&appid=${apiKey}`;

1. результат записываем в любую переменную и выводим нужную информацию в табе 'Now' согластно верстке

Все вместе: получить имя города из поля ввода, сделать fetch по нужному урлу, изменить содержимое пары полей в верстке. Согласитесь просто?

#### #58 Погода: Ошибки

Типичная обработка ошибок запроса по сети. У пользователя может отвалиться интернет, сервер может не ответить вовремя или ответить но с ошибкой - все это мы должны обработать и донести эту информацию до пользователя. Для простоты будем показывать обычный alert с сообщением “ошибка” (потом научимся делать круче)

Но, перед этим прочитайте тему про обработку ошибок в промисах и используйте эту технику: [Обработка ошибок](https://learn.javascript.ru/promise-error-handling)

*Чтобы имитировать ошибку и проверить себя - поменяйте что-то в url, например адрес сервера*

#### #59 Погода: Добавить и удалить

Добавим возможность добавлять города, чтобы не вписывать их вручную в строку поиска каждый раз. Узнаете? Это ж ровно то же самое что вы делали в ТУДУ листе. Можете использовать ту же логику, а можете написать все с полного нуля.

По клику на иконку с сердцем добавляйте города в список справа, по клику на крестик рядом с именем города в этом списке - удаляйте его навсегда По клику на сам город - добавляйте его данные в окно 'Now'

Вот вы уже пол программы и написали, не сложно же?)

#### #60 Погода: Запомнить

Сейчас после перезагрузки страницы все ваши города исчезают и это ужасно не удобно.

localStorage - отличное место для хранения этой информации в браузере - добавленные в избранное города всегда будут на своем месте. А еще, давайте сохранять текущий "выбранный" город и после перезагрузки снова делать выбранным именно его

Для работы с localStorage можно создать отдельный модуль, чтобы не сваливать все в кучу. Да и выглядеть это будет ой как красиво

storage.saveFavoriteCities(favoriteCities)

const favoriteCities = storage.getFavoriteCities();

const currentCity = storage.getCurrentCity();

*Это просто пример, делайте как сами считаете нужным.*

#### #61 Погода: Детали

По примеру с табом 'Now' сделайте таб 'Details' - здесь будет похожая логика, только набор данных чуть больше. Верстка уже готова, данные у вас уже есть, осталось только подставить их в DOM. Уверен вы запросто справитесь

*А вот вам* [*несколько советов по стилю кода*](https://learn.javascript.ru/coding-style)

#### #62 Погода: Рефакторинг

К этому дню все должно работать как задумано.

Если это не так - доделывайте вашу программу и обязательно обращайтесь за помощью в чат. (создавайте запросы) Если все уже готово - посмотрите на свой код еще раз. Представьте что его будет обслуживать и поддерживать серийный убийца, который знает где вы живете. Сделайте выводы и приведите код в порядок.

В этом здорово может помочь [деструктуризация](https://learn.javascript.ru/destructuring-assignment) (ее вы видели в примерах кода)

Бонусное задание: помните про вкладку forecast в вашей погоде? Самое время её сделать! Получить данные для этой вкладки вы можете на <http://api.openweathermap.org/data/2.5/forecast>