**QA JS Junior Interview:**

1. **Что такое Javascript?**

Javascript – высокоуровневный, динамически типизированный, слабо типизированный, однопоточный, интерпретируемый, встраиваемый, мультипарадигеменный язык программирования.

*Высокоуровневый* – в JS высокий уровень абстрации благодаря удобным инструментам и конструкциям, которые сокращают код и делает порог вхождения легче. Если сравнивать с Ассемблером, то чтобы вывести в консоль “Hello World!” в JS это 1 строка console.log(“Hello Wolrd!”), а в Ассемблере надо весь процесс описывать в несколько строк.

*Динамический типизированный* – тип переменной может переопределяться на этапе выполнения (runtime), в зависимости от присваиваемого значения. Это один из минусов JS – но это решается благодаря Typescript.

*Слабо типизированный* – это некое закругление углов от JS, выполняя автоматическое приведение типов, чтобы приложение было более отказоустойчивым. Это также мешает нам как разработчикам в долгосрочным перспективе в разработке приложения. Также это исправляется балгодаря Typescript.

*Однопоточный* – JS работает в одном потоке, в каждый момент времени выполняется только одна инструкция. Однако благодаря механизму событийного цикла (Event Loop) и асинхронным конструкциям, таким как callback-и, promise, async/await, JS может быть асинхронным.

*Интерпретируемый* – то есть JS код неподстредственно переводится в код машинного языка, интерпретируется.

*Встраиваемый* – JS может быть встроен в различные среды выполнения, такие как Node.js, React, Vue, React Native и другие.

*Мультипарадигменный* – JS поддерживает несколько парадигм программирования, позволяя разработчикам выбирать подходящий стиль программирования для решения конкретных задач. Есть несколько видов парадигм:  
**Императивное программирование** - фокусировка выполнение задачи с использованием последовательных инструкций. Плюс: простота. Минус: сложность или невозможность масштабировать.

**ООП (Объектно-ориентированное программирование)** – это парадигма, в которой программирование строится вокруг объектов, являющихся экземплярами классов.

**Функциональное программирование** – парадигма, при которой ключевым блоком является фукнция. Функция высшего порядка (High Order Function) – Функция которая принимает как аргумент другую функцию или возвращает функцию.

**Событийно-ориентированное программирование** – события как ключевая конструкция, когда мы описывем события и вызываем callback, когда это события происходит. document.getElementById(“myButton”).addEventListener(“click”,

function() {

alert(“Button clicked!”);

});

**Асинхронное программирование** – поддерживание асинхронного программирование через callback, promise, async/await.

1. **Отличие var, let, const?**

let и const были введены в EcmaScript 2015 (ES6) и предоставляют улучшенные возможности по сравнению с var. var в современных проектах почти что не используют. Но если нужна производительность в разных серовнованиях по программированию, можно использовать var, ведь в нем нет валидации, которые есть в let и const. Но profit мизерный, так что var устарел в проектах.

**Область видимости** - var обладает функциональной областью видимости и глобальной областью видимости. let, const обладают блочной областью видимости, переменные объявленные с помощью let и const, доступны только внутри блока, в котором они объявлены.

**Всплытие (Hoisting)** – обращение к переменной var до его объявление всплывает как undefined, let и const всплывает в виде ошибки ReferenceError (TDZ). (Временная мертвая зона)

**Переопределение и объявление** – var позволяет повторно объявлять переменную в одной и той же области видимости, let не позволяет повторное объявление в одной и той же области видимости, но позволяет переопределять значение переменной. const не позволяет ни повторно объявлять, ни переопределять значние. Исключение: если мы переопределяем внутри const значение объекта или массива, в том случаем мы просто меняем значение объекта или массива, которые являются значением переменной const.

1. **Какие скобки есть в JS?**

**Круглые скобки ():**

Функции: Используется для вызова функций и передачи аргументов. Например, const myFunction = (arg1, arg2) => {…}.  
Условные выражения: В if, while, for. Например if(x > 5) {….}.

Группировки операций: Для задания приоритета операций как в математике. Например: (2 + 3) \* 4.

**Фигурные скобки {}:**

Блоки кода: Для обозначение блоков для функций, операторов if (x = 5) {…};

Объекты: Для создание объектов const a = { name: “Merey”, “age”: 30};

**Квадратные скобки []:**

Массивы: Для создание массивов: const array = [1, “merey”, name];

Доступ к элементам массива или объекта:

К свойству массива по индексу array[0];

К объекту по строковому ключу person[“name”];

1. **Типы данных в JS?**

**number** – 42(целые числа), 3.14(плавающие числа) и NaN(NotANumber) –все они относятся к типу number.

**string** – ординарные, двойные ковычки и backtick(обратный апостраф, шаблонные литералы) все они относятся к типу string.

Преимущества backtick – записывание внутри него значение переменной с помощью знака $ и фигурных скобок без конкатенации. Также можно переносить строки без спецсимвола /n.

**boolean** – логический тип. True и False.

Console.log(Boolean(“Hello”)) // true – Проверка на boolean

Console.log(!!”Hello”) // true - Проверка на Boolean

Console.log(!“”) – Проверка на Boolean !”” = Не False = // true

**null** – “Ничего”, тип который инициализируем всегда мы сами, JS никогда его самостоятельно не инициализирует.

**undefined** – тип по умолчению, “Неопределенный”, когда у переменной не присвоено значение, несуществующий объект и т.д

**Symbol** – тип уникальности, можем присвоить уникальное значение, которое не будет ровно ни к чему, ведь оно уникально. Let uniqueKey = Symbol(“key”);

**BigInt** – ипользуется для представление целых чисел, которые выходят за переделы допустимых значений для типа number.

**Object** – тип объект.

**Array** – внутри него мы можем хранить микс всех данных с разными значениями. Typeof(numbers) //object. Можем проверить только через Array.isArray(numbers) //true.

**Function** – объекты, которые могут быть вызваны.

**Date** – дата, только в милисекундах хранится дата.

Во-первых, они принадлежат к 7 «примитивам» (примитивным типам) JS:

**let** primitiveTypes = ['string', 'number', 'null', 'undefined', 'boolean', 'symbol', 'bigint']

В JavaScript есть два типа данных: примитивный и не примитивный. Примитивный тип не является объектом и не имеет собственных методов. Все примитивные значения неизменяемы. Они включают логическое значение, нуль, неопределенное значение, число, строку и символ.

1. **Различие между обычной и стрелоной фукнцией.**

Синтаксис. Также с function можно взаимодействовать до его объявление, с стрелочными нельзя. Контекст this, в function она возвращает, в const она не существует. Function функции можно создать его экземпляр внутри переменной с помощью const newFunc = new myFunction();

1. **Сравнение в JS.**

== (нестрогое равенство) – сравнение значений без строгого сравнение типов.

=== (строго равенство) – сравнение значений с строгой типизацией.

NaN всегда не равен NaN, можно сравнить его только с помощью Number.isNaN(NaN) //true

Также может использоваться с не !==, !===

1. **Оператор && И.**

Логическое амперсанты && И. Возращает первое значение в виде false или последнее значение в виде true, когда все значение истинные.

Аналог оператора if &&. Condition && console.log(“condition is true”);

1. **Тернарный(условный) оператор.**

Сокращный вид оператора if else.

Который принимает условие ? выражениеПравда : выражениеЛожь

Можно использовать и развивать множественные тернарные операции.

1. **Сравнение объектов**

Объекты и примитивы сравниваются по разному. Примитивы сравниваются по значению, а объекты по ссылке или адресу.

Let a = { a: 1}; let b = { a: 1 }; let c = a; a===b //false a==c //true

1. **Что такое замыкание (Closures)?**

Это функция, которая сохраняет доступ к переменным из своего внешнего окружения даже после того как это окружение перестало существовать.

Function urlGenerator(domain) {

Return function(url) {

Return `hhtps://{url}.{domain}`;

};

};

Const comDomin = urlGenerator(“com”);

Console.log(comDomin); //function(url) { return `https://{url}.{domain //com}`};

Com(“google”); // hhtps://google.com

1. **Что такое Callback?**

Это функция, которая передается в другую функцию в качестве аргумента и вызывается») по завершении определенного действия или события.

1. **Что такое область видимости (Scope)?**

Есть три вида области видимости.

Глобальный область видимости, Функциональный область видимости, Блочный область видимости.

Сначала идет поиск переменной внутри блочной, потом функциональной, дальше глобальной области видимости.

1. **Что такое Event Loop(Цикл событий)?**

Это механизм, который управляет последователностью выполнением кода. Javascript является однопоточным, у него есть синхронные задачи, но также есть асинхронные. Которые в свою очередь делятся на микротаски и макротаски.

1. **Что такое Макротаски и Микротаски?**

**Макротаски** – это крупные задачи, такие как:

SetTimeout, SetInterval, вызовы DOM-событий(click, load), сетевые операции (fetch).

Макротаски ставятся в очередь макротасок (или Task Queue) и выполняются по одной, после того как стек вызовов становится пустым.

**Микротаски** – это задачи меньшего приоритета, которые выполняются сразу после текущего выполнения скрипта, но до начала следующей макротаски.

Promise.then/catch/finally

 JavaScript выполняет код синхронно в **стеке вызовов (Call Stack)**.

 Когда встречаются асинхронные задачи (таймеры, сетевые запросы, промисы), они отправляются в соответствующие очереди.

 После завершения синхронного кода, Event Loop начинает обрабатывать микротаски (если они есть).

 После выполнения всех микротасок, Event Loop переключается на макротаски и обрабатывает первую из очереди.

Но может быть такое что выполнялись все микротаски, стек микротасков пустой, начинает выполнятся макротаски. Макротасков к примеру штук 3. Выполняется первый, внутри этого макротаски есть промис микротаск. Макротаск зародил миротаск, этот миротаск отпарвляется в очередь микротасков. Но стоит заметить что макротаск 2 не выполняется следующим, выполняется микротаск, который только что зародился. После него уже выполняется макротаск 2.

1. **Что такое Promise?**

Promise – это специальный объект, который содержит своё состояние. Вначале это всегда Pending, затем в зависимости ответа, fulfilled – выполнено успешно или же Rejected – выполнено с ошибкой.

В Promise можно навешить два коллбэка resolve, rejected.

У Promise есть три обработчика, then – когда выполнено успешно или с ошибкой, оно задает сценарий какие действие предпринять при ответе. Также есть catch, обработчик только ошибок. Обрабатывает только на rejected. И в конце есть finally, обрабатывает в любом случае, какой бы ответ не был.

1. **Что такое Promise.all?**

Это метод который принимает массив Promisов и возвращает новый Promise. Оно возвращает только тогда когда все переданные в него промисы обработали успешно или когда один из них вышел с ошибкой.

1. **Что такое async/await?**

Async/await – это синтаксический сахар в JS, который позволяет работать с асинхронным кодом более удобно, делая его более читаемым и похожим на синхронный. Он строится на основе промисов и упрощает их использование.

Async используется для объявление асинхронной функции.

Await используется внутри асинхронной функции для приостановки выполнения функции до тех пор, пока промис не завершится (resolve, reject).

Когда промис завершится await вернет его результат.

Это позволяет писать асинхронный код так будто он синхронный, без использование .then().

Также вместо .catch() можно использовать try/catch, что делает обработку ошибок более удобной.

### 4. Какой тип наследования используется в JavaScript?

В JavaScript прототипное наследование проще и гибче, чем наследование классов. Прототипное наследование реализовано одной конструкцией: объектами. Каждый объект имеет внутреннюю ссылку на другой объект, который является его прототипом. Если объект B является прототипом объекта A, то всякий раз, когда B имеет свойство, такое как цвет, A наследует тот же цвет, если не указано иное. A может быть унаследован от B. B, в свою очередь, может быть унаследован от C. Это называется цепочкой прототипов.

1. **Что такое HTTP?**

HTTP(Hypertext Transfer Protocol) – это протокол для передачи данных в Интернете. Он работает как “вопрос-ответ”, на каждый вопрос серверу, он так или иначе возращает нам некий ответ.

Он имеет такие методы как:

GET – получить данные

POST – создать данные

PUT – полностью обновить данные

PATCH – частично обновить данные

DELETE – удалить данные

Также у него есть статусы ответа такие как 200 – успешный ответ, 400 – ошибка со стороны отправителя, 500 – ошибка со стороны сервера.

Также есть HTTPS это защищенный протокол.

1. **Что такое JSON?**

JSON(JavaScript Object Notation) – это легковесный формат передачи и получение данных. Имеет формат ключ-значение. Может хранить в себе данные также с разными типами string, number, Boolean, array, object.

1. **Деструктуризация объекта?**

Деструктуризация – относительно новый способ извлечение значений объекта или массива.  
Использование деструктуризации позволяет сделать код чище и отнимает меньше времени. Синтаксис деструктуризации следующий: заключаем свойства объекта, которые хотим получить, в фигурные скобки ({ }), а если речь идет о массиве — в квадратные скобки ([ ]):

let { firstName, lastName, position, yearHired } = employee.

1. **Что такое LocalStorage?**

LocalStorage — это встроенное в веб-браузеры хранилище, которое позволяет сохранять данные на стороне клиента (в браузере) в виде пар ключ-значение. Это хранилище предоставляет возможность сохранять данные в браузере без срока действия (то есть данные будут сохраняться даже после закрытия браузера или перезагрузки компьютера).

Основные характеристики LocalStorage:

Асинхронность: Доступ к данным осуществляется синхронно, что означает, что операция чтения или записи данных завершается немедленно.

Объем хранилища: Обычно браузеры предоставляют около 5-10 МБ для LocalStorage, хотя точный размер может варьироваться.

Долговечность данных: Данные хранятся в LocalStorage до тех пор, пока они не будут удалены вручную (например, через JavaScript) или пользователем через настройки браузера.

Доступ к данным: Данные доступны только для домена, на котором они были сохранены (политика одного источника), и не могут быть доступны с других доменов.

1. **Что такое AJAX?**

**AJAX (Asynchronous JavaScript and XML)** — это технология, позволяющая веб-приложениям асинхронно загружать данные с сервера без необходимости перезагружать всю страницу.

1. **Что такое Generics?**

**Генерики** (или обобщения) — это механизм, который позволяет параметризировать типы данных. Это означает, что вы можете написать функцию или класс, которые могут работать с разными типами, не теряя информации о типах и не дублируя код для каждого типа.

1. **Что такое Итераторы и Генарторы?**

**Итераторы** — это объект с методом next(), который возвращает следующий элемент последовательности.

**Генераторы** — это функции обозначенные с \*, которые могут приостанавливать свое выполнение и возвращать значения по мере необходимости с использованием ключевого слова yield.

1. **Что такое прототип объекта?**

Прототип – это механизм, с помощью которого объекты могут наследовать свойста и методы друг от друга. Каждый объект в JS имеет скрытую ссылку на другой объект.

Если при вызове какого-то свойства или метода, мы не на находим его в дочернем объекте, оно идет искать его дальше, и дойдет до Родительского объекта прототипа, у которого есть глобальные/родительские методы доступные всем объектам hasOwnProperty, toString, valueOf, toLocaleString()

1. **В чем разница между методами функции call и apply?**

Apply и Call используются для привязки определенного объекта к значению this вызываемой функции.

Отличие между ними что apply аргумент передаются в виде массива, а в call – через запятую.

1. **Для чего используется метод Function.prototype.bind?**

Bind возвращает новую функцию, значеием this которой является объект, указанный в качестве первого параметра. В отличие от bind, call и apply сразу же вызывает функцию.

1. **Какие методы массивов есть?**

**forEach** – это цикл, для итерации по элементам массива.

Тоже самое что и **for(let i=0; I < people.lenght; i++)** в ES5

или **for(let person of people)** в ES6

**Map** – это метод, который выполняет функцию для каждого элемента массива, но в отличие от forEach, и возвращает новый массив с результатами этой функции.

forEach больше подходит, когда надо просто пробежаться по массиву, а map чтобы в дальнейшнем модифицировать элементы массива.

**Filter** – это метод массива, который вовзращает только те элементы массива, который возвращают значение true при какой-то определенной условии, которые мы задали.

**Reduce** – это метод массива, который который накапливает в первый аргумент значение, который накапливается за чем второго аргумента.

**Find** – метод, который находит элемент или элементы, который равны значению которые мы передали

**FindIndex** – тоже, самое что и Find, но находит его индекс.

"На данный момент я работал только с монолитной архитектурой, но я хорошо понимаю концепцию микросервисов, их преимущества, такие как масштабируемость, модульность и независимость разработки, а также их вызовы, например, управление распределёнными системами и обеспечение связи между сервисами."

1. **Что такое DOM?**

DOM(Document Object Model) – это программный интерфейс для HTML и XML документов. Он представляет структуру веб-страницы в виде дерева, где каждый элемент является объектом, с которым можно манипулировать с помощью программных средств, таких как JavaScript.

DOM позволяет нам взаимодействовать с содержимым страницы: изменять элементы, добавлять новые, удалять, изменять стили, навешивать слушатели. Элемент объвялен в виде тега(div, h1) и у него есть атрибуты(id, class). HTML – это то что мы объявлем внутри документа, а XML внтури React.

1. **Что такое Семантическая верстка?**

**Семантическая верстка** — это подход к разработке веб-страниц, при котором используются HTML-теги, отражающие смысл (семантику) контента. Это означает, что вместо использования общих тегов, таких как <div> и <span>, используются теги, которые четко описывают содержание и структуру страницы, что делает код более понятным и доступным как для людей, так и для машин (например, поисковых систем и вспомогательных технологий).

**Основные характеристики семантической верстки:**

1. **Использование семантических тегов**:
   * Применяются теги, которые описывают содержимое страницы. Например:
     + <header> — для заголовка страницы или секции.
     + <nav> — для навигационного меню.
     + <article> — для независимого содержимого, например, статьи или блога.
     + <section> — для разделения контента на логические части.
     + <aside> — для вспомогательной информации, которая не является основной, например, боковая панель.
     + <footer> — для нижнего колонтитула страницы или секции.
2. **Повышение доступности**:
   * Семантическая верстка улучшает доступность для пользователей с ограниченными возможностями. Теги помогают вспомогательным технологиям (таким как экранные считыватели) правильно интерпретировать структуру и смысл контента.
3. **Улучшение SEO**:
   * Семантическая разметка способствует улучшению поисковой оптимизации (SEO), так как поисковые системы лучше понимают, о чем страница и как структурирована информация. Это может повысить видимость сайта в результатах поиска.
4. **Читаемость кода**:
   * Использование семантических тегов делает код более понятным для разработчиков и упрощает его поддержку. Это особенно важно в команде, где разные люди могут работать над одним проектом.
5. **Структурирование контента**:
   * Семантическая верстка помогает создать четкую структуру страницы, что упрощает восприятие информации пользователями и помогает им легче находить нужные разделы.
6. **Что такое display?**

Свойства CSS display управляют тем, как элементы отображаются на веб-странице inline, block, inline-block, flex и grid.

Inline – не имеет ширины и высоты, это описание отображение текста и таких тегов как a string span

Block – имеет ширину и высоту, это описание отображение какого-то блока, у него есть margin, padding. Его теги div nav header.

Inline-block – это элементы которые имеют свойство inline, и block. Он может содержать себе строку inline и может иметь ширину и высоту как block. Используется к примеру для создание кнопок.

Flex – гибкий элемент, который разделяется на элементы внутри себя на контейнеры и дает гибкость их управление по горизонтали и вертикали. Дает возможность легко создавать адаптивную верстку.

Grid – элемент который управляет своими дочерными элементами в виде сетки.

1. **Что такое псевдоклассы?**

Это свойства которые закреплены в тег, такие как hover, after, before.

1. **Что такое React?**

React – это библиотека Javascript для создание пользовательских интерфейсов, разработанная компанией Facebook. Она позволяет разработчикам строить многоразовые компоненты. Основные особенности:

**Компонентный поход:**

Каждый компонент может быть представлен как отдельный кусочек UI, который можно переиспользовать в разных частях приложения.

**Виртуальный DOM:**

React использует Виртуальный DOM для повышения производительности. Вместо того чтобы обновлять целый DOM при каком-то изменений, React сначала обновляет Виртуальный DOM, отталкиваясь на его результатах вычисляет минимальное количество изменений который нужно внести в реальный DOM. Это делает обновление интерфейса более быстрым.

1. **Хуки в React?**

Хуки – это функции, которые работает с состоянями в Реакте.

Основные хуки:

useState – хранит в себе внутренне состояние компонента

useEffect – обрабатывает побочный эффект на какой-то сценарий

useNavigate – маршрутизирует на некий адрес

useLocation – передает и вызывается для сохранение данных между компонентами.

useRef – С помощью useRef можно привязать реф на некий элемент DOMа, что позволяет напрямую взаимодействовать с ним, например, для управления фокусом, изменения значений или выполнения анимаций.

useAppSelector, вызывает данные которые хранятся в глобальной хранилище

1. **Что такое Redux?**

Redux — это библиотека для управления состоянием приложений на JavaScript, особенно популярных в экосистеме React. Она предоставляет централизованное хранилище (store) для состояния приложения и предсказуемый способ управления этим состоянием, что делает его особенно полезным для больших и сложных приложений. Вот основные аспекты Redux:

**1. Централизованное хранилище**

* В Redux все состояние приложения хранится в одном объекте (store). Это облегчает отслеживание состояния и его изменений, поскольку все данные находятся в одном месте.

1. **Что такое Redux Tool Kit?**

Это билиотека для Redux, который предоставляет интрументы для запросов с какими-то параметрами, которые мы задали.

1. **Что такое event.stopPropagation()**

Метод в JS который предотваращет вызов родителя, если метод объявлен внутри элемента ребенка.

1. **Что такое event.PreventDefault()**

Предотваращает обработкий дефолтных действий бразуера.

1. **Что такое SASS и его преимущества.**

**SASS** – это препроцессор, который делает удобный для написание стилей, особенно в больших проектов. Он затем компилируется в обычный CSS.

**Фрагментирование и импорт** – можно какие-то стили фрагментировать с помощью \_нижнего подчеркивание, чтобы в дальнейшнем переиспользовать его в других частях стилизации импортируя его.

**Переменные** – можно задавать переменные, например для цветов, чтобы при смене этой переменной, менялись все цвета элементов, у которых в стилях указана эта переменная.

**Вложенность** – есть иерехия, с помощью которого через & амперсант, мы делаем вложенноть HTML элементов.

**Миксины** – объявляется через = знак равенства, вызываем через + плюс. внутри можно записывать переменные. Это когда унас есть повторяющийся элемент, у которого похожие параметры стиля, но отличающиеся по значению. Также можно туда передавать некий конетент через @.

**Наследование** – можно наследовать стиль, который мы ранее прописали, с помощью @extend, также можно наследовать переменную с значениями стилей, который обозначены знаком % - процента.

**Вложенные свойства** – при таком синтаксисе стилей как у font, чтобы каждый раз неповторять его, можно наследовать его с помощью отсутпа.

1. **Что такое props?**

Это передача данных с родительского элемента к дочернему.

1. **Что такое управляемые и неуправляемые компоненты?**

**Управляемые компоненты** – это компонент, в котором React контролирует поля ввода через состоние, useState например через обработчик событий.

**Неуправляемые компоненты** – это компонент, в котором React не контролирует значение, а хранится напрямую в DOM. Вместо сстояния для доступа к значению используется ref.

1. **Что такое методы жизненного цикла компонента?**

Методы жизненного цикла компоненты – это специальные методы которые вызываются на разных стадиях жизни компонента.

Сначал идет Инициализация, обработка всех данных

Потом монтирование, монтируется в DOM элемент

Обновление данных, когда мы как то их модифицируем, обновляем

Также размонтирование, когда мы уходим из этого компонента.

1. **Что такое JSX?**

JSX – это расширенние синтаксиса JS для создание React элементов.

Есть также TSX – тоже самое, но на TypeScriptе.

1. **Что такое условный рендеринг элементов?**

Это когда прорисовывается те данные, которые соответсуют условиям тернарного или условного оператора.

1. **Что такое фрагмент React?**

Это отрисовка нескольких компонентов внутри одного компонента.

1. **Что такое React Reconciliation?**

React использует алгортим согласования (reconciliation) для опеределения, сравнивает изменения между реальным DOM и вируатльным DOM, затем изменяет только только те элементы, которые были изменены, удалены и модифицированны. Это повышает производительность.

1. **Зачем нужен key при отсировке элементов массива через map?**

Во время прорисовки элементов массива, лучше к каждому элементу привязывать какой-то уникальный ключ, например id. Благодаря этому React четко понимать какой элемент был модифицирован, изменен, удален.

1. **Что такое useMemo?**

Хук useMemo используется для запоминвание/кэширование данных. В него передаются функции и массив зависимости. Оно сделает вычислительные операции только один раз и запомнит его, затем будет занова использоват эти данные и поставлять к DOM элементу, нежели занова будет проводить эти вычислительные операции. Это нужно в том случае если вычислительные операции функции масштабные и его каждая итерация вредит производительности, при мелких функциях лучше его не использовать. useCallback идентичен и по синтаксису, не передает функцию.

1. **Что такое useContext?**

Хук useContext нужен для передачи данных через несколько стадии вложенности в разные компоненты, чтобы не передавать данные через пропсы, и нагружать компненты которому эти данные не нужны. Оборачиваем компонент в названиеКомпонента.Provider, которую создали через createContext и вызываем его в нужном нам компоненте через useContext.

1. **Схлопование**

Когда два блока с внешним отступом паддингом 30 и 20 встречаются, будет не 50, а 30. Можно предотворить с помощью бордеров или с маржинами внутренний отступ

1. **Позиционирование**

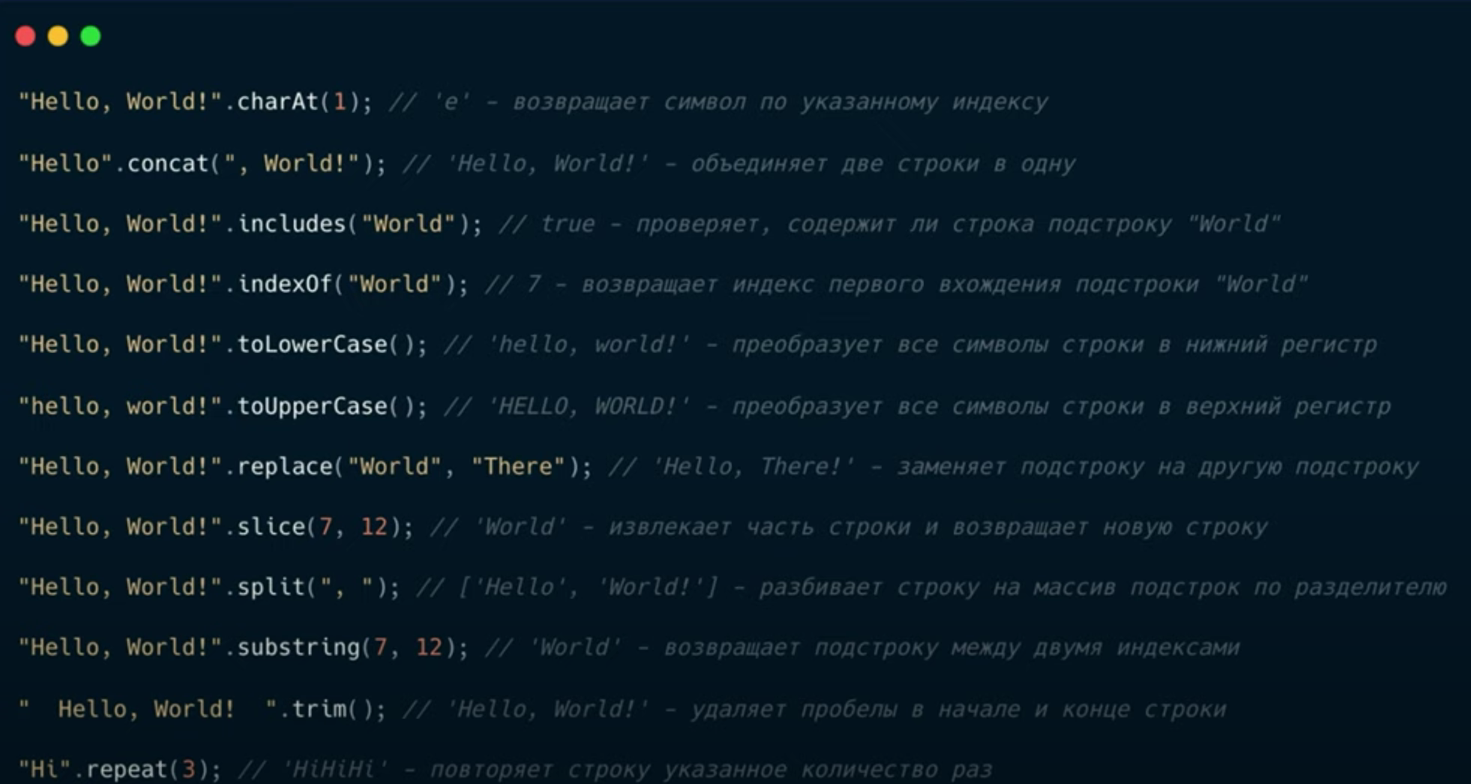
Все по умолчанию **Static**. **Relative** – может двигаться относительно от своей позиции, сохраняя невидимое место в свой позиции которую покинул. **Absolute** – позицинурется относительно от своего ближайшего родителя, у которого позиция relative, если такого нет, то от документа. **Fixed** – позиционруется относительно от окна браузера, всегда на месте при скролле.

1. **Что такое чистая функция?**

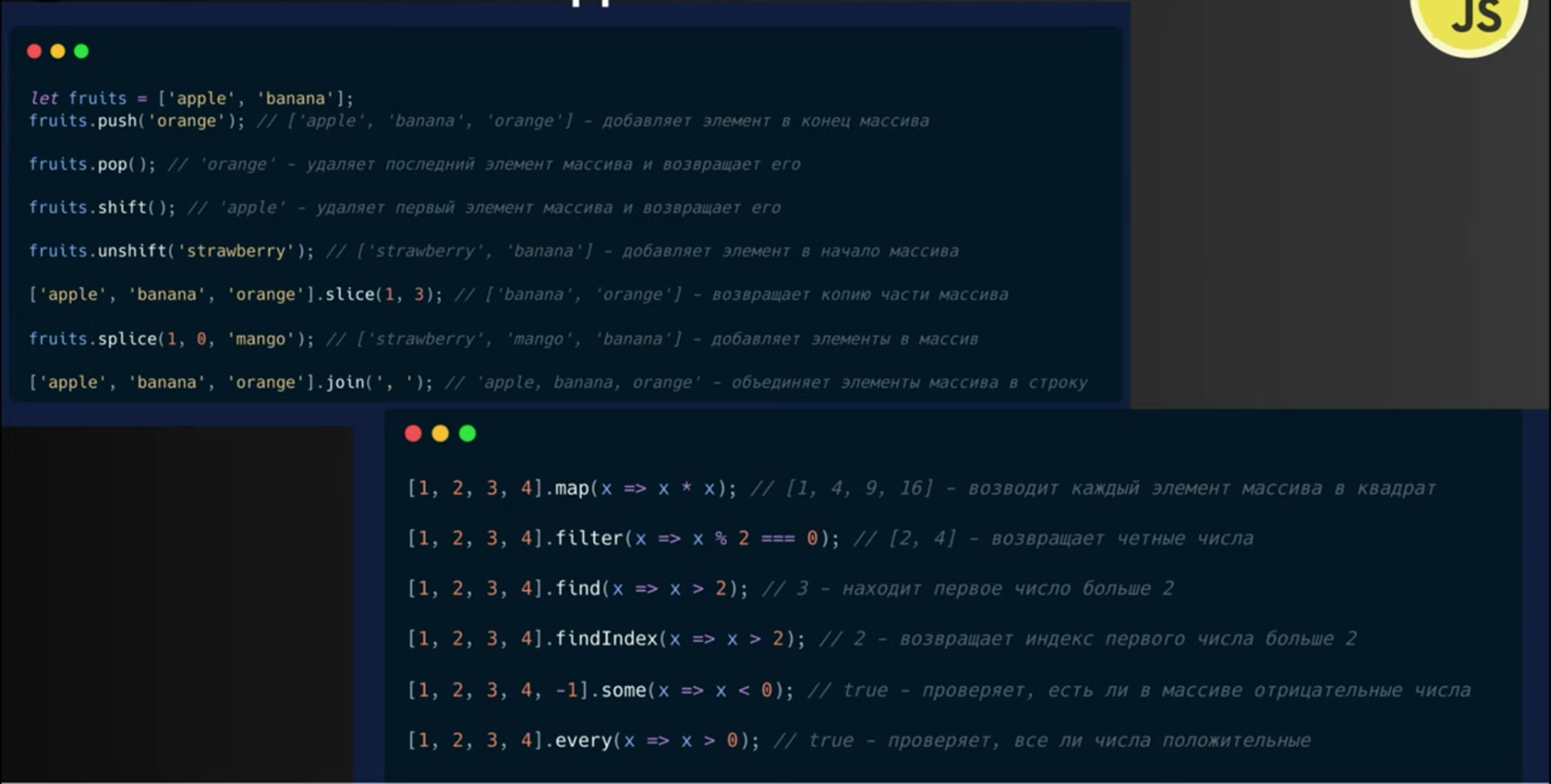
Чистая функция – не изменяет внешние данные. Всегда возвращает одинаковый результат для одинаковых внешних данных. Не имеет побочных эффектов.

Нечистая функция – Делает все наоборот, изменяет внешние данные переменных, результат может зависить от внешнего состояние, может влиять на внешний мир (DOM)

1. **Методы строк**



1. **Методы массивов?**



1. **Оператор Spread и Rest.**

**Spread** – расширяет массив или объект, добавляя новое свойство или значение в коней. Троеточие исходная переменная и новые данные [..sum, 1]

**Rest** – отделяет значение от переменной на две переменные.