1.Основы JavaScript:

1.1 

1.2 Область видимости:

var: Обладает функциональной областью видимости. Это означает, что переменная видна внутри функции, независимо от блока, в котором она объявлена.

let и const: Обладают блочной областью видимости. Переменные, объявленные с использованием let или const, видны только в блоке, в котором они объявлены (например, внутри {}).

Поднятие (hoisting):

var: Поднимается (hoisted) в начало своей области видимости. Это означает, что переменная может быть использована до ее фактического объявления.

let и const: Также поднимаются (hoisted), но не могут быть использованы до своего объявления. Это известно как "временная мертвая зона" (temporal dead zone).

Переопределение:

var: Может быть повторно объявлен в той же области видимости.

let: Не может быть повторно объявлен в той же области видимости.

const: Не только не может быть повторно объявлен, но и не может быть переопределен после присваивания значения.

Использование до объявления:

var: Может быть использована до объявления (возвращается undefined).

let и const: Вызывает ошибку, если пытаемся использовать переменную до ее объявления

1.3 Hoisting (поднятие) - это механизм в JavaScript, который поднимает объявления переменных и функций вверх по их области видимости перед выполнением кода. Это означает, что переменные и функции могут быть использованы даже до того, как они были фактически объявлены в коде. Однако, стоит отметить, что только объявления поднимаются, а не инициализации (присваивания значений).

2.1 В JavaScript существует несколько способов объявления функций:

Function Declaration (Объявление функции): Используется ключевое слово function, за которым следует имя функции. Функции, объявленные таким образом, поднимаются вверх в процессе hoisting.

Function Expression (Функциональное выражение): Создается путем присваивания функции переменной. Функции, созданные как выражения, не поднимаются вверх в процессе hoisting.

Arrow Function (Стрелочная функция): Введены в ECMAScript 6. Это краткая форма записи функций, особенно удобная для анонимных функций.

Named Function Expression (Именованное функциональное выражение): Функциональное выражение с именем. Имя функции доступно только внутри самой функции.

2.2

Главное различие между function declaration и function expression в JavaScript заключается в том, как они обрабатываются в процессе hoisting и доступности к переменной, которой присваивается функция.

3.2 Стрелочные функции (Arrow Functions) - это синтаксический сахар в JavaScript, представленный в ECMAScript 6 (ES6), который предоставляет более краткую и удобную форму записи функций. Они являются альтернативой обычному синтаксису функций.

Основные особенности стрелочных функций:

Краткий синтаксис:

Стрелочные функции предоставляют более краткий синтаксис для определения функций.

Отсутствие собственного this:

Стрелочные функции не создают собственный контекст this. Они наследуют this из окружающего контекста. Это может быть полезно в избежании некоторых проблем с this в функциях.

Не могут быть вызваны с new:

Стрелочные функции не могут быть использованы как конструкторы и не имеют своего собственного prototype.

Не содержат собственных arguments:

Стрелочные функции не имеют собственного объекта arguments. Они используют аргументы из своего внешнего контекста.

3.1 В JavaScript объект можно создать несколькими способами:

Литерал объекта: Используйте фигурные скобки {} и определите пары ключ-значение.

Конструктор Object: Используйте конструктор Object и ключевое слово new для создания пустого объекта.

Функция-конструктор: Определите функцию, которая выступает в роли конструктора объекта, и используйте ее с ключевым словом new.

Object.create: Используйте метод Object.create, который создает новый объект с указанным прототипом.

Использование классов (ES6): Определите класс с помощью ключевого слова class и используйте его для создания объектов.

3.2 Метод push: Метод push добавляет один или несколько элементов в конец массива и возвращает новую длину массива.

Оператор распространения (Spread Operator): Оператор распространения позволяет создать новый массив с добавленным элементом из существующего массива.

Присвоение по индексу: Вы можете присвоить новое значение по индексу, равному текущей длине массива.

3.3 Цикл for: Используйте цикл for для итерации по индексам массива и доступа к каждому элементу.

Цикл for...of: Цикл for...of предоставляет удобный способ перебора значений массива без явного использования индексов.

Метод forEach: Метод forEach является встроенным методом массива, который позволяет передать функцию обратного вызова для выполнения над каждым элементом.

Метод map: Метод map создает новый массив, применяя функцию к каждому элементу существующего массива.

Методы filter, reduce, some, every и другие: Встроенные методы массива также предоставляют специфические способы обработки итераций в зависимости от вашего намерения.

4.1 С использованием атрибута on: Добавьте атрибут on к элементу события и присвойте ему значение функции-обработчика.

С использованием метода addEventListener: Используйте метод addEventListener элемента для добавления обработчика события. Этот метод позволяет добавить несколько обработчиков для одного события.

С использованием свойства события внутри HTML (например, onclick): Присвойте функцию-обработчик напрямую свойству события элемента внутри HTML-кода.

4.2

Event delegation (делегирование событий) - это подход в веб-разработке, при котором один обработчик событий назначается родительскому элементу для обработки событий от всех его потомков. Вместо того чтобы назначать обработчики событий каждому отдельному элементу, обработчик размещается на общем предке, и события от потомков обрабатываются этим одним обработчиком.

4.3 Для отмены стандартного поведения события в JavaScript используют метод preventDefault(). Вызов этого метода на объекте события предотвращает выполнение стандартных действий, связанных с этим событием. Это может быть полезно, например, при предотвращении отправки формы при клике на кнопку submit или открытии ссылки по умолчанию при клике.

Также, в контексте обработчика события, возвращение false также приводит к вызову preventDefault(). Этот подход может использоваться в HTML-атрибутах событий.

5.1 Синхронный код выполняется последовательно от начала до конца, одна операция за другой. Каждая операция должна завершиться, прежде чем начнется следующая. Это означает, что выполнение кода блокируется до завершения текущей операции.

Асинхронный код, напротив, позволяет выполнять операции параллельно, без блокировки выполнения кода. Операции, которые могут занять время (например, запрос к серверу, чтение файла), выполняются асинхронно. Когда асинхронная операция запускается, она не блокирует остальной код, и выполнение продолжается. Позднее, когда асинхронная операция завершится, будет вызван обработчик или выполнена колбэк-функция.

Асинхронность в JavaScript обычно реализуется с использованием колбэков, промисов или асинхронных функций (async/await). Эти механизмы позволяют управлять асинхронным кодом и обрабатывать результаты после завершения операций.

5.2

Promise (обещание) - это объект в JavaScript, предназначенный для управления асинхронными операциями. Он представляет собой обещание о завершении асинхронной операции, которое может быть в состоянии "выполнено" (resolved), "отклонено" (rejected) или "ожидается" (pending).

5.3

async/await - это концепция в JavaScript, которая облегчает работу с асинхронным кодом.

Объявление функции с ключевым словом async: Помечайте функцию ключевым словом async для того, чтобы указать, что она содержит асинхронный код.

Использование ключевого слова await: Используйте await внутри асинхронной функции перед вызовом промиса, чтобы дождаться его выполнения.

Обработка ошибок с try/catch: Используйте блок try/catch для обработки ошибок в асинхронном коде.

Возвращение значений:

Если вы хотите вернуть результат асинхронной операции, просто возвращайте его из функции.