Выполнил: Андряков Матвей

Студент группы: ИСП-420п

1. Основы JavaScript

- Типы данных:

Типы данных в JavaScript		
Примитивные типы данных	**Объектные типы данных**	**Специальные типы данных**
Number (Число): представляет как целые, так и десятичные числа. String (Строка): представляет текстовые данные. Boolean (Логический): представляет значение true или false. Null: представляет отсутствие значения или значение "ничего". undefined: представляет переменную, которая объявлена, но не имеет присвоенного значения. Symbol (Символ): представляет уникальный и неизменяемый идентификатор (в ECMAScript 6).	Object (Объект): сложный тип данных, представляющий коллекцию ключ-значение. Array (Массив): представляет упорядоченный список значений. Function (Функция): представляет блок кода, который может быть вызван.	BigInt: представляет целые числа произвольной точности. Object (Null): представляет отсутствие значения объекта.

- В JavaScript существует три ключевых слова для объявления переменных: var, let и const.

var: обладает функциональной областью видимости, что означает, что переменная, объявленная с помощью var, видна внутри всей функции, в которой она объявлена, независимо от блочной структуры кода. Переменные, объявленные с помощью var, поднимаются вверх в начало своей области видимости. Это может привести к неожиданным результатам.

let: обладает блочной областью видимости, что означает, что переменная let видна только внутри блока, в котором она объявлена (например, внутри условия, цикла или функции). Переменные, объявленные с помощью let, также поднимаются вверх, но остаются в "зоне видимости" блока, в котором они объявлены.

const: также обладает блочной областью видимости. Как и let, переменная const видна только внутри блока, в котором она объявлена. Переменные, объявленные с помощью const, должны быть инициализированы при объявлении, и их значение не может быть изменено после инициализации. Это означает, что const используется для объявления постоянных (неизменяемых) переменных.

Таким образом, const, let – соблюдают блочную область видимости. var – игнорирует блочную область видимости. Переменные, объявленные таким способом будут видны за пределами блока.

- Hoisting – это механизм в JavaScript, который позволит сначала вызвать переменную, а потом передать в нее значения исходя из области видимости. Это работает с переменными и функциями. Он позволяет поднимать объявления переменных и функций вверх в текущей области видимости, что упрощает понимание и редактирование кода.

2. Функции

- Существует три способа объявить функцию в JavaScript.

Function Declaration (декларативное объявление): для любой функции, если не планируете записывать её в переменную или передавать в качестве аргумента другим функциями. Сначала указываем ключевое слово function. За ним идёт имя функции — в нашем случае greet. В круглых скобках пишем параметры, если они есть. Если параметров нет, оставляем просто — (). В фигурных скобках содержится код, который будет выполняться при вызове функции. Функция создается и присваивается переменной. Может быть анонимной (без имени) или именованной.

Function Expression (функциональное выражение): для функции, которую нужно записать в переменную. А ещё способ полезен, когда функция должна быть доступна только после её объявления. Объявляем переменную — у нас это greet. Затем присваиваем ей значение — функцию. Эта функция пишется так же, как и при декларативном объявлении. Но в отличие от него здесь имя функции чаще всего опускается — то есть она является анонимной. Используется ключевое слово function. Может быть анонимной или именованной.

Arrow Function (стрелочная функция): для любых функций, в которых нет контекста this. Стрелочные функции компактны. У них нет имени и ключевого слова function. Вместо них в круглых скобках сразу же пишутся параметры — (a, b), за которыми идёт стрелка =>, а после неё — фигурные скобки с телом функции. Краткая форма для объявления функций. Отсутствует собственный this.

- Разница между function declaration и function expression

Function Declaration: используется для объявления функции в верхней части своей области видимости. Можно вызвать функцию до её объявления в коде. Это объявление функции с использованием ключевого слова function. Функция может быть вызвана до того, как она появится в коде, благодаря поднятию. Имеет функциональную область видимости, доступную внутри блока кода, в котором функция объявлена.

Function Expression: создаёт функцию во время выполнения кода, как часть выражения. Нельзя вызвать функцию до её фактического объявления. то создание функции внутри выражения и присвоение её переменной.Функцию нельзя вызвать до того, как переменная будет инициализирована. Имеет блочную область видимости, если объявлена внутри блока кода.

- Стрелочные функции

Стрелочные функции (Arrow Functions) представляют собой новый синтаксис для определения функций в JavaScript, введённый в стандарте ECMAScript 2015 (ES6). Они предоставляют более краткую и выразительную форму для создания функций, особенно для простых случаев.

Основные особенности стрелочных функций:

Сокращенный синтаксис: стрелочные функции обладают кратким и лаконичным синтаксисом, что делает их удобными для использования, особенно при определении небольших функций.

Отсутствие собственного this: в стрелочных функциях нет своего собственного контекста this. Они наследуют значение this из окружающего контекста, в котором они были созданы. Это может быть полезно в избежании некоторых трудностей с this в обычных функциях.

Не могут быть использованы как конструкторы: стрелочные функции не могут быть вызваны с использованием new и не имеют своего собственного объекта prototype, что ограничивает их использование в качестве конструкторов.

Стрелочные функции часто применяются в ситуациях, где нужно создать небольшие анонимные функции или когда нужно избежать проблем с контекстом this.

3. Объекты и массивы

- Создание объектов в JavaScript

Литерал объекта: это самый простой способ создания объекта, используя фигурные скобки {} и пары ключ-значение.

Создание объекта через конструктор: можно использовать конструктор Object() для создания пустого объекта и добавления свойств и методов после создания.

Использование функции-конструктора: можно создать объект, используя функцию в качестве конструктора с помощью оператора new.

Использование классов (ES6 и выше): современный способ создания объектов, представленный в ES6, — это использование классов.

Создание с использованием метода Object.create(): можно создать объект с использованием другого объекта в качестве прототипа.

- Добавление нового элемента в конец массива

В JavaScript для добавления нового элемента в конец массива можно использовать метод push().Метод push() добавляет один или более элементов в конец массива и возвращает новую длину массива. Он принимает неограниченное количество аргументов, позволяя добавить сразу несколько элементов. Кроме того, push() возвращает новую длину массива, что может быть полезно в некоторых случаях.

- Перебор всех элементов массива

Цикл for: используется обычный цикл for для итерации по индексам массива.

Цикл for...of (ES6 и выше): используется цикл for...of для более краткого синтаксиса итерации.

Метод forEach(): используется метод forEach(), который предоставляет функцию обратного вызова для каждого элемента массива.

Метод map(): используется метод map(), который также применяет функцию к каждому элементу массива.

4. Обработка событий

- Назначение обработчика событий элементу

С использованием свойства on

С использованием метода addEventListener: ветод addEventListener предпочтителен, так как он позволяет назначать несколько обработчиков для одного и того же события и элемента. Также, при использовании addEventListener, вы можете легко удалить обработчик события с помощью метода removeEventListener.

Используя атрибут HTML (onclick)

С использованием свойства DOM-элемента (onclick)

Используя свойство DOM-элемента (onmouseover для события наведения курсора).

- Event delegation

Event delegation (Делегирование событий) - это паттерн в программировании на JavaScript, который используется для эффективного управления обработкой событий для группы элементов через их общего родителя. Вместо того чтобы присваивать обработчики событий каждому элементу отдельно, делегирование событий позволяет назначить один обработчик на родительский элемент.

Принцип работы event delegation: добавление обработчика события на родительский элемент, использование свойства события (event) для определения целевого элемента, обработка события в зависимости от целевого элемента.

Преимущества event delegation:

A) Эффективность: позволяет снизить количество обработчиков событий в приложении, особенно для больших списков элементов.

B) Динамическое добавление элементов: работает даже с элементами, добавленными динамически после загрузки страницы.

C) Уменьшение нагрузки на память: меньше обработчиков событий означает меньше занимаемой памяти.

D) Этот подход особенно полезен при работе с динамическим содержимым и уменьшает сложность управления событиями.

- Отмена стандартного поведения события

Чтобы отменить стандартное поведение события в JavaScript, можно использовать метод preventDefault() объекта события (event). Этот метод используется для предотвращения выполнения действия, которое обычно выполняется по умолчанию в ответ на событие.

Этот метод полезен при работе с формами, гиперссылками и другими элементами, где требуется предотвратить стандартное действие и выполнить свой код вместо него.

Отмена действия по умолчанию производится с помощью объекта Event . Для этого у него есть специальный метод preventDefault() , который следует вызвать в любом месте обработчика события.

5. Асинхронность

- Работа синхронного и асинхронного кода

JavaScript является однопоточным (single-threaded) языком программирования, что означает, что у него есть только один основной поток выполнения кода. Это также означает, что код выполняется последовательно от начала до конца, и любая операция может блокировать выполнение кода, ожидая завершения.

Синхронный код:

В синхронном коде каждая операция выполняется последовательно. Код ожидает завершения одной задачи, прежде чем перейти к следующей. Когда функция или операция вызывается, выполнение кода блокируется, и программа ожидает завершения этой задачи, прежде чем продолжить.

В асинхронном коде некоторые операции выполняются независимо от основного потока выполнения. Вместо блокировки выполнения кода, асинхронные операции могут выполняться параллельно. Это означает, что код может продолжать выполнение, не дожидаясь завершения некоторых операций. Когда асинхронная операция завершится, будет вызван соответствующий обработчик. Это позволяет улучшить производительность и реактивность программы, особенно при работе с операциями ввода-вывода, таймерами или сетевыми запросами.

Обработка асинхронного кода может осуществляться с использованием колбэков, промисов или асинхронных функций, которые обеспечивают удобный способ работы с асинхронными операциями и избегания "callback hell" (цепочек колбэков).

- Promise

Promise - это объект в JavaScript, предназначенный для работы с асинхронными операциями. Promise представляет собой концепцию, которая обещает выполнить некоторую операцию в будущем и предоставляет удобный интерфейс для обработки её завершения (выполнения) или ошибки.

Основные состояния Promise:

Pending (Ожидание): исходное состояние, когда операция ещё не завершена.

Fulfilled (Выполнено): состояние, когда операция завершена успешно.

Rejected (Отклонено): состояние, когда операция завершена с ошибкой.

Основные методы Promise:

then(): вызывается при успешном выполнении операции и получает результат.

catch(): вызывается при ошибке в выполнении операции и получает ошибку.

finally(): вызывается в любом случае, после завершения операции (выполнения или ошиби).

Promise позволяет структурировать код для более читаемой и управляемой обработки асинхронных операций, а также избежать "callback hell" при использовании колбэков.

- Использование async/await

async/await в JavaScript предоставляют удобный синтаксис для работы с асинхронным кодом.

async функции:

Объявление: когда вы объявляете функцию с ключевым словом async, это означает, что функция возвращает Promise и может содержать операторы await.

Promise: функция всегда возвращает Promise, даже если она возвращает не-Promise значение. Если функция возвращает значение, это значение обернется в Promise, который будет успешно выполнен.

await оператор:

Ожидание: когда await используется внутри асинхронной функции, выполнение функции приостанавливается, ожидая завершения Promise. Когда Promise завершается, значение Promise извлекается.

Несинхронность: await делает код асинхронного кода более похожим на синхронный. Вместо использования колбэков или цепочек .then(), вы можете использовать await для ожидания завершения асинхронных операций.

try/catch для обработки ошибок:

try: Код, который может вызвать ошибку, помещается в блок try.

catch: Если в блоке try происходит ошибка, выполнение кода переходит в блок catch, где можно обработать ошибку.

Улучшенная читаемость:

async/await делают код более читаемым и структурированным, особенно при работе с множеством асинхронных операций.

Ошибки обрабатываются ближе к месту их возникновения, что облегчает отладку и понимание кода.

Последовательность выполнения:

Асинхронные функции с использованием await выполняются последовательно, что упрощает восприятие потока кода.

В целом, async/await предоставляют удобный и понятный способ работы с асинхронным кодом, сокращая количество шаблонов колбэков и повышая читаемость кода.