1. **Переменные. Имена переменных. Константы.**
   * **Переменные** используются для хранения данных в JavaScript. Вы можете объявить переменные с помощью ключевых слов **var**, **let** и **const**.
   * **Имена переменных** должны начинаться с буквы, знака подчеркивания (\_) или доллара ($), и могут содержать буквы, цифры и знаки подчеркивания.
   * **Константы** объявляются с помощью **const** и предназначены для хранения неизменяемых значений.
2. **Типы данных. Оператор typeof.**
   * JavaScript имеет разнообразные типы данных, такие как числа, строки, булевы значения, объекты и другие. Оператор **typeof** позволяет определить тип данных переменной. Например, **typeof 42** вернет **"number"**, а **typeof "Hello"** вернет **"string"**.
3. **Преобразование типов.**
   * Преобразование типов может быть явным (когда вы явно указываете, какой тип данных использовать) и неявным (когда JavaScript автоматически преобразует типы). Например, вы можете использовать **Number("42")** для явного преобразования строки в число.
4. **Операторы. Бинарные и унарные операторы. Приоритет операторов. Совмещение операторов.**
   * **Бинарные операторы** работают с двумя операндами, например, **+** для сложения. **Унарные операторы** работают с одним операндом, например, **-** для смены знака числа.
   * **Приоритет операторов** определяет порядок выполнения операций. Например, умножение (**\***) имеет более высокий приоритет, чем сложение (**+**).
   * **Совмещение операторов** позволяет выполнять несколько операций в одном выражении, например, **x = a + b - c**.
5. **Операторы сравнения. Сравнение с null и undefined.**
   * Операторы сравнения (**==** и **===**) используются для сравнения значений. **==** выполняет неявное преобразование типов, тогда как **===** не выполняет преобразование типов. Сравнение с **null** и **undefined** может быть выполнено с использованием этих операторов.
6. **Условные операторы: if, '?'. Конструкция switch.**
   * **if** используется для выполнения блока кода, если условие истинно.
   * Тернарный оператор **?** позволяет выполнить разные действия в зависимости от условия.
   * Конструкция **switch** позволяет выбирать различные варианты выполнения кода в зависимости от значения выражения.
7. **Логические операторы:**
   * Логические операторы (**&&**, **||**, **!**) используются для выполнения операций над булевыми значениями. **&&** выполняет логическое "И", **||** - логическое "ИЛИ", **!** - логическое "НЕ". Логические операторы могут быть использованы для составления сложных условий.
8. **Циклы while и for:**
   * **while** и **for** - это циклы, которые позволяют выполнить блок кода несколько раз. **while** выполняет код, пока условие истинно, а **for** предоставляет более удобный способ для работы с счетчиками и условиями цикла.
9. **Функции (Function Declaration). Параметры по умолчанию:**
   * Функции в JavaScript объявляются с использованием ключевого слова **function**. Вы можете определить параметры функции и использовать их внутри функции. Параметры по умолчанию позволяют задать значения по умолчанию для параметров, которые могут не быть переданы при вызове функции.
10. **Функциональные выражения (Function Expression) и функции-стрелки:**
    * Функциональные выражения позволяют определять функции как значения переменных. Функции-стрелки (**=>**) - это сокращенный синтаксис для объявления функций, который обычно используется для анонимных функций.
11. **Числа. Способы записи числа. Системы счисления. Методы типа Number: преобразование к числу, округление, проверка специальных числовых значений:**
    * В JavaScript числа можно записать в форме целых чисел, чисел с плавающей точкой, экспоненциальной записи и других форматах. JavaScript поддерживает десятичную и шестнадцатеричную системы счисления. Методы типа **Number** позволяют преобразовывать значения в числа, округлять числа, и проверять специальные числовые значения, такие как **NaN** и **Infinity**.
12. **Строки. Методы типа String: изменение регистра, поиск подстроки. Сравнение строк:**
    * Строки в JavaScript могут быть объявлены в одинарных или двойных кавычках. Методы типа **String** позволяют изменять регистр строки, находить подстроки, сравнивать строки и выполнять множество других операций.
13. **Массивы:**

* Массивы в JavaScript используются для хранения упорядоченных списков данных. Массивы могут содержать элементы различных типов данных и имеют числовые индексы.

1. **Методы массивов: добавление, удаление и замена элементов, объединение массивов, поиск в массиве:**

* Массивы в JavaScript предоставляют разнообразные методы для добавления (**push**, **unshift**), удаления (**pop**, **shift**), замены элементов (**splice**), объединения массивов (**concat**) и поиска элементов (**indexOf**, **includes**) и других операций.

1. **Методы перебора и преобразование массива:**

* Методы перебора массива, такие как **forEach**, **map**, **filter**, **reduce**, позволяют выполнить действия над каждым элементом массива или создать новый массив на основе существующего.

1. **Объекты. Литералы и свойства. Вычисляемые и короткие свойства. Проверка существования свойства. Перебор и упорядочение свойств объекта:**

* В JavaScript объекты используются для хранения данных в виде пар ключ-значение. Объекты могут быть созданы с использованием литералов, а также могут иметь вычисляемые и короткие свойства. Для проверки существования свойств объектов используются методы **hasOwnProperty** и оператор **in**. Перебор свойств объекта можно осуществить с помощью цикла **for...in**.

1. **Копирование, клонирование, сравнение, объединение объектов. Объекты-константы:**

* Копирование и клонирование объектов может быть выполнено с использованием различных методов, включая **Object.assign**, оператор расширения **{...}** и библиотеки для глубокого клонирования. Для сравнения объектов можно использовать рекурсивное сравнение. Объекты-константы создаются с использованием **Object.freeze**, что предотвращает изменение свойств объекта.

1. **Тип данных Symbol:**
   * Тип данных **Symbol** в JavaScript представляет уникальное и неизменное значение, которое может быть использовано в качестве ключей в объектах. Символы полезны, когда вам необходимо создать уникальные имена свойств.
2. **Коллекции Set, WeakSet:**

* Коллекция **Set** предоставляет уникальные значения без дубликатов. Она поддерживает методы для добавления, удаления и проверки наличия элементов.
* Коллекция **WeakSet** подобна **Set**, но она позволяет только хранить объекты и не предотвращает удаление объектов из памяти, если на них больше нет ссылок.

1. **Коллекции Map, WeakMap:**

* Коллекция **Map** представляет собой ассоциативный массив, где ключи могут быть любого типа данных. Она предоставляет методы для установки, получения и удаления значений по ключу.
* Коллекция **WeakMap** аналогична **Map**, но не предотвращает удаление ключей из памяти, если на них больше нет ссылок.

1. **Деструктурирующее присваивание. Вложенная деструктуризация:**

* Деструктурирующее присваивание позволяет извлекать значения из массивов и объектов и присваивать их переменным. Вложенная деструктуризация позволяет извлекать значения из вложенных структур.

1. **Дата и Время. Объект Date. Получение и установка компонентов даты. Автоисправление даты. Разность дат:**

* Объект **Date** в JavaScript используется для работы с датой и временем. Вы можете получать и устанавливать компоненты даты (год, месяц, день и так далее) и выполнять операции с датами, такие как вычисление разности между ними.

1. **Глобальный объект. Создание функции с помощью конструктора (new Function):**

* Глобальный объект в браузере - это **window**, в Node.js - это **global**. Вы можете создавать функции динамически с помощью конструктора **Function**. Это позволяет создавать функции из строкового представления JavaScript-кода.

1. **Лексическое окружение (LexicalEnvironment). Замыкание:**

* Лексическое окружение - это внутренний механизм JavaScript, который отвечает за связь переменных с их значениями. Замыкание возникает, когда функция имеет доступ к переменным из окружения, в котором она была создана, даже если это окружение больше не существует.

1. **Объект функции. Именованное функциональное выражение (Named Function Expression):**

* Объект функции представляет функцию как объект. Именованное функциональное выражение - это функциональное выражение, которому присвоено имя. Имя функции может быть использовано внутри функции и помогает в отладке.

1. **Остаточные параметры и оператор расширения:**

* Остаточные параметры (Rest parameters) позволяют функции принимать переменное количество аргументов в виде массива. Оператор расширения (Spread operator) позволяет развернуть элементы массива или объекта для передачи их в функцию или объединения.

1. **Каррирование и частичное применение функции:**

* Каррирование - это техника, при которой функция с несколькими аргументами преобразуется в последовательность функций с одним аргументом. Частичное применение - это процесс создания новой функции, фиксируя некоторые из аргументов.

1. **Генераторы. Функции-генераторы. Перебор объектов-генераторов:**

* Генераторы - это специальный тип функций, которые позволяют приостанавливать своё выполнение и восстанавливать позже. Они используют ключевое слово **yield**. Генераторы могут быть использованы для итерации по данным в ленивом режиме.

1. **Методы объектов, this. Оператор опциональной последовательности:**

* Методы объектов - это функции, связанные с объектами, и могут использовать ключевое слово **this** для доступа к свойствам объекта. Оператор опциональной последовательности (**?.**) позволяет избегать ошибок, когда пытаемся обратиться к свойству объекта, которое может быть **null** или **undefined**.

1. **Итерируемые объекты. Псевдомассивы:**

* Итерируемые объекты - это объекты, которые могут быть перебраны с использованием циклов, например, **for...of**. Псевдомассивы - это объекты, которые имеют числовые индексы и свойство **length**, но не являются настоящими массивами.

1. **Преобразование объектов:**

* Преобразование объектов может быть выполнено с использованием методов **toString()** и **valueOf()**. Эти методы позволяют определить, как объект должен быть представлен в виде строки или числа.

1. **Создание объектов через "new":**

* Объекты могут быть созданы с использованием оператора **new** в связке с конструктором. Например, **new Object()** создает новый пустой объект, а **new Date()** создает объект типа **Date**.

1. **Флаги и дескрипторы свойств:**

* Флаги и дескрипторы свойств объекта могут быть использованы для определения доступа к свойствам и их характеристик, таких как возможность чтения (**writable**), возможность перечисления (**enumerable**), и возможность изменения (**configurable**).

1. **Геттеры и сеттеры:**

* Геттеры и сеттеры - это специальные методы объекта, которые позволяют контролировать доступ к свойствам. Геттер используется для получения значения свойства, а сеттер для установки значения.

1. **Декораторы. Методы call(), apply():**

* Декораторы - это паттерн, который позволяет добавлять функциональность к объектам или функциям. Методы **call()** и **apply()** используются для вызова функций с явным указанием контекста и аргументов.

1. **Привязка контекста:**

* Привязка контекста в JavaScript позволяет задать значение **this** для функции. Это может быть выполнено с использованием методов **bind()**, **call()**, и **apply()**, что позволяет управлять, к какому объекту привязан контекст выполнения функции.