Ирина Кириченко

100 вопросов на собеседование по JavaScript

с подробными объяснениями

Ирина Кириченко

100 вопросов на собеседование по JavaScript. С подробными объяснениями

http://www.litres.ru/pages/biblio_book/?art=70129267
ISBN 9785006204843

Аннотация

Это практическое руководство представляет собой сборник вопросов и ответов по JavaScript, охватывающих ключевые аспекты языка, важные для подготовки к техническим собеседованиям. От основ до актуальных тем, книга предоставляет обзор основных вопросов, позволяя вам обновить и проверить свои знания. Здесь вы найдете не просто ответы, а инструменты и стратегии для уверенного прохождения собеседований по JavaScript.

Содержание

100 вопросов на собеседование по JavaScript С подробными объяснениями

Ирина Кириченко

ISBN 978-5-0062-0484-3
Создано в интеллектуальной издательской системе Ridero

Быстрый старт

100 ВОПРОСОВ

НА СОБЕСЕДОВАНИЕ

по JavaScript с подробными объяснениями

И. Кириченко

Это практическое руководство представляет собой сборник вопросов и ответов по JavaScript, охватывающих ключевые аспекты языка, важные для подготовки к техническим собеседованиям. От основ до актуальных тем, книга предоставляет обзор основных вопросов, позволяя вам обновить и проверить свои знания. Здесь вы найдете не просто ответы, а инструменты и стратегии для уверенного прохождения собеседований по JavaScript.

Кроме того, это руководство будет полезно для тех, кто хочет освежить свои знания или разобраться в непонятных для себя особенностях языка. Вы найдете здесь не только ответы на типичные вопросы, но и разъяснения сложных концепций, часто встречающихся на собеседованиях. Практические примеры и подробные объяснения помогут вам не только запомнить ключевые моменты, но и понять, как их эффективно применять в реальных проектах.

JavaScript: Ответы на вопросы

1. Что такое JavaScript?

JavaScript – это высокоуровневый, объектно-ориентированный язык программирования, который используется для создания интерактивных веб-страниц. Он позволяет добавлять различные функциональные возможности на сайты, взаимодействовать с пользователем, обрабатывать данные, анимировать элементы и многое другое. JavaScript является одним из основных инструментов веб-разработки и используется во множестве современных веб-проектах.

2. Как объявить переменную в JavaScript?

Для объявления переменной в JavaScript используются операторы let, const, и (устаревший) var.

let x = 5; // объявление переменной x с помощью let

const PI = 3.14159; // объявление константы PI с помощью const

var y = "Пример"; // объявление переменной y с помощью var

3. Чем отличаются let, const и var?

· var является устаревшим методом объявления переменных, он был использован до появления let и const;

· let используется для объявления переменных, значения которых могут изменяться;

· const используется для объявления констант, то есть переменных, значения которых не могут быть изменены после инициализации.

Например:

```
let count = 10;
count = 20; // Допустимо для let
const PI = 3.14159;
PI = 3; // Недопустимо для const, приведет к ошибке
```

Кроме того, существуют отличия в области видимости между var, let и const в JavaScript:

· var: Переменные, объявленные с помощью var, имеют функциональную область видимости (function scope). Это означает, что они видны только в функции, в которой были объявлены.

· let и const: Они имеют блочную область видимости (block scope), что означает, что они видны только внутри блока, в котором были объявлены (обычно это блоки кода в фигурных скобках, такие как условия if, циклы for, функции и другие).

Это означает, что переменная, объявленная с помощью var, может быть видна внутри функции, но и за ее пределами, если она не является блочной. В то время как переменные, объявленные с помощью let и const, будут видны только внутри блока кода, в котором были определены.

4. Что такое тип данных undefined в JavaScript?

undefined – это специальное значение, которое получает переменная, если ей не было присвоено никакое значение.

Например:

```
let x; // переменная x имеет значение undefined
console.log(x); // Выведет: undefined
```

5. Какие методы у массивов в JavaScript?

Основные методы массивов в JavaScript включают push, pop, shift, unshift, map и reduce, filter. Они предоставляют различные способы изменения и обработки элементов в массиве.

Например:

```
let arr = [1, 2, 3];
arr.push(4); // добавляет элемент в конец массива
arr.pop(); // удаляет последний элемент массива
arr.map(item => item * 2); // создает новый массив, умножая каждый элемент на 2
```

6. Чем отличаются замыкания (closures) от обычных функций в JavaScript?

Замыкание (closure) – это функция, которая имеет доступ к переменным из внешней области видимости, даже после завершения выполнения внешней функции. Это позволяет сохранять доступ к переменным и использовать их внутри вложенной функции.

```
function outerFunction() {
let outerVariable = 'I am from the outer function';
function innerFunction() {
console.log(outerVariable); // innerFunction имеет доступ к outerVariable из внешней функции
}
return innerFunction;
}
const inner = outerFunction();
inner(); // Выведет: "I am from the outer function"
```

7. Что такое прототип в JavaScript? Какие механизмы наследования используются в JavaScript?

В JavaScript объекты могут иметь прототипы, из которых они наследуют свойства. Прототип – это механизм, позволяющий объектам наследовать свойства других объектов. JavaScript использует прототипное наследование, где каждый объект имеет ссылку на прототип (другой объект), и он может наследовать свойства этого прототипа.

```
// Создаем конструктор объекта
function Animal(name) {
this.name = name;
}
// Добавляем метод в прототип объекта Animal
Animal.prototype.sayHello = function () {
console.log("Hello, my name is " + this.name);
};
// Создаем экземпляр объекта Animal
let cat = new Animal("Whiskers");
// Вызываем метод sayHello у объекта cat
cat.sayHello(); // Выводит "Hello, my name is Whiskers"
```

8. Что такое стрелочные функции (arrow functions) и в чём их особенность? Какие у них могут быть ограничения?

Стрелочные функции (arrow functions) отличаются от обычных функций тем, что не имеют своего собственного контекста this, используя контекст окружающего кода. Они также короче и автоматически возвращают значение, если оно записано в одну строку без использования фигурных скобок. Однако у стрелочных функций есть ограничения, например, они не могут быть использованы как конструкторы или иметь свои собственные методы this.

```
const regularFunction = function(a, b) {
return a + b;
};
const arrowFunction = (a, b) => a + b; // Короткий синтаксис для однострочных выражений
// Стрелочная функция с использованием фигурных скобок
const arrowFunctionWithBlock = (a, b) => {
const result = a + b;
return result;
};
```

9. Что такое асинхронность в JavaScript? Какие инструменты предоставляет JavaScript для работы с асинхронным кодом?

Асинхронность в JavaScript – это способ выполнения операций, который не блокирует основной поток выполнения. В JavaScript для работы с асинхронным кодом используются колбэки (callbacks), промисы (promises), async/await и другие методы, позволяющие эффективно управлять асинхронными операциями.

```
// Пример использования Promise и fetch для асинхронного получения данных
fetch('https://api.example.com/data')
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Error fetching data:', error));
// Пример использования async/await
async function fetchData() {
try {
const response = await fetch('https://api.example.com/data');
const data = await response.json();
console.log(data);
} catch (error) {
```

```
console.error('Error fetching data:', error);
}
}
```

10. Как работает система обработки ошибок (error handling) в JavaScript?

Система обработки ошибок (error handling) в JavaScript позволяет обнаруживать и обрабатывать ошибки в коде. Это включает в себя использование конструкции try…catch, где код, который может вызвать ошибку, помещается в блок try, а обработка ошибки происходит в блоке catch.

```
function divide(a, b) {
try {
if (b === 0) {
throw new Error('Division by zero');
}
return a / b;
} catch (error) {
console.error('Error:', error.message);
}
}
console.log(divide(10, 2)); // Выведет: 5
console.log(divide(10, 0)); // Выведет: "Error: Division by zero"
```

11. Что такое hoisting в JavaScript? Как это работает с переменными и функциями?

Hoisting в JavaScript – это механизм, при котором переменные и функции "поднимаются" вверх своей области видимости перед тем, как код начнет выполняться. Это означает, что переменные могут быть объявлены после их использования, а функции могут быть вызваны до их объявления. Однако только объявления переменных и функций поднимаются, а присвоения значений остаются на своих местах.

```
console.log(myVar); // Выведет: undefined
let myVar = 10;
```

12. Чем отличается синхронный код от асинхронного? Какие могут быть преимущества использования асинхронного кода?

Синхронный код выполняется последовательно, один шаг за другим, в то время как асинхронный код позволяет выполнить некоторые операции параллельно или отложить их выполнение, продолжая выполнение кода. Преимущества асинхронного кода включают повышение производительности за счет параллельного выполнения операций и предотвращение блокировки потока выполнения, что позволяет приложению быть более отзывчивым.

13. Что такое Event Loop в JavaScript? Как он влияет на выполнение асинхронного кода?

Event Loop (Цикл событий) – это механизм в JavaScript, который контролирует порядок выполнения кода. Он следит за стеком вызовов и очередью колбэков, перемещая колбэки из очереди в стек при завершении синхронного кода. Это позволяет асинхронному коду выполняться после завершения синхронных операций, не блокируя основной поток выполнения.

14. Какие основные принципы функционального программирования можно использовать в JavaScript?

Принципы функционального программирования в JavaScript включают использование функций высшего порядка, чистых функций, неизменяемости данных и функций map, filter и reduce для обработки данных. Эти концепции помогают создавать более чистый, модульный и легко поддерживаемый код.

Рассмотрим каждый из этих принципов более детально:

1) Функции первого класса (First-Class Functions). В JavaScript функции являются объектами первого класса, что означает, что их можно присваивать переменным, передавать как аргументы, возвращать из других функций.

```
const greet = function (name) {
return `Hello, ${name}!`;
};
const sayHello = greet;
console.log(sayHello("John")); // Output: Hello, John!
```

2) Чистые функции (Pure Functions). Чистая функция возвращает результат, основываясь только на своих аргументах, не имеет побочных эффектов и не зависит от глобального состояния.

```
function add(a, b) {
return a + b;
}
console.log(add(2, 3)); // Output: 5
```

3) Неизменяемость (Immutability). Изменение состояния может привести к ошибкам и сложностям в отладке. В функциональном программировании ценится неизменяемость данных, и вместо изменения существующих данных создаются новые.

```
const numbers = [1, 2, 3];
const newNumbers = [...numbers, 4]; // создание нового массива с добавлением элемента
console.log(newNumbers); // Output: [1, 2, 3, 4]
```

4) Функции высшего порядка (Higher-Order Functions). Это функции, которые принимают другие функции в качестве аргументов или возвращают их. Они позволяют абстрагировать операции и создавать более гибкий и читаемый код.

```
const multiplyBy = function (factor) {
return function (number) {
return number * factor;
};
};
const double = multiplyBy(2);
console.log(double(5)); // Output: 10
```

5) Рекурсия. Вместо циклов используется рекурсия для выполнения повторяющихся задач. Рекурсивные функ-

ции вызывают сами себя с изменяющимися аргументами.

```
function factorial(n) {
return n === 0 ? 1 : n * factorial(n – 1);
}
console.log(factorial(5)); // Output: 120
```

6) Функциональные комбинаторы. Это функции, которые комбинируют другие функции, чтобы создавать новые. Примеры включают map, filter, и reduce.

```
const square = x => x * x;
const increment = x => x + 1;
const squareAndIncrement = compose(increment, square);
console.log(squareAndIncrement(3)); // Output: 10
```

7) Каррирование (Currying). Процесс преобразования функции с множеством аргументов в цепочку функций, каждая из которых принимает только один аргумент.

```
const square = x => x * x;
const increment = x => x + 1;
const squareAndIncrement = compose(increment, square);
console.log(squareAndIncrement(3)); // Output: 10
```

15. Что такое RESTful API? Какие HTTP методы обычно используются для взаимодействия с RESTful API?

RESTful API (Representational State Transfer API) представляет собой стандарт архитектуры веб-сервисов, основанный на принципах REST. Он использует стандартные протоколы и методы HTTP для обмена данными между клиентом и сервером. Основные принципы REST включают отсутствие состояния (stateless), клиент-серверную архитектуру и использование унифицированных интерфейсов.

HTTP методы (или HTTP глаголы) обеспечивают различные операции в RESTful API:

1) GET. Используется для запроса данных или информации от сервера. Не должен изменять состояние сервера.
2) POST. Используется для создания новых ресурсов на сервере. Часто используется при отправке данных формы.
3) PUT. Используется для обновления существующего ресурса на сервере. Полностью заменяет текущее состояние ресурса.
4) PATCH. Аналогичен PUT, но применяется для частичного обновления ресурса, а не его полной замены.
5) DELETE. Используется для удаления ресурса на сервере.

6) OPTIONS. Используется для запроса информации о возможных методах HTTP, поддерживаемых сервером для указанного ресурса.

7) HEAD. Аналогичен GET, но возвращает только заголовки без тела ответа. Часто используется для проверки доступности ресурса или получения метаданных.

8) TRACE. Этот метод запрашивает сервер отправить обратно полученный запрос, что позволяет клиенту видеть, как запрос прошел через промежуточные серверы.

9) CONNECT. Используется для установки туннеля к серверу, идентифицированному по ресурсу.

Эти методы предоставляют различные способы взаимодействия с ресурсами на сервере в рамках RESTful API.

16. Что такое Callback функции в JavaScript? Как они используются при работе с асинхронным кодом?

Callback функции – это ключевой механизм в асинхронном JavaScript, который позволяет управлять асинхронными операциями и выполнением кода после завершения этих операций.

Когда мы работаем с асинхронным кодом, например, при загрузке данных с сервера или выполнении запросов, мы не можем ожидать завершения этих операций, так как это может занять время. Вместо этого, мы используем callback функции, чтобы указать, что нужно сделать после завершения определенной асинхронной операции.

Пример использования callback функции при выполнении асинхронной операции, например, загрузке данных с сервера:

```
function fetchData(callback) {
// Процесс загрузки данных с сервера
setTimeout(function() {
const data = 'Данные с сервера';
callback(data); // Вызываем callback функцию и передаем ей полученные данные
}, 2000); // Например, имитация задержки загрузки дан-
```

ных на 2 секунды
}
function displayData(*data*) {
console.log('Получены данные:', *data*);
}
fetchData(displayData); // Вызываем функцию fetchData и передаем ей displayData в качестве callback функции

В этом примере fetchData – это функция, которая имитирует загрузку данных с сервера. Она принимает callback функцию в качестве аргумента и вызывает ее, когда данные будут доступны. displayData – это callback функция, которая принимает данные и выводит их в консоль после их получения.

Такой подход позволяет продолжать выполнение кода после завершения асинхронной операции, делая асинхронный код более гибким и эффективным.

17. Чем отличаются методы массивов map (), filter () и reduce ()? Приведите примеры использования каждого из них?

Методы массивов map(), filter() и reduce() – каждый из этих методов предоставляет различные способы обработки массивов в JavaScript:

· map() – создает новый массив, содержащий результат вызова предоставленной функции для каждого элемента исходного массива. Этот метод не изменяет исходный массив, а возвращает новый массив с результатами применения функции к каждому элементу.

```
const numbers = [1, 2, 3, 4];
const doubled = numbers.map(num => num * 2);
/ doubled: [2, 4, 6, 8]
```

· filter() – создает новый массив, содержащий только те элементы исходного массива, для которых функция возвращает true. Этот метод также не изменяет исходный массив, а возвращает новый массив с отфильтрованными элементами.

```
const numbers = [1, 2, 3, 4];
const evenNumbers = numbers.filter(num => num % 2 === 0);
// evenNumbers: [2, 4]
```

· reduce() – применяет функцию-аккумулятор к каждому элементу массива, сводя его к единственному значению. Этот метод может выполняться на массиве для вычисления суммы, подсчета количества элементов, объединения элементов и многого другого.

const numbers = [1, 2, 3, 4];

const sum = numbers.reduce((*accumulator*, *currentValue*) => *accumulator* + *currentValue*, 0);

// sum: 10

в данном примере 0 – это начальное значение аккумулятора.

18. Что такое Promise в JavaScript? Какие преимущества они предоставляют при работе с асинхронным кодом?

Promise – это объект, который представляет результат успешного выполнения или ошибку асинхронной операции. Они позволяют работать с асинхронным кодом, управлять последовательностью операций и обрабатывать результаты этих операций.

19. Что такое async и await в JavaScript? Как они упрощают написание асинхронного кода?

Ключевые слова async и await помогают упростить асинхронное программирование. Когда функция объявляется с ключевым словом async, она всегда возвращает Promise. await используется внутри async функций для приостановки выполнения кода до тех пор, пока Promise не завершится, и затем возвратит результат этого Promise. Это позволяет писать асинхронный код в более линейном стиле, без использования цепочек колбэков или методов обработки Promise.

Например:

```
function fetchData() {
return new Promise(resolve => {
setTimeout(() => {
resolve('Данные с сервера');
}, 2000);
});
}
async function getData() {
try {
const result = await fetchData();
console.log(result); // Выведет: 'Данные с сервера'
```

```
} catch (error) {
console.error('Ошибка:', error);
}
}
getData();
```

Эти два ключевых слова помогают улучшить читаемость и структуру асинхронного кода, делая его более понятным и легким для работы.

20. Какие методы объекта Math существуют в JavaScript? Приведите примеры их использования?

Math в JavaScript предоставляет различные методы для выполнения математических операций:

Math.random() – возвращает псевдослучайное число от 0 (включительно) до 1 (не включительно);

```
const randomNum = Math.random();
console.log(randomNum); // Выведет случайное число между 0 и 1
```

Math.floor() – округляет число вниз до ближайшего целого числа;

```
const num = 4.9;
const roundedDown = Math.floor(num);
console.log(roundedDown); // Выведет: 4
```

Math.ceil() – округляет число вверх до ближайшего целого числа:

```
const num = 4.1;
const roundedUp = Math.ceil(num);
console.log(roundedUp); // Выведет: 5
```

Math.abs() – возвращает абсолютное значение числа:

```
const num = -10;
const absoluteValue = Math.abs(num);
```

```
console.log(absoluteValue); // Выведет: 10
```

Math.max() и Math.min() – возвращают максимальное и минимальное значение из набора чисел соответственно:

```
const maxNum = Math.max(10, 5, 8);
```

Конец ознакомительного фрагмента.