Gee Test

Тест по основам программирования на JavaScript

Тест по основам программирования на JavaScript

Тест «Основы программирования на JavaScript» — это практичный набор вопросов, который охватывает ключевые разделы языка: базовые конструкции, массивы, функции, объекты, работу с DOM и событиями, а также асинхронность и Promises.

Основы программирования на JavaScript

Подойдёт для начинающих разработчиков и тех, кто хочет закрепить фундаментальные навыки JavaScript перед собеседованием или экзаменом.

JavaScript остаётся одним из самых востребованных языков программирования благодаря своей универсальности и использованию в веб-разработке. Этот тест помогает структурировать знания и проверить понимание ключевых тем, которые лежат в основе написания любого современного JavaScript-кода. Каждая тема включает 24 практических вопроса, построенных так, чтобы охватить типичные ошибки, базовые паттерны и основные приёмы работы.

От простых алгоритмических конструкций до асинхронного программирования — тест постепенно усложняется и погружает в реальные сценарии разработки. Вопросы помогут лучше разбираться в областях видимости, динамике объектов, работе с DOM-деревом, обработчиками событий и управлении промисами. Такой формат позволяет не просто повторить материал, но и научиться применять его в практических задачах.

Тема 1. Базовые алгоритмические конструкции в JavaScript.

1. Что выведет код: `console.log(2 + 3 * 4)`

1) `20`

2)+ `14`

3) `2`

ⓘ Правильный ответ - 14, потому что операция умножения выполняется раньше сложения по приоритету операторов

2. Какие конструкции относятся к циклам в JavaScript?

1)+ `for`

2)+ `while`

3) `switch`

4)+ `do…while`

ⓘ Правильные варианты: for, while, do…while. switch - не цикл, а условная конструкция

3. Что вернёт выражение: `typeof null`

1) `"null"`

2)+ `"object"`

3) `"undefined"`

ⓘ Правильный ответ - "object", это известная особенность JavaScript, null считается объектом по историческим причинам

4. Какие инструкции являются условными?

1)+ `if`

2)+ `switch`

3) ~~`for`~~

4) `break`

ⓘ Правильные: if и switch позволяют выполнять код по условию, for - цикл, break - прерывание цикла или switch

5. Что выведет `console.log("5" - 2)`

1) `"52"`

2)+ `3`

3) `"3"`

ⓘ Строка "5" при вычитании преобразуется в число, поэтому 5-2=3

6. Что относится к блочным областям видимости?

1)+ `let`

2)+ `const`

3) `var`

4) `function`

ⓘ Правильные: let и const имеют блочную область видимости. var - функциональная, function - зависит от объявления

7. Что делает оператор `===` ?

1) Сравнивает только значения

2)+ Сравнивает значения и типы

3) Преобразует типы автоматически

ⓘ Строгое сравнение учитывает и значение, и тип, поэтому верный ответ - сравнивает значения и типы

8. Выбери циклы JavaScript:

1)+ `for`

2)+ `while`

3)+ `do…while`

4) `repeat`

ⓘ for, while, do…while - корректные циклы, repeat в JS нет

9. Что вернёт `Boolean(0)`

1)+ `false`

2) `true`

3) `undefined`

ⓘ 0 приводится к false при преобразовании к логическому типу

10. Что является операторами сравнения?

1)+ `==`

2)+ `===`

3)+ `>`

4) `+=`

ⓘ ==, === и > сравнивают значения, поэтому верны. += -оператор присваивания с суммой, не сравнения

11. Что выведет `console.log("5" + 2)`

1)+ `"52"`

2) `7`

3) `5`

ⓘ При конкатенации строки и числа число приводится к строке, получается "52"

12. Какие конструкции являются управляющими?

1)+ `if`

2)+ `switch`

3)+ `try…catch`

4) `console`

ⓘ if, switch, try…catch управляют выполнением кода; console - объект, не управляющая конструкция

13. Что выведет `console.log(10 % 3)`

1) `3`

2)+ `1`

3) `0`

ⓘ Остаток от деления 10 на 3 равен 1, поэтому верный ответ - 1

14. Расположи этапы выполнения кода сверху вниз

1 Интерпретатор читает код

2 Выполняются объявления переменных

3 Выполняются инструкции

4 Возвращается результат

ⓘ Правильный порядок: интерпретатор читает код, выполняются объявления переменных, выполняются инструкции, возвращается результат

15. Что выведет `[1,2,3].length`

1)+ `3`

2) `2`

3) `undefined`

ⓘ Свойство length содержит количество элементов массива, здесь 3

16. Упорядочи стадии работы цикла `for`

1 Инициализация

2 Проверка условия

3 Тело цикла

4 Шаг итерации

ⓘ Правильная последовательность: инициализация, проверка условия, тело цикла, шаг итерации

17. Что вернёт `"abc".toUpperCase()`

1)+ `"ABC"`

2) `"abc"`

3) `undefined`

ⓘ Метод toUpperCase возвращает строку в верхнем регистре, здесь "ABC"

18. Порядок работы `if-else`

1 Проверка условия

2 Ветка `if` (если условие `true`)

3 Ветка `else` (если условие `false`)

4 Продолжение выполнения

ⓘ Правильная последовательность: проверка условия → ветка if → ветка else → продолжение выполнения

19. Что выведет `console.log(true + 1)`

1)+ `2`

2) `true1`

3) `NaN`

ⓘ true приводится к 1, поэтому 1+1=2

20. Что делает оператор `!` ?

1)+ Инвертирует логическое значение

2) Сравнивает переменные

3) Завершает цикл

ⓘ Инвертирует логическое значение: !true → false, !false → true

21. Что выведет `console.log([] == false)`

1)+ true

2) false

3) Ошибка

ⓘ При приведении типов [] становится '', который равен false, поэтому true

22. Чему равен `null == undefined`

1)+ `true`

2) `false`

3) `undefined`

ⓘ null и undefined равны при нестрогом сравнении, поэтому true

23. Что выведет `console.log([1] + [2]);`

1) `3`

2)+ `"12"`

3) `[1,2]`

ⓘ Массивы приводятся к строке при конкатенации, "1" + "2" = "12"

24. Установите соответствия между конструкцией и описанием

`if`	Условное выполнение кода
`for`	Цикл с счётчиком
`switch`	Множественный выбор
`try…catch`	Обработка ошибок

Тема 2. Массивы и операции над ними в JavaScript.

1. `.indexOf(2)`

1)+ `1`

2) `2`

3) `true`

ⓘ Метод indexOf возвращает индекс первого совпадения элемента, здесь 2 находится на позиции 1

2. Установи соответствия между методом и его действием

`push`	Добавляет элемент в конец
`pop`	Удаляет элемент с конца
`shift`	Удаляет элемент с начала
`unshift`	Добавляет элемент в начало

3. Что делает метод `Array.isArray()`

1)+ Возвращает `true` , если значение — массив

2) Создаёт массив

3) Преобразует объект в массив

ⓘ Метод проверяет, является ли значение массивом. Возвращает true для массивов

4. Какие методы изменяют исходный массив?

1)+ `push`

2)+ `pop`

3) `slice`

4)+ `splice`

ⓘ push, pop и splice изменяют массив. slice возвращает новый массив и исходный не трогает

5. Что выведет `[10,20,30].length`

1)+ `3`

2) `2`

3) `undefined`

ⓘ Свойство length содержит количество элементов, здесь 3

6. Что относится к методам перебора массива?

1)+ `forEach`

2)+ `map`

3) `push`

4) `length`

ⓘ forEach и map перебирают элементы и создают новые данные. push добавляет элемент, length — просто свойство

7. Что вернёт `[1,2,3].includes(3)`

1)+ `true`

2) `false`

3) `3`

ⓘ includes проверяет наличие элемента, возвращает true если элемент найден

8. Установи соответствия: метод → тип возвращаемого результата

`map`	Новый массив
`filter`	Новый массив с отфильтрованными
`forEach`	undefined
`find`	Найденный элемент

9. Что делает метод `concat()`

1)+ Возвращает новый объединённый массив

2) Удаляет последний элемент

3) Добавляет элемент в конец

ⓘ concat возвращает новый массив, объединяя два или более массива, исходный не изменяет

10. Какие методы НЕ изменяют массив?

1)+ `slice`

2)+ `concat`

3) `push`

4) `pop`

ⓘ slice и concat создают новый массив, push и pop изменяют исходный массив

11. Что вернёт `[1,2,3].slice(1)`

1)+ `[2,3]`

2) `[1,2]`

3) `[3]`

ⓘ slice возвращает новый массив, начиная с индекса 1: [2,3]

12. Какие методы перебора возвращают новый массив?

1)+ `map`

2)+ `filter`

3) `forEach`

4) `reduce`

ⓘ map и filter создают новый массив. forEach возвращает undefined, reduce сворачивает в одно значение

13. Что делает метод `find()`

1)+ Возвращает первый удовлетворяющий элемент

2) Возвращает индекс

3) Возвращает массив

ⓘ Возвращает первый элемент, удовлетворяющий условию callback

14. Какие методы могут менять длину массива?

1)+ `push`

2)+ `pop`

3) `map`

4) `filter`

ⓘ push и pop изменяют длину массива. map и filter создают новый массив и длину исходного не меняют

15. Что вернёт `[1,2,3].at(-1)`

1)+ `3`

2) `1`

3) `undefined`

ⓘ Метод at позволяет обратиться к элементу с конца. at(-1) — последний элемент массива

16. Какие методы сортируют массив?

1)+ `sort`

2)+ `toSorted`

3) `filter`

4) `reduce`

ⓘ sort и toSorted возвращают отсортированный массив. filter и reduce не сортируют

17. `Что делает метод reverse()`

1)+ Разворачивает массив

2) Создаёт копию в обратном порядке

3) Удаляет элементы

ⓘ reverse изменяет массив, разворачивая порядок элементов

18. Расположи этапы работы `map()`

1 Итерация по элементам

2 Вызов callback для каждого

3 Формирование нового массива

4 Возврат результата

ⓘ map: итерация по элементам → вызов callback → формирование нового массива → возврат результата

19. Что делает метод `reduce()`

1)+ Возвращает одно итоговое значение

2) Возвращает массив

3) Удаляет элементы

ⓘ reduce сворачивает массив в одно значение, применяя функцию к каждому элементу

20. Расположи шаги работы `filter()`

1 Итерация по массиву

2 Проверка условия

3 Добавление подходящих элементов

4 Возврат нового массива

ⓘ filter: итерация → проверка условия → добавление подходящих → возврат нового массива

21. Что вернёт `[1,2,3].join("-")`

1)+ `"1-2-3"`

2) `["1","2","3"]`

3) `"123"`

ⓘ join объединяет элементы массива в строку с указанным разделителем

22. Что делает метод `findIndex()`

1)+ Возвращает индекс первого подходящего элемента

2) Возвращает элемент

3) Возвращает новый массив

ⓘ Возвращает индекс первого элемента, удовлетворяющего условию callback

23. Что возвращает метод `every()`

1)+ `true` , если все элементы соответствуют условию

2) Массив с отфильтрованными

3) Первый подходящий элемент

ⓘ every возвращает true, если все элементы удовлетворяют условию

24. Что делает метод `some()`

1)+ Возвращает `true` , если хотя бы один элемент подходит

2) Возвращает новый массив

3) Всегда возвращает `false`

ⓘ some возвращает true, если хотя бы один элемент удовлетворяет условию

Тема 3. Функции и области видимости в JavaScript.

1. Что выведет код:

```javascript
function f(){
    return 5;
}
console.log(f());
```

1)+ `5`

2) `undefined`

3) `f`

ⓘ Вызов функции возвращает значение, здесь return 5, поэтому результат 5

2. Какие способы объявления функций существуют?

1)+ Function Declaration

2)+ Function Expression

3)+ Arrow Function

4) Class Function

ⓘ Function Declaration, Function Expression и Arrow Function — корректные способы. Class Function не является стандартным способом объявления функции

3. Что выведет `console.log(typeof (()=>{}))`

1)+ `"function"`

2) `"object"`

3) `"undefined"`

ⓘ Стрелочные функции в JavaScript имеют тип function

4. Какие области видимости существуют в JS?

1)+ Глобальная

2)+ Локальная (функция)

3)+ Блочная (let/const)

4) Статическая

ⓘ Глобальная, локальная (функция) и блочная (let/const) области видимости корректны. Статическая области в JS нет

5. Что делает оператор `return` внутри функции?

1)+ Завершает функцию и возвращает значение

2) Прерывает цикл

3) Ничего

ⓘ return завершает выполнение функции и возвращает указанное значение

6. Что из перечисленного — корректные способы вызвать функцию?

1)+ `f()`

2)+ `f.call(this)`

3)+ `f.apply(this)`

4) `f{}`

ⓘ f(), f.call(this) и f.apply(this) корректны. f{} — синтаксически неверно

7. Что выведет `console.log(f.name);` если `function f(){}`

1)+ `"f"`

2) `undefined`

3) `"function"`

ⓘ Свойство name функции содержит её имя, здесь "f"

8. Какие переменные относятся к глобальной области видимости?

1)+ `var` объявленные вне функций

2)+ `const` объявленные вне функций

3) `let` внутри функции

4) переменные внутри блока

ⓘ var и const, объявленные вне функций, глобальны. let внутри функции и переменные внутри блока локальны

9. Что вернёт вызов функции `g()` если `let g = () => 10`

1)+ `10`

2) `undefined`

3) `Ошибка`

ⓘ Стрелочная функция возвращает результат выражения, здесь 10

10. Упорядочи этапы объявления и вызова функции

1 Объявление функции

2 Вызов функции

3 Выполнение тела функции

4 Возврат значения

ⓘ Порядок: объявление функции → вызов → выполнение тела → возврат значения

11. Что делает стрелочная функция `() => x * 2`

1)+ Умножает x на 2 и возвращает

2) Создаёт массив

3) Ничего

ⓘ Стрелочная функция возвращает результат выражения, здесь x умножается на 2

12. Упорядочи этапы замыкания

1 Функция объявляется

2 Внутри создаются локальные переменные

3 Возвращается функция из внешней функции

4 Функция использует переменные внешней функции

ⓘ Сначала объявляется функция → создаются локальные переменные → возвращается функция → функция использует переменные внешней функции

13. Что вернёт `console.log(typeof f)` если `let f = function() {}`

1)+ `"function"`

2) `"object"`

3) `"undefined"`

ⓘ Тип переменной — function

14. Что делает функция высшего порядка?

1)+ Принимает функцию как аргумент

2)+ Возвращает функцию

3) Изменяет глобальный объект

4) Ничего

ⓘ Функция высшего порядка принимает функцию как аргумент и/или возвращает функцию

15. Что выведет `console.log((function(){})())`

1)+ `undefined`

2) `function`

3) `Error`

ⓘ Анонимная функция вызывается сразу, возвращает undefined

16. Какие функции имеют блочную область видимости для переменных?

1)+ Стрелочные функции с let/const

2)+ Функции внутри блока с let/const

3) Function Declaration с var

4) Глобальные функции

ⓘ Стрелочные функции с let/const и функции внутри блока с let/const имеют блочную область видимости. Function Declaration с var и глобальные функции — нет

17. Что делает оператор `arguments` внутри функции

1)+ Позволяет обратиться к аргументам функции

2) Возвращает длину массива

3) Создаёт новый массив

ⓘ arguments позволяет обращаться ко всем переданным аргументам функции

18. Установи соответствия: тип функции → характеристика

Function Declaration	Поднимается вверх области видимости
Function Expression	Доступна после объявления
Arrow Function	Не имеет своего this
Anonymous Function	Не имеет имени

19. Что вернёт `console.log((x)=>x+1)(5)`

1)+ `6`

2) `5`

3) `undefined`

ⓘ Стрелочная функция возвращает x+1, здесь 5+1=6

20. Упорядочь действия при рекурсивной функции

1 Вызов функции

2 Выполнение тела функции

3 Вызов самой себя

4 Возврат значения

21. Что делает `bind()`

1)+ Возвращает новую функцию

2) Изменяет оригинальную функцию

3) Ничего

ⓘ bind создаёт новую функцию с привязанным контекстом this

22. Что делает метод call()

1)+ Выполняет функцию с указанным this

2) Создаёт новую функцию

3) Возвращает массив

ⓘ call вызывает функцию с указанным this

23. Что делает `apply()`

1)+ Выполняет функцию с this и массивом аргументов

2) Создаёт копию функции

3) Ничего

ⓘ apply вызывает функцию с this и массивом аргументов

24. Какая область видимости имеет переменная внутри функции без var/let/const

1)+ Глобальная

2) Локальная

3) Блочная

ⓘ Если переменная не объявлена явно, она становится глобальной

Тема 4. Объекты и свойства в JavaScript.

1. Как создать объект с пустым набором свойств?

1)+ `let obj = {}`

2) `let obj = []`

3)+ `let obj = new Object()`

ⓘ Пустой объект создаётся через фигурные скобки {}. let obj = [] создаёт массив, new Object() тоже корректно, но чаще используют {}

2. Как получить значение свойства `name` объекта `user` ?

1)+ `user.name`

2) `user["namee"]`

3) `user->name`

ⓘ Доступ к свойству через точечную нотацию user.name. Скобочная нотация с ошибкой "namee" некорректна. user->name в JS не используется

3. Что выведет `console.log(Object.keys({a:1,b:2}));`

1)+ ["a","b"]

2) ["1","2"]

3) []

ⓘ Object.keys возвращает массив ключей объекта: ["a","b"]

4. Какие действия можно выполнять с объектами?

1)+ Добавлять свойства

2)+ Удалять свойства

3)+ Перебирать свойства

4) Перебирать массивы

ⓘ Можно добавлять, удалять и перебирать свойства. Перебирать массивы к объекту не относится

5. Что делает оператор `delete obj.prop` ?

1)+ Удаляет свойство

2) Удаляет объект

3) Присваивает `undefined`

ⓘ Удаляет указанное свойство из объекта

6. Какие способы создать объект корректны?

1)+ `let obj = {}`

2)+ `let obj = new Object()`

3) `let obj = []`

4) `let obj = function(){}`

ⓘ {} и new Object() создают объект, let obj = [] создаёт массив, let obj = function(){} — функция

7. Что вернёт `console.log("name" in {name:"John"})`

1)+ `true`

2) `false`

3) `undefined`

ⓘ Оператор in проверяет наличие свойства в объекте, возвращает true

8. Какие методы позволяют перебирать объекты?

1)+ `Object.keys()`

2)+ `Object.values()`

3)+ `Object.entries()`

4) `forEach`

ⓘ Object.keys(), Object.values(), Object.entries() — корректные методы. forEach применяется к массивам

9. Что делает `Object.assign(target, source)`

1)+ Копирует свойства из `source` в `target`

2) Создаёт новый объект

3) Удаляет target

ⓘ Копирует свойства из source в target. Новый объект не создаёт, target не удаляется

10. Какие свойства объектов можно перебирать?

1)+ Свойства, у которых `enumerable=true`

2)+ Свойства объекта напрямую

3) Свойства с `enumerable=false`

4) Методы прототипа

ⓘ Перечисляемые свойства объекта (enumerable=true) и свойства объекта напрямую доступны для перебора. Свойства с enumerable=false и методы прототипа — нет

11. Что выведет `console.log(Object.values({x:1,y:2}))`

1)+ `[1,2]`

2) `["x","y"]`

3) `undefined`

ⓘ Object.values возвращает массив значений объекта, здесь [1,2].

12. Какие методы можно использовать для клонирования объекта?

1)+ `Object.assign({}, obj)`

2)+ `structuredClone(obj)`

3) `slice()`

4) `map()`

ⓘ Object.assign({}, obj) и structuredClone(obj) создают копию объекта. slice и map — методы массивов

13. Что делает `Object.freeze(obj)?`

1)+ Запрещает изменение свойств

2) Удаляет объект

3) Позволяет менять свойства

ⓘ Замораживает объект: запрещает изменение существующих свойств и добавление новых

14. Упорядочите шаги доступа к свойству объекта

1 Указать имя объекта

2 Указать имя свойства

3 Получить значение свойства

4 Использовать значение

15. Что делает `Object.seal(obj)` ?

1)+ Можно изменять существующие свойства

2) Можно добавлять новые свойства

3) Удаляет объект

ⓘ Запрещает добавление новых свойств, но позволяет изменять существующие

16. Упорядочь этапы создания объекта через конструктор

1 Определяем функцию-конструктор

2 Создаём новый объект через new

3 Присваиваем свойства через this

4 Используем объект

17. Что вернёт `console.log(Object.entries({a:1,b:2}))`

1)+ `[["a",1],["b",2]]`

2) `[1,2]`

3) `[a,b]`

ⓘ Object.entries возвращает массив пар ключ-значение.

18. Какие действия корректны с объектом?

1)+ Добавлять свойства

2)+ Изменять свойства

3) Присваивать объект в массиве

4) Удалять массив

ⓘ Можно добавлять и изменять свойства объекта, присваивание объекта в массиве или удаление массива не относится

19. Что делает метод `hasOwnProperty()`

1)+ Возвращает `true` , если свойство принадлежит объекту напрямую

2) Возвращает все свойства

3) Удаляет свойство

ⓘ Возвращает true, если свойство принадлежит объекту напрямую, не через прототип

20. Установите соответствия: объект → действие

`Object.freeze`	Запрещает добавление и изменение
`Object.seal`	Запрещает добавление новых свойств
`Object.assign`	Копирует свойства
`Object.keys`	Возвращает массив ключей

21. Что вернёт `console.log({a:1,b:2}.hasOwnProperty("a"))`

1)+ `true`

2) `false`

3) `undefined`

ⓘ Свойство "a" существует в объекте напрямую → true

22. Что делает `Object.getOwnPropertyNames(obj)`

1)+ Все ключи, включая не перечисляемые

2) Только перечисляемые

3) Значения свойств

ⓘ Возвращает все ключи объекта, включая не перечисляемые

23. Что делает `Object.getPrototypeOf(obj)`

1)+ Возвращает прототип объекта

2) Возвращает объект

3) Возвращает свойства объекта

24. Какие методы возвращают массивы?

1)+ `Object.keys()`

2)+ `Object.values()`

3)+ `Object.entries()`

4) `Object.freeze()`

ⓘ Object.keys(), Object.values(), Object.entries() возвращают массивы. Object.freeze() — нет

Тема 5. Работа с DOM и событиями в JavaScript.

1. Что делает `element.cloneNode(true)` ?

1)+ Глубокое копирование элемента и детей

2) Удаляет элемент

3) Изменяет исходный элемент

ⓘ Создаёт копию элемента вместе с его потомками (глубокое копирование)

2. Какие методы позволяют перемещать элемент в DOM?

`appendChild`	`insertBefore`
`cloneNode`	`getElementById`
ⓘ appendChild и insertBefore перемещают элемент в DOM. cloneNode создаёт копию, getElementById возвращает элемент по id	

3. Что делает `document.createElement("div")` ?

1)+ Новый div элемент

2) Изменяет существующий

3) Удаляет элемент

ⓘ Создаёт новый div-элемент, который ещё не вставлен в DOM

4. Соотнеси события и тип действия

`click`	Щелчок мыши
`keydown`	Нажатие клавиши
`mouseover`	Наведение мыши
`submit`	Отправка формы

5. Что делает `element.style.backgroundColor = "red"` ?

1)+ Устанавливает цвет фона

2) Удаляет элемент

3) Добавляет класс

ⓘ Изменяет цвет фона элемента на красный

6. Что делает `element.getAttribute("id")` ?

1)+ Возвращает значение атрибута id

2) Создаёт id

3) Удаляет id

ⓘ Возвращает текущее значение атрибута id

7. Что делает `document.body` ?

1)+ Возвращает элемент body

2) Создаёт новый body

3) Удаляет body

ⓘ Возвращает элемент текущего документа

8. Какие методы позволяют вставлять HTML-контент?

1)+ `innerHTML`

2)+ `insertAdjacentHTML`

3) `getElementById`

4) `removeAttribute`

9. Что делает `element.addEventListener("click", fn)` ?

1)+ Добавляет обработчик события клика

2) Вызывает событие

3) Удаляет обработчик

ⓘ Привязывает обработчик события click к элементу

10. Что делает `element.removeEventListener("click", fn)` ?

1)+ Удаляет обработчик

2) Вызывает событие

3) Создаёт событие

ⓘ Удаляет ранее добавленный обработчик события click

11. Что делает `event.preventDefault()` ?

1)+ Отменяет стандартное поведение элемента

2) Останавливает выполнение функции

3) Удаляет событие

ⓘ Отменяет действие по умолчанию браузера для события

12. Что делает `event.stopPropagation()` ?

1)+ Останавливает всплытие

2) Удаляет обработчик

3) Выполняет функцию

ⓘ Прекращает всплытие события вверх по дереву DOM

13. Какие события можно обрабатывать на кнопке?

1)+ `click`

2)+ `dblclick`

3)+ `mouseover`

4) `keydown`

ⓘ click, dblclick и mouseover — корректные. keydown обрабатывается на document/input

14. Какие методы изменяют атрибуты элементов?

1)+ `setAttribute`

2)+ `getAttribute`

3)+ `removeAttribute`

4) `querySelector`

ⓘ setAttribute, getAttribute, removeAttribute — работают с атрибутами. querySelector — выборка элементов

15. Что делает `element.classList.add("active")` ?

1)+ Добавляет класс

2) Удаляет элемент

3) Возвращает класс

ⓘ Добавляет CSS-класс "active" элементу

16. Какие методы работы с классами элементов существуют?

1)+ `add`

2)+ `remove`

3)+ `toggle`

4) `appendChild`

ⓘ add, remove и toggle — стандартные методы classList. appendChild не относится

17. Какие методы позволяют вставлять элемент в DOM?

1)+ `appendChild`

2)+ `insertBefore`

3) `cloneNode`

4) `getElementById`

ⓘ appendChild и insertBefore — корректные. cloneNode — копия, getElementById — поиск

18. Какие методы работы с событиями существуют?

1)+ `addEventListener`

2)+ `removeEventListener`

3) `click`

4) `keydown`

ⓘ addEventListener и removeEventListener управляют событиями. click, keydown — события, не методы

19. Как удалить элемент из DOM?

1)+ `element.remove()`

2) `element.delete()`

3) `removeChild()`

ⓘ Метод remove() удаляет элемент из DOM

20. Как добавить HTML-контент внутрь элемента?

1)+ `innerHTML`

2)+ `insertAdjacentHTML`

3) `appendChild`

4) `getElementById`

ⓘ innerHTML и insertAdjacentHTML вставляют HTML-код

21. Что возвращает `document.querySelector(".class")` ?

1)+ Первый элемент с указанным классом

2) Все элементы с классом

3) undefined

ⓘ Первый найденный элемент с указанным классом

22. Что делает `cloneNode(false)` ?

1)+ Поверхностное копирование элемента

2) Глубокое копирование

3) Удаляет исходный элемент

ⓘ Копирует элемент без дочерних элементов

23. Что делает `element.textContent` ?

1)+ Текстовое содержимое элемента

2) HTML-контент

3) CSS-класс

ⓘ Возвращает или задаёт текстовое содержимое элемента

24. Что делает `insertAdjacentHTML(position, html)` ?

1)+ Вставляет HTML в указанное место

2) Заменяет элемент полностью

3) Создаёт новый документ

ⓘ Вставляет HTML в указанное место относительно элемента (beforebegin, afterbegin, beforeend, afterend)

Тема 6. Асинхронность и Promises в JavaScript.

1. Что вернёт `await Promise.resolve(10)` внутри async функции?

1)+ `10`

2) `Promise {: 10}`

3) `undefined`

ⓘ await ожидает выполнение Promise и возвращает его результат, поэтому результат — 10

2. Что делает метод `Promise.allSettled([p1,p2])` ?

1)+ Возвращает массив статусов и значений

2) Прерывает при ошибке

3) Возвращает первый результат

ⓘ Возвращает массив статусов и значений всех Promise, независимо от того, выполнены они или отклонены

3. Что делает `setTimeout(fn, 0)` в асинхронном коде?

1)+ Отложенный вызов функции

2) Выполняется сразу

3) Создаёт Promise

ⓘ Запланированный вызов функции выполняется после текущего стека, не блокируя код

4. Какие методы позволяют работать с асинхронным кодом параллельно?

1)+ `Promise.all`

2)+ `Promise.race`

3)+ `fetch c then`

4) `console.log`

5. Что делает `Promise.race([p1,p2])` ?

1)+ Возвращает первый завершившийся `Promise`

2) Ожидает завершения всех ~~`Promise`~~

3) Возвращает массив результатов

ⓘ Возвращает результат первого завершившегося Promise, остальные игнорируются

6. Какие методы `Promise` можно комбинировать?

1)+ `then`

2)+ `catch`

3)+ `finally`

4) `setInterval`

ⓘ then, catch и finally можно комбинировать для последовательной обработки Promise. setInterval не относится

7. Что делает `fetch(url).then(res=>res.json())` ?

1)+ Преобразует ответ в объект

2) Возвращает текст

3) Создаёт новый Promise без данных

ⓘ Метод then обрабатывает ответ fetch и преобразует его в объект через res.json()

8. Какие типы состояния у `Promise` существуют?

1)+ `pending`

2)+ `fulfilled`

3)+ `rejected`

4) `stopped`

ⓘ Pending — ожидание, fulfilled — выполнен, rejected — отклонён. stopped не существует

9. Что делает `async/await` в синтаксисе?

1)+ Ожидание `Promise` внутри функции

2) Создаёт синхронный код

3) Прерывает выполнение всего скрипта

ⓘ Позволяет писать асинхронный код синхронно: await ждёт Promise внутри async функции

10. Соотнеси метод/конструктор → результат

Promise.resolve(value)	Fulfilled с value
Promise.reject(error)	Rejected с error
fetch(url)	Возвращает Promise с ответом
async function	Возвращает Promise

11. Что делает метод `finally()` у `Promise` ?

1)+ Запускается после `then` или `catch`

2) Останавливает выполнение

3) Возвращает результат

ⓘ finally выполняется всегда после then или catch, не изменяя результат

12. Какие конструкции позволяют отловить ошибки Promise?

1)+ `catch`

2)+ `try/catch внутри async функции`

3) then

4) finally

ⓘ catch и try/catch внутри async функции позволяют обрабатывать ошибки. then и finally не ловят ошибки сами

13. Что делает метод `then()` у `Promise` ?

1)+ Добавляет обработчик успешного результата

2) Возвращает новый Promise всегда

3) Прерывает выполнение

ⓘ Добавляет обработчик успешного выполнения Promise

14. Что делает `catch()` у `Promise` ?

1)+ Обрабатывает отклонённый Promise

2) Обрабатывает успешный результат

3) Создаёт новый Promise

ⓘ Обрабатывает отклонённый (rejected) Promise, ошибки

15. Что делает `async function f() {}` ?

1)+ Функция возвращает `Promise`

2) Функция синхронная

3) Функция завершает выполнение сразу

ⓘ Async function всегда возвращает Promise, внутри можно использовать await

16. Что делает `await` внутри `async` функции?

1)+ Ожидает завершения `Promise`

2) Приостанавливает весь скрипт

3) Создаёт новый Promise

ⓘ await приостанавливает выполнение async функции до завершения Promise, но не блокирует весь скрипт

17. Упорядочи шаги выполнения async функции

1 Вызов функции

2 Выполнение тела функции до await

3 Ожидание завершения await

4 Возврат результата

18. Что делает `Promise.all([p1,p2])` ?

1)+ Возвращает один `Promise` , выполненный когда все завершены

2) Возвращает массив значений сразу

3) Прерывает первый Promise

ⓘ Возвращает один Promise, который выполнится, когда все входящие Promise завершены

19. Упорядочи работу `Promise.all`

1 Создание массива Promise

2 Запуск всех Promise параллельно

3 Ожидание завершения всех

4 Возврат массива результатов

20. Что делает `Promise.race([p1,p2])` ?

1)+ Возвращает первый завершившийся Promise

2) Ожидает все Promise

3) Возвращает массив результатов

ⓘ Возвращает результат первого выполненного Promise, остальные игнорируются

21. Что делает `fetch(url)` ?

1)+ Возвращает `Promise` с ответом

2) Синхронный запрос

3) Создаёт объект `XMLHttpRequest`

ⓘ Запрос к серверу возвращает Promise с ответом, который можно обработать через then/catch

22. Что делает метод `then()` у `Promise` ?

1)+ Добавляет обработчик успешного результата

2) Возвращает новый Promise всегда

3) Прерывает выполнение

ⓘ Обработчик для успешного результата Promise

23. Что делает метод `catch()` у `Promise` ?

1)+ Обрабатывает отклонённый `Promise`

2) Обрабатывает успешный результат

3) Создаёт новый Promise

ⓘ Обрабатывает отклонённый Promise, ловит ошибки

24. Что делает `async/await` в синтаксисе?

1)+ Ожидание `Promise` внутри функции

2) Создаёт синхронный код

3) Прерывает выполнение всего скрипта

ⓘ Позволяет писать асинхронный код как синхронный, ожидая Promise внутри async функции

Table of Contents