**1. Переменные. Имена переменных. Константы.**

**Переменные** – это именованные контейнеры для хранения данных. В JavaScript используются ключевые слова var, let и const.

* var – устаревший способ объявления, имеет функциональную область видимости.
* let – блочная область видимости, значение можно изменять.
* const – блочная область видимости, значение нельзя перезаписать (но можно изменять свойства объектов и элементы массивов).

**Имена переменных:**

* Могут содержать буквы, цифры, \_, $.
* Не могут начинаться с цифры.
* Чувствительны к регистру (myVar ≠ myvar).
* Нельзя использовать зарезервированные слова (if, function, return и др.).

**Константы (const):**

* Должны быть инициализированы при объявлении.
* Нельзя переопределять, но можно изменять содержимое объектов и массивов.

**2. Типы данных. Оператор typeof.**

**Типы данных в JavaScript:**

1. **Примитивные:**
   * number – целые и дробные числа, Infinity, NaN.
   * string – строки.
   * boolean – true/false.
   * null – специальное значение "ничего".
   * undefined – значение не определено.
   * symbol – уникальные идентификаторы.
   * bigint – большие целые числа.
2. **Объекты (object):**
   * Объекты, массивы, функции, даты и др.

**Оператор typeof:**  
Возвращает строку с типом значения. Особенности:

* typeof null → "object" (историческая ошибка).
* typeof function → "function".

**3. Преобразование типов.**

**Явное преобразование:**

* String(value) – в строку.
* Number(value) – в число (NaN при неудаче).
* Boolean(value) – в логическое значение.

**Неявное преобразование:**

* В арифметических операциях ("5" - 2 → 3).
* В логическом контексте (if (value)).
* В строках ("5" + 1 → "51").

**Логическое преобразование:**

* false, 0, "", null, undefined, NaN → false.
* Остальные значения → true.

**4. Операторы. Бинарные и унарные операторы. Приоритет операторов. Совмещение операторов.**

**Операторы:**

* **Унарные** (+x, -x, ++x, typeof x).
* **Бинарные** (a + b, a > b).
* **Тернарный** (условие ? a : b).

**Приоритет операторов:**

1. Группировка ().
2. Унарные (!, ++, --).
3. Умножение/деление (\*, /, %).
4. Сложение/вычитание (+, -).
5. Сравнение (>, <, >=, <=).
6. Равенство (==, ===, !=, !==).
7. Логические (&&, ||).
8. Присваивание (=, +=).

**Совмещение операторов:**

* a += b → a = a + b.

**5. Условные операторы: if, '?'. Конструкция switch.**

**if:**  
Проверяет условие и выполняет код в зависимости от его истинности.

**Тернарный оператор ?:**  
Короткая форма if-else: условие ? выражение1 : выражение2.

**switch:**  
Сравнивает значение с вариантами case и выполняет соответствующий блок кода. Без break выполнение продолжится дальше.

**6. Логические операторы.**

* && (И) – возвращает первое ложное или последнее истинное.
* || (ИЛИ) – возвращает первое истинное или последнее ложное.
* ! (НЕ) – инвертирует значение.

**7. Циклы while, for.**

**while:**  
Выполняет код, пока условие истинно.

**do...while:**  
Выполняет код хотя бы один раз, затем проверяет условие.

**for:**  
Инициализация → условие → тело → шаг → повтор.

**8. Функции (Function Declaration). Параметры по умолчанию.**

**Function Declaration:**  
Объявляется как function имя(параметры) { ... }. Поднимается (hoisting).

**Параметры по умолчанию:**  
Можно задать прямо в объявлении: function f(a = 1, b = 2).

**9. Функциональные выражения (Function Expression) и функции-стрелки.**

**Function Expression:**  
Функция как значение переменной: const f = function() { ... }.

**Стрелочные функции:**  
const f = () => { ... }.

* Нет своего this.
* Нельзя использовать как конструктор.

**10. Числа. Способы записи числа. Системы счисления. Методы типа Number.**

**Способы записи:**

* Десятичные (123), шестнадцатеричные (0xFF), двоичные (0b101), восьмеричные (0o777).

**Методы Number:**

* parseInt(), parseFloat() – преобразование строк в числа.
* isNaN(), isFinite() – проверки.
* toFixed(), toPrecision() – округление.

**11. Строки. Методы типа String.**

**Методы:**

* toUpperCase(), toLowerCase() – регистр.
* indexOf(), includes() – поиск подстроки.
* slice(), substring() – извлечение подстрок.

**Сравнение строк:**  
Лексикографический порядок (по кодам Unicode).

**12. Массивы.**

**Массив** – упорядоченная коллекция элементов любого типа.

**Особенности:**

* Динамический размер.
* Индексация с 0.

**13. Методы массивов: добавление, удаление и замена элементов, объединение массивов, поиск.**

**Изменение массива:**

* push(), pop() – конец.
* shift(), unshift() – начало.
* splice() – вставка, удаление.

**Поиск:**

* indexOf(), includes().

**Объединение:**

* concat().

**14. Методы перебора и преобразование массива.**

**Перебор:**

* forEach().

**Преобразование:**

* map(), filter(), reduce().

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

**Объекты:**

* Создание через {}.
* Свойства: obj.key или obj["key"].

**Вычисляемые свойства:**  
[выражение]: значение.

**Проверка свойства:**

* "key" in obj.
* obj.hasOwnProperty("key").

**Перебор:**

* for...in.

**16. Копирование, клонирование, сравнение, объединение объектов. Объекты-константы.**

**Клонирование:**

* Поверхностное: Object.assign(), {...obj}.
* Глубокое: рекурсия или JSON.parse(JSON.stringify(obj)).

**Объекты-константы:**  
const obj = {} – нельзя перезаписать, но можно изменять свойства.

**17. Коллекции Set, WeakSet.**

**Set:**  
Хранит уникальные значения любого типа.

**WeakSet:**  
Только объекты, слабая ссылка (не препятствует сборке мусора).

**18. Коллекции Map, WeakMap.**

**Map:**  
Ключи любого типа, порядок элементов сохранен.

**WeakMap:**  
Ключи – только объекты, слабая ссылка.

**19. Деструктурирующее присваивание. Вложенная деструктуризация.**

**Деструктуризация:**  
Разбор массива/объекта на переменные:  
const [a, b] = arr;  
const {x, y} = obj;

**Вложенная:**  
const {a: {b}} = obj;

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

**Глобальный объект (window, global):**  
Содержит глобальные переменные и функции.

**new Function:**  
Динамическое создание функции из строки:  
const f = new Function("a", "b", "return a + b");

**21. Лексическое окружение (LexicalEnvironment). Замыкание.**

**Лексическое окружение:**  
Хранит локальные переменные и ссылку на внешнее окружение.

**Замыкание:**  
Функция запоминает свое окружение даже после выполнения.

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

**Свойства функции:**

* name – имя функции.
* length – количество параметров.

**NFE:**  
const f = function name() {} – имя видно только внутри функции.

**23. Остаточные параметры и оператор расширения.**

**Остаточные параметры (...rest):**  
Собирают аргументы в массив.

**Оператор расширения (...arr):**  
Разбирает массив/объект на элементы.

**24. Каррирование и частичное применение функции.**

**Каррирование:**  
Преобразование функции в цепочку вызовов с одним аргументом.

**Частичное применение:**  
Фиксация части аргументов функции.

**25. Генераторы. Функции-генераторы. Перебор объектов-генераторов.**

**Генератор (function\*):**  
Функция, которая может приостанавливаться (yield).

**Перебор:**  
for...of или вручную через .next().

**26. Методы объектов, this. Оператор опциональной последовательности.**

**this:**  
Зависит от контекста вызова.

**?.:**  
Проверяет существование свойства перед доступом.

**27. Преобразование объектов.**

**Приведение к примитивам:**  
Методы valueOf(), toString().

**28. Создание объектов через "new".**

**Конструктор:**  
Функция, вызываемая с new, создает объект.

**29. Флаги и дескрипторы свойств.**

**Дескрипторы:**  
value, writable, enumerable, configurable.

**Изменение:**  
Object.defineProperty().

**30. Геттеры и сеттеры.**

**get/set:**  
Функции, вызываемые при чтении/записи свойства.

**31. Декораторы. Методы call(), apply(), bind().**

**Декораторы:**  
Функции, модифицирующие поведение других функций.

**call/apply:**  
Вызов функции с заданным this.

**bind:**  
Создает новую функцию с привязанным this.

**32. Функции setTimeout и setInterval.**

**setTimeout:**  
Выполняет код через указанное время.

**setInterval:**  
Повторяет выполнение с интервалом.

**33. Прототипное наследование. Собственные и унаследованные свойства. Свойство F.prototype.**

**Прототип:**  
Объект, от которого наследуются свойства.

**F.prototype:**  
Объект, используемый как прототип для новых экземпляров.

**34. Классы. Class Expression. Приватные и защищённые методы и свойства.**

**Классы (class):**  
Синтаксический сахар над прототипами.

**Приватные поля (#field):**  
Доступны только внутри класса.

**35. Наследование классов. Переопределение методов. Статические свойства и методы. Оператор instanceof.**

**extends:**  
Наследование классов.

**super:**  
Вызов родительского метода.

**static:**  
Методы/свойства класса, а не экземпляра.

**instanceof:**  
Проверяет принадлежность к классу.

**36. Модули. Основные возможности модулей.**

**Модули (import/export):**  
Изолируют код, предотвращают конфликты.

**37. Модули: экспорт и импорт.**

**export:**  
Позволяет использовать переменные/функции в других файлах.

**import:**  
Загружает экспортированные значения.

**38. Окружение: DOM, BOM. Дерево DOM.**

**DOM:**  
Представление HTML-документа в виде дерева объектов.

**BOM:**  
Браузерные API (window, navigator).

**39. Навигация и методы поиска DOM-элементов.**

**Поиск:**

* document.getElementById().
* querySelector(), querySelectorAll().

**Навигация:**

* parentNode, childNodes, nextSibling.

**40. Свойства узлов: тип, тег и содержимое.**

**Тип узла (nodeType):**  
1 – элемент, 3 – текст.

**Тег (tagName):**  
Имя HTML-элемента.

**Содержимое (textContent, innerHTML):**  
Текст или HTML внутри узла.

**41. Атрибуты и DOM-свойства.**

**Атрибуты (getAttribute):**  
HTML-атрибуты.

**DOM-свойства:**  
Свойства объекта-узла.

**42. Добавление и удаление DOM-узлов.**

**Создание:**  
document.createElement().

**Добавление:**  
append(), prepend(), insertBefore().

**Удаление:**  
remove(), removeChild().

**43. Стили DOM-узлов.**

**Доступ к стилям:**  
element.style.property.

**Классы:**  
classList.add(), classList.remove().

**44. Размеры и прокрутка элементов и страницы.**

**Размеры:**  
offsetWidth, clientWidth, scrollWidth.

**Прокрутка:**  
scrollTop, scrollTo().

**45. Размеры и прокрутка окна. Координаты.**

**Окно:**  
window.innerWidth, window.scrollY.

**Координаты:**  
getBoundingClientRect().

**46. Браузерные события.**

**Типы событий:**  
Клик, наведение, ввод и др.

**Обработчики:**  
addEventListener().

**47. Всплытие и погружение событий.**

**Всплытие:**  
Событие поднимается от элемента к родителям.

**Погружение (capture):**  
Событие опускается от документа к элементу.

**48. Делегирование событий. Действия браузера по умолчанию.**

**Делегирование:**  
Обработка событий на родителе.

**Отмена действий:**  
event.preventDefault().

**49. Генерация событий.**

**Создание события:**  
new Event().

**Инициация:**  
dispatchEvent().

**50. События мыши. События mouseover/out, mouseenter/leave.**

**mouseover/mouseout:**  
Срабатывают при входе/выходе с элемента (включая потомков).

**mouseenter/mouseleave:**  
Только сам элемент (без всплытия).

**51. События клавиатуры: keyup, keydown. Прокрутка: событие scroll.**

**Клавиатура:**

* keydown – нажатие.
* keyup – отпускание.

**Прокрутка:**  
scroll – при скролле элемента.

**52. События жизненного цикла HTML-страницы.**

**DOMContentLoaded:**  
DOM загружен (без стилей и картинок).

**load:**  
Полная загрузка страницы.

**beforeunload:**  
Перед закрытием страницы.

**53. Загрузка скриптов, ресурсов.**

**<script>:**

* async – загрузка параллельно, выполнение сразу.
* defer – выполнение после загрузки DOM.

**54. Свойства и методы формы.**

**Элементы формы:**  
form.elements.

**Методы:**  
submit(), reset().

**55. Фокусировка элементов формы.**

**Фокус:**  
focus(), blur().

**События:**  
focus, blur.

**56. Изменение значений элемента формы. Формы: отправка, событие и метод submit.**

**Значения:**  
input.value, checkbox.checked.

**Отправка:**  
submit – событие перед отправкой.

**57. JS-библиотека React. Понятие иммутабельности и согласования.**

**React:**  
Библиотека для создания UI.

**Иммутабельность:**  
Данные не изменяются, а создаются новые.

**Согласование (Reconciliation):**  
Алгоритм обновления DOM.

**58. Расширение языка JavaScript – JSX. Рендеринг элементов.**

**JSX:**  
Синтаксис для описания UI в React.

**Рендеринг:**  
ReactDOM.render().

**59. React: компоненты и пропсы.**

**Компоненты:**  
Функции или классы, возвращающие JSX.

**Пропсы (props):**  
Данные, передаваемые в компонент.

**60. React: состояние и жизненный цикл: монтирование, обновление, размонтирование.**

**Состояние (state):**  
Локальные данные компонента.

**Жизненный цикл:**

* componentDidMount – после монтирования.
* componentDidUpdate – после обновления.
* componentWillUnmount – перед удалением.

**61. React: обработка событий.**

**События:**  
onClick, onChange и др.

**Обработчики:**  
Функции, переданные в props.

**62. React: условный рендеринг, рендеринг списка, ключи.**

**Условный рендеринг:**  
{condition && <Component />}.

**Списки:**  
map() с key для идентификации элементов.

**63. React: формы.**

**Управляемые компоненты:**  
Значения формы хранятся в state.

**Неуправляемые:**  
Доступ через ref.

**64. React: хуки, правила хуков.**

**Хуки (useState, useEffect):**  
Функции для работы с состоянием и эффектами.

**Правила:**

* Вызывать только на верхнем уровне.
* Только в функциональных компонентах.

**65. React: использование хука состояния.**

**useState:**  
Возвращает состояние и функцию для его обновления.

**66. React: использование хука эффекта.**

**useEffect:**  
Выполняет побочные эффекты (запросы, подписки).

**67. Redux: назначение, основные концепции.**

**Redux:**  
Управление состоянием приложения.

**Концепции:**

* Единое хранилище (store).
* Неизменяемость состояния.

**68. Redux: экшены.**

**Экшены (actions):**  
Объекты с type и данными.

**69. Redux: редьюсеры.**

**Редьюсеры (reducers):**  
Чистые функции, обновляющие состояние.

**70. Redux: стор.**

**Стор (store):**  
Хранит состояние и методы для работы с ним.

**71. Redux: поток данных.**

**Однонаправленный поток:**  
View → Action → Reducer → Store → View.

**72. Redux: использование с React (react-redux).**

**react-redux:**

* Provider – передает store в компоненты.
* useSelector, useDispatch – доступ к состоянию и экшенам.