Конспект по JS

**ООП**  
  
Главное

* Инкапсулируйте все, что может изменяться;
* Уделяйте больше внимания интерфейсам, а не их реализациям;
* Каждый класс в вашем приложении должен иметь только одно назначение;
* Классы — это их поведение и функциональность.

Базовые принципы ООП

* Абстракция — отделение концепции от ее экземпляра;
* Полиморфизм — реализация задач одной и той же идеи разными способами;
* Наследование — способность объекта или класса базироваться на другом объекте или классе. Это главный механизм для повторного использования кода. Наследственное отношение классов четко определяет их иерархию;
* Инкапсуляция — размещение одного объекта или класса внутри другого для разграничения доступа к ним.

Используйте следующее вместе с наследованием

* Делегация — перепоручение задачи от внешнего объекта внутреннему;
* Композиция — включение объектом-контейнером объекта-содержимого и управление его поведением; последний не может существовать вне первого;
* Агрегация — включение объектом-контейнером ссылки на объект-содержимое; при уничтожении первого последний продолжает существование.

Не повторяйся (Don’t repeat yourself — DRY)

Избегайте повторного написания кода, вынося в абстракции часто используемые задачи и данные. Каждая часть вашего кода или информации должна находиться в единственном числе в единственном доступном месте. Это один из [принципов читаемого кода](https://tproger.ru/articles/how-to-write-readable-code/).

Принцип единственной обязанности

Для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.

**Базовые структуры данных**

Базовые структуры данных являются фундаментальными элементами программирования. Вот краткое описание некоторых базовых структур данных и примеры их реализации на JavaScript:

**Массив (Array):**

Массив - это упорядоченная коллекция элементов, доступ к которым осуществляется по индексу.  
  
const myArray = [1, 2, 3, 4, 5];

console.log(myArray[2]); // Вывод: 3  
  
 **Связанный список (Linked List):**  
Связанный список - это структура данных, состоящая из узлов, каждый из которых содержит данные и ссылку на следующий узел.  
  
class Node {

constructor(data) {

this.data = data;

this.next = null;

}

}

class LinkedList {

constructor() {

this.head = null;

}

append(data) {

const newNode = new Node(data);

if (this.head === null) {

this.head = newNode;

} else {

let current = this.head;

while (current.next !== null) {

current = current.next;

}

current.next = newNode;

}

}

}

const myList = new LinkedList();

myList.append(1);

myList.append(2);

myList.append(3);

**Стек (Stack):**  
Стек - это структура данных, в которой элементы добавляются и удаляются только с одного конца (вершины).  
  
class Stack {

constructor() {

this.stack = [];

}

push(item) {

this.stack.push(item);

}

pop() {

return this.stack.pop();

}

}

const myStack = new Stack();

myStack.push(1);

myStack.push(2);

myStack.push(3);

console.log(myStack.pop()); // Вывод: 3

**Очередь (Queue):**

Очередь - это структура данных, в которой элементы добавляются в конец и удаляются с начала (первыми пришли - первыми ушли).

class Queue {

constructor() {

this.queue = [];

}

enqueue(item) {

this.queue.push(item);

}

dequeue() {

return this.queue.shift();

}

}

const myQueue = new Queue();

myQueue.enqueue(1);

myQueue.enqueue(2);

myQueue.enqueue(3);

console.log(myQueue.dequeue()); // Вывод: 1

**Дерево (Tree):**

Дерево - это иерархическая структура данных, состоящая из узлов, где каждый узел имеет родительский узел и ноль или более дочерних узлов.

Пример на JavaScript:  
javascript  
  
class TreeNode {

constructor(data) {

this.data = data;

this.children = [];

}

addChild(node) {

this.children.push(node);

}

}

const rootNode = new TreeNode('A');

const childNode1 = new TreeNode('B');

const childNode2 = new TreeNode('C');

rootNode.addChild(childNode1);

rootNode.addChild(childNode2);

**Хэш-таблица (Hash Table):**

Хэш-таблица - это структура данных, которая использует хэш-функцию для преобразования ключей в индексы таблицы, где значения хранятся.

Пример на JavaScript (используя встроенный объект Map):  
javascript  
  
const myHashTable = new Map();

myHashTable.set('key1', 'value1');

myHashTable.set('key2', 'value2');

myHashTable.set('key3', 'value3');

console.log(myHashTable.get('key2')); // Вывод: value2

**Базовые алгоритмы сортировки и поиска**

**Алгоритмы сортировки:**

**1. Сортировка пузырьком (Bubble Sort):**

- Этот алгоритм проходит по списку несколько раз, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке.

- Процесс продолжается до тех пор, пока весь список не будет отсортирован.

Пример на JavaScript:

function bubbleSort(arr) {

const len = arr.length;

for (let i = 0; i < len - 1; i++) {

for (let j = 0; j < len - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

[arr[j], arr[j + 1]] = [arr[j + 1], arr[j]]; // Меняем элементы местами

}

}

}

return arr;

}

const unsortedArray = [64, 34, 25, 12, 22, 11, 90];

const sortedArray = bubbleSort(unsortedArray);

console.log(sortedArray); // Вывод: [11, 12, 22, 25, 34, 64, 90]

**2. Сортировка выбором (Selection Sort):**

- В этом алгоритме на каждом шаге находится наименьший элемент в оставшейся части списка и меняется местами с элементом на текущей позиции.

- Процесс продолжается до тех пор, пока список не будет отсортирован.

Пример на JavaScript:

function selectionSort(arr) {

const len = arr.length;

for (let i = 0; i < len - 1; i++) {

let minIndex = i;

for (let j = i + 1; j < len; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

[arr[i], arr[minIndex]] = [arr[minIndex], arr[i]]; // Меняем элементы местами

}

return arr;

}

const unsortedArray = [64, 34, 25, 12, 22, 11, 90];

const sortedArray = selectionSort(unsortedArray);

console.log(sortedArray); // Вывод: [11, 12, 22, 25, 34, 64, 90]

**Алгоритмы поиска:**

**1. Линейный поиск (Linear Search):**

- В этом алгоритме каждый элемент списка последовательно сравнивается с искомым значением.

- Если элемент найден, возвращается его индекс; в противном случае возвращается -1.

Пример на JavaScript:

function linearSearch(arr, target) {

const len = arr.length;

for (let i = 0; i < len; i++) {

if (arr[i] === target) {

return i;

}

}

return -1;

}

const array = [12, 45, 67, 23, 9];

const targetValue = 23;

const index = linearSearch(array, targetValue);

console.log(index); // Вывод: 3

**2. Бинарный поиск (Binary Search):**

- В этом алгоритме предполагается, что список отсортирован.

- Он находит средний элемент списка и сравнивает его с искомым значением.

- Если значения совпадают, возвращается индекс; если искомое значение меньше, поиск продолжается в левой половине списка; если оно больше, поиск продолжается в правой половине списка.

- Процесс повторяется до тех пор, пока значение не будет найдено или список не будет исчерпан.

Пример на JavaScript:

function binarySearch(arr, target) {

let low = 0;

let high = arr.length - 1;

while (low <= high) {

const mid = Math.floor((low + high) / 2);

if (arr[mid] === target) {

return mid;

} else if (arr[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

const sortedArray = [11, 12, 22, 25, 34, 64, 90];

const targetValue = 25;

const index = binarySearch(sortedArray, targetValue);

console.log(index); // Вывод: 3

**Двоичная система счисления**

Двоичная система счисления - это система, основанная на двух цифрах: 0 и 1. В ней числа представляются в виде последовательности двоичных цифр, называемых битами. Каждый бит имеет два возможных значения: 0 или 1.

JavaScript предоставляет встроенные функции и методы для работы с двоичными числами. Вот некоторые примеры использования двоичной системы счисления в JavaScript:

**1. Представление двоичных чисел:**

const binaryNumber = 0b10101; // Префикс "0b" указывает на двоичное число

console.log(binaryNumber); // Вывод: 21

2. Преобразование чисел в двоичную систему счисления:

const decimalNumber = 42;

const binaryString = decimalNumber.toString(2);

console.log(binaryString); // Вывод: "101010"

**3. Преобразование двоичных чисел в десятичную систему счисления:**

const binaryString = "101010";

const decimalNumber = parseInt(binaryString, 2);

console.log(decimalNumber); // Вывод: 42

**4. Операции с двоичными числами:**

const binaryNumber1 = 0b1010;

const binaryNumber2 = 0b1100;

const result = binaryNumber1 & binaryNumber2; // Побитовое И

console.log(result.toString(2)); // Вывод: "1000"

const result2 = binaryNumber1 | binaryNumber2; // Побитовое ИЛИ

console.log(result2.toString(2)); // Вывод: "1110"

const result3 = binaryNumber1 ^ binaryNumber2; // Побитовое исключающее ИЛИ

console.log(result3.toString(2)); // Вывод: "0110"

const result4 = ~binaryNumber1; // Побитовое отрицание

console.log(result4.toString(2)); // Вывод: "-1011"

const result5 = binaryNumber1 << 2; // Побитовый сдвиг влево

console.log(result5.toString(2)); // Вывод: "101000"

const result6 = binaryNumber1 >> 1; // Побитовый сдвиг вправо

console.log(result6.toString(2)); // Вывод: «101"

**Основы HTML/CSS**

**1. Значения атрибута display:**

Атрибут `display` используется в CSS для определения отображения элемента на веб-странице. Значение этого атрибута определяет, как элемент будет показываться на странице. Вот некоторые распространенные значения атрибута `display`:

1. `block`: Элемент отображается как блочный элемент и занимает всю доступную горизонтальную ширину. Новые элементы начинаются с новой строки.

2. `inline`: Элемент отображается как строчный элемент и не прерывает текущую строку текста. Он занимает только необходимое для своего содержимого пространство.

3. `inline-block`: Элемент отображается как строчный блок. Он занимает только необходимое пространство для своего содержимого, при этом его можно стилизовать, как блочный элемент.

4. `none`: Элемент не отображается на странице. Он полностью скрывается и не занимает места в макете страницы.

5. `flex`: Элемент отображается как гибкий контейнер, который позволяет управлять распределением элементов внутри него. Он позволяет использовать мощные функции построения адаптивных макетов.

6. `grid`: Элемент отображается как контейнер сетки. С помощью этого значения можно создавать сложные макеты с разделением на строки и столбцы.

**2. Веса селекторов:**

Вес селектора (specificity) - это показатель, определяющий приоритет применения стилей к элементам на веб-странице. Когда применяются несколько правил стилей к одному элементу, вес селектора используется для определения, какое правило будет преобладать.

Вес селектора состоит из нескольких компонентов, каждый из которых имеет свое значение:

1. Идентификаторы (#id):

- Идентификаторы имеют наивысший вес, так как они уникальны для каждого элемента на странице. Использование селектора с идентификатором будет иметь больший вес по сравнению с другими селекторами.

- Пример: `#myElement`.

2. Классы, псевдоклассы и атрибуты (.class, :pseudo-class, [attribute]):

- Классы, псевдоклассы и атрибуты имеют средний вес. Они используются для стилизации групп элементов с общими свойствами или определенных состояний элементов.

- Примеры: `.myClass`, `:hover`, `[type="text"]`.

3. Теги и псевдоэлементы (tag, ::pseudo-element):

- Теги и псевдоэлементы имеют наименьший вес. Они применяются к определенным типам элементов или частям элемента.

- Примеры: `div`, `::before`, `::after`.

В случае конфликта стилей для одного элемента используется селектор с более высоким весом. Если у двух селекторов одинаковый вес, то более поздний по порядку стиль будет применен.

Например, рассмотрим следующий код:

<div id="myElement" class="myClass">Hello, World!</div>

#myElement {

color: red; /\* Вес: 100 (идентификатор) \*/

}

.myClass {

color: blue; /\* Вес: 10 (класс) \*/

}

div {

color: green; /\* Вес: 1 (тег) \*/

}

В этом случае стиль с идентификатором `#myElement` будет иметь наивысший вес и перекроет стили с классом `.myClass` и тегом `div`. Таким образом, текст "Hello, World!" будет отображаться красным цветом.

**3. Псевдоклассы и псевдоэлементы:**

- Псевдоклассы - это ключевые слова, добавляемые к селекторам, чтобы указать определенное состояние элемента или его позицию в документе. Например, :hover указывает на элемент, когда курсор наводится на него. Примеры псевдоклассов: :hover, :active, :focus и :nth-child.

- Псевдоэлементы - это части элементов, которые могут быть выбраны и стилизованы независимо от содержимого элемента. Они указываются с помощью двойного двоеточия (::). Примеры псевдоэлементов: ::before, ::after, ::first-letter и ::first-line.

Псевдоклассы:

1. :hover - Применяет стили, когда курсор находится над элементом.

2. :active - Применяет стили, когда элемент активен (нажатие кнопки мыши).

3. :focus - Применяет стили, когда элемент получает фокус (например, при нажатии на поле ввода).

4. :visited - Применяет стили к посещенным ссылкам.

5. :first-child - Применяет стили к первому дочернему элементу родителя.

6. :last-child - Применяет стили к последнему дочернему элементу родителя.

7. :nth-child(n) - Применяет стили к элементу, который является n-м дочерним элементом родителя (можно использовать различные формулы, например, :nth-child(2n) для выбора каждого второго элемента).

8. :nth-of-type(n) - Применяет стили к элементу, который является n-м элементом выбранного типа (например, :nth-of-type(3) выберет третий элемент типа).

Псевдоэлементы:

1. ::before - Вставляет псевдоэлемент перед содержимым выбранного элемента.

2. ::after - Вставляет псевдоэлемент после содержимого выбранного элемента.

3. ::first-letter - Применяет стили к первой букве содержимого элемента.

4. ::first-line - Применяет стили к первой строке содержимого элемента.

5. ::selection - Применяет стили к выделенному пользователем тексту.

Примеры использования псевдоклассов и псевдоэлементов:

a:hover {

color: red; /\* Изменение цвета ссылки при наведении \*/

}

input:focus {

border: 2px solid blue; /\* Изменение границы поля ввода при фокусировке \*/

}

ul li:first-child {

font-weight: bold; /\* Жирный шрифт для первого элемента списка \*/

}

p::first-letter {

font-size: 2em; /\* Увеличение размера первой буквы в параграфе \*/

}

div::before {

content: "Before content"; /\* Вставка текста перед содержимым блока \*/

}

div::after {

content

: "After content"; /\* Вставка текста после содержимого блока \*/

}

**4. Box model:**

Box model (модель блока) является основным понятием в CSS, определяющим, как элементы HTML располагаются и взаимодействуют друг с другом на веб-странице. Она описывает, как браузер выделяет пространство для элементов и какова их структура.

Box model включает в себя четыре основных компонента, которые определяют размер и расположение элемента:

1. Content (содержимое):

- Это внутренняя область элемента, которая содержит фактическое содержимое, такое как текст, изображения или другие вложенные элементы.

- Размер содержимого определяется свойствами `width` (ширина) и `height` (высота).

2. Padding (внутренний отступ):

- Внутренний отступ является пространством между содержимым элемента и его границей. Он создает отступ вокруг содержимого.

- Величина внутреннего отступа определяется свойствами `padding-top`, `padding-right`, `padding-bottom` и `padding-left`.

3. Border (граница):

- Граница представляет собой линию, окружающую содержимое и внутренний отступ элемента. Она разделяет элемент от соседних элементов.

- Свойство `border` используется для задания ширины, стиля и цвета границы.

4. Margin (внешний отступ):

- Внешний отступ представляет собой пространство вокруг элемента, разделяющее его с другими элементами на странице.

- Размер внешнего отступа определяется свойствами `margin-top`, `margin-right`, `margin-bottom` и `margin-left`.

Таким образом, модель блока состоит из содержимого, внутреннего отступа, границы и внешнего отступа, которые образуют прямоугольную область вокруг элемента.

Пример CSS-кода, демонстрирующий применение модели блока:

.box {

width: 200px;

height: 100px;

padding: 20px;

border: 1px solid #000;

margin: 10px;

}

**5. Относительные величины**

Относительные величины в CSS позволяют задавать размеры и расстояния элементов относительно других элементов или контекста страницы. Использование относительных величин способствует созданию более гибкого и адаптивного дизайна, который может легко масштабироваться на разных устройствах.

Ниже перечислены некоторые распространенные относительные величины:

1. em:

- Значение `em` определяет размеры элемента относительно размера шрифта его родителя.

- Например, если размер шрифта родительского элемента равен 16 пикселям, и у дочернего элемента задан размер `2em`, то размер дочернего элемента будет равен 32 пикселям (2 \* 16).

2. rem:

- Значение `rem` задает размеры элемента относительно размера шрифта корневого элемента (`<html>`).

- Например, если размер шрифта корневого элемента равен 16 пикселям, и у элемента задан размер `2rem`, то размер элемента будет равен 32 пикселям (2 \* 16).

3. %:

- Значение `%` позволяет задавать размер элемента в процентах от размеров родительского элемента.

- Например, если у элемента задана ширина `50%`, то он будет занимать половину ширины своего родительского элемента.

4. vw и vh:

- Значения `vw` и `vh` определяют размер элемента в процентах от размеров видимой области окна браузера.

- Значение `1vw` равно 1% ширины видимой области, а `1vh` равно 1% высоты видимой области.

5. vmin и vmax:

- Значения `vmin` и `vmax` позволяют задавать размеры элемента в процентах от наименьшей или наибольшей размерности видимой области окна браузера.

- `vmin` берет во внимание наименьшую размерность (ширину или высоту), а `vmax` - наибольшую.

Примеры использования относительных величин:

.parent {

font-size: 20px;

}

.child {

font-size: 1.5em; /\* Размер шрифта будет равен 30 пикселям (1.5 \* 20) \*/

}

.container {

width: 50%; /\* Ширина контейнера будет равна 50% ширины родителя \*/

}

.box {

width: 50vw; /\* Ширина элемента будет равна

50% ширины окна браузера \*/

}

**6. Позиционирование:**   
Позиционирование в CSS позволяет контролировать расположение элементов на веб-странице. С помощью свойства `position` и связанных с ним свойств можно задавать различные способы позиционирования элементов. Вот некоторые из них:

1. Поток документа (static):

- Это значение по умолчанию для свойства `position`. Элементы позиционируются в соответствии с обычным потоком документа, их расположение определяется в порядке их появления в HTML-коде.

- Для элементов с `position: static` игнорируются свойства `top`, `right`, `bottom` и `left`.

2. Относительное позиционирование (relative):

- Элементы с `position: relative` позиционируются относительно своего нормального местоположения.

- Можно использовать свойства `top`, `right`, `bottom` и `left`, чтобы сдвигать элемент относительно его исходной позиции.

3. Абсолютное позиционирование (absolute):

- Элементы с `position: absolute` позиционируются относительно ближайшего родительского элемента с позиционированием, отличным от `position: static`.

- Можно использовать свойства `top`, `right`, `bottom` и `left`, чтобы точно указать позицию элемента.

4. Фиксированное позиционирование (fixed):

- Элементы с `position: fixed` позиционируются относительно окна браузера и остаются на своем месте при прокрутке страницы.

- Можно использовать свойства `top`, `right`, `bottom` и `left`, чтобы задать фиксированную позицию элемента.

5. Закрепленное позиционирование (sticky):

- Элементы с `position: sticky` позиционируются относительно своего ближайшего контейнера прокрутки или родительского элемента с прокруткой.

- Элемент будет прилипать к определенной позиции, когда пользователь прокручивает страницу.

**7. Flexbox:**

Flexbox (гибкая модель разметки) - это техника в CSS для создания гибких и адаптивных макетов на веб-странице. Она предоставляет набор свойств для контроля над расположением и выравниванием элементов внутри родительского контейнера.

Основные концепции и свойства Flexbox:

1. Flex-контейнер (Flex Container):

- Это родительский элемент, который содержит дочерние элементы и определяет контекст для применения гибкой модели разметки.

- Чтобы создать Flexbox-контейнер, нужно применить к элементу свойство `display` со значением `flex` или `inline-flex`.

2. Flex-элементы (Flex Items):

- Это дочерние элементы, которые находятся внутри Flex-контейнера и подвергаются гибкой модели разметки.

- Flex-элементы могут изменять свою ширину, высоту и порядок, чтобы адаптироваться к различным условиям.

3. Основные свойства Flexbox:

- `flex-direction`: Устанавливает направление основной оси (main axis) внутри Flex-контейнера. Определяет, будет ли контейнер располагать элементы в строку или в столбец.

- `justify-content`: Определяет выравнивание элементов вдоль основной оси.

- `align-items`: Определяет выравнивание элементов вдоль поперечной оси (cross axis).

- `flex-wrap`: Определяет, должны ли элементы переноситься на новую строку или оставаться в одной строке при нехватке места.

- `align-content`: Определяет выравнивание строк Flex-элементов вдоль поперечной оси, если они занимают несколько строк.

4. Дополнительные свойства Flexbox:

- `flex-grow`: Определяет, насколько элемент может растягиваться внутри контейнера, если есть свободное пространство.

- `flex-shrink`: Определяет, насколько элемент может сжиматься внутри контейнера, если места недостаточно.

- `flex-basis`: Определяет исходный размер элемента до распределения свободного пространства или сжатия.

- `order`: Определяет порядок отображения Flex-элементов внутри контейнера.

Пример использования Flexbox:

.container {

display: flex;

flex-direction: row;

justify-content: center;

align-items: center;

}

.item {

flex: 1;

margin: 10px;

}

В этом примере `.container` будет являться Flex-контейнером, а `.item` - Flex-элементами. Свойство `flex-direction` задает расположение элементов в строку (`row`), `justify-content` центрирует элементы вдоль главной оси, а `align-items` центрирует их вдоль поперечной оси. Свойство `flex: 1` указывает, что элементы должны занимать равное пространство, а `margin` задает отступы между элементами.

**8. Grid:**

Grid (сетка) - это мощный инструмент в CSS, который позволяет создавать гибкие и сложные макеты на веб-странице. Он предоставляет возможность размещать элементы в виде сетки с ячейками и задавать их размеры, выравнивание и расположение с помощью гибких свойств.

Основные концепции и свойства Grid:

1. Grid-контейнер (Grid Container):

- Это родительский элемент, который содержит дочерние элементы и определяет контекст для применения сетки.

- Чтобы создать Grid-контейнер, нужно применить к элементу свойство `display` со значением `grid` или `inline-grid`.

2. Grid-ячейки (Grid Items):

- Это дочерние элементы, которые находятся внутри Grid-контейнера и размещаются в ячейках сетки.

- Grid-ячейки могут быть пустыми или содержать контент.

3. Основные свойства Grid:

- `grid-template-rows` и `grid-template-columns`: Определяют размеры и количество ячеек в строках и столбцах сетки.

- `grid-gap` или `gap`: Устанавливают промежутки между ячейками.

- `justify-items`: Определяет горизонтальное выравнивание Grid-ячеек внутри их ячейки.

- `align-items`: Определяет вертикальное выравнивание Grid-ячеек внутри их ячейки.

- `justify-content`: Определяет горизонтальное выравнивание Grid-ячеек внутри Grid-контейнера.

- `align-content`: Определяет вертикальное выравнивание Grid-ячеек внутри Grid-контейнера.

4. Дополнительные свойства Grid:

- `grid-template-areas`: Определяет расположение и имена областей сетки, позволяя группировать ячейки внутри них.

- `grid-auto-rows` и `grid-auto-columns`: Определяют размеры и количество ячеек, которые будут созданы автоматически, если их не хватает в явно определенной сетке.

- `grid-auto-flow`: Определяет порядок автоматического размещения Grid-ячеек, когда они не помещаются в явно определенную сетку.

Пример использования Grid:

.container {

display: grid;

grid-template-columns: 1fr 2fr 1fr;

grid-template-rows: 100px 200px;

grid-gap: 10px;

}

.item

{

background-color: #eaeaea;

padding: 10px;

}

.item-1 {

grid-area: 1 / 1 / 2 / 4;

}

В этом примере `.container` является Grid-контейнером, у которого заданы размеры и количество ячеек в строках и столбцах сетки. Свойство `grid-gap` задает промежутки между ячейками. `.item` - это Grid-ячейки с фоновым цветом и отступами. `.item-1` с помощью свойства `grid-area` задает расположение в ячейках сетки.

**Основы JS**

**1. Типы данных:**

В JavaScript существуют различные типы данных, которые используются для представления различных видов значений. Вот некоторые из основных типов данных в JavaScript:

1. Числа (Numbers):

- Числа в JavaScript могут быть целыми или числами с плавающей точкой.

- Примеры чисел:

var age = 25;

var price = 9.99;

2. Строки (Strings):

- Строки представляют последовательности символов и заключаются в одинарные или двойные кавычки.

- Примеры строк:

var name = "John";

var message = 'Привет, как дела?';

3. Булевый тип (Boolean):

- Булевый тип представляет логические значения `true` (истина) и `false` (ложь).

- Булевые значения используются для логических операций и условных выражений.

- Примеры булевых значений:

var isTrue = true;

var isFalse = false;

4. Массивы (Arrays):

- Массивы представляют упорядоченные списки элементов.

- Каждый элемент в массиве имеет свой индекс, начиная с 0.

- Пример массива:

var numbers = [1, 2, 3, 4, 5];

var fruits = ["яблоко", "банан", "апельсин"];

5. Объекты (Objects):

- Объекты представляют коллекции свойств и значений.

- Свойства объекта состоят из пар ключ-значение.

- Пример объекта:

var person = {

name: "John",

age: 25,

city: "New York"

};

6. Null и Undefined:

- Значение `null` указывает на отсутствие значения.

- Значение `undefined` указывает на переменную, которая не имеет присвоенного значения.

- Примеры:

var value = null;

var notDefined;

Кроме этих основных типов данных, в JavaScript также существуют функции, даты и другие сложные типы данных. Понимание различных типов данных поможет вам эффективно работать с данными и выполнять различные операции в JavaScript.

**2. Переменные:**

- Объявление переменной происходит с помощью ключевого слова `var`, `let` или `const` и затем указания имени переменной.  
Область видимости переменной определяет, где переменная доступна и может быть использована.

Переменные, объявленные с помощью var, имеют функциональную область видимости. Они видны только внутри функции, в которой они объявлены.

Переменные, объявленные с помощью let и const, имеют блочную область видимости. Они видны только внутри блока, в котором они объявлены (например, внутри циклов или условных операторов).

- Примеры:

var age = 25;

let name = "John";

const PI = 3.14;

**3. Преобразование типов:**

В JavaScript существует автоматическое и явное преобразование типов данных. Преобразование типов позволяет изменять тип значения переменной в другой тип данных. Вот некоторые основные способы преобразования типов данных в JavaScript:

1. Явное преобразование:

- Явное преобразование типов осуществляется с использованием определенных функций и операторов.

- `Number()`: Используется для явного преобразования значения в числовой тип данных.

var str = "42";

var num = Number(str); // num = 42

- `String()`: Используется для явного преобразования значения в строковый тип данных.

var num = 42;

var str = String(num); // str = "42"

- `Boolean()`: Используется для явного преобразования значения в логический (булевый) тип данных.

var num = 42;

var bool = Boolean(num); // bool = true

2. Автоматическое преобразование:

- Автоматическое преобразование типов выполняется JavaScript автоматически при выполнении операций или сравнений между значениями разных типов.

- Пример автоматического преобразования типов:

var num = 42;

var str = "The answer is " + num; // str = "The answer is 42"

3. Преобразование в число:

- При выполнении математических операций, JavaScript автоматически преобразует значения в числа.

- Пример преобразования в число:

var str = "42";

var result = Number(str) + 8; // result = 50

4. Преобразование в строку:

- Когда значение используется в контексте, ожидающем строку, JavaScript автоматически преобразует его в строку.

- Пример преобразования в строку:

var num = 42;

var str = "The number is " + num; // str = "The number is 42"

5. Преобразование в логический тип:

- Логическое преобразование выполняется, когда значение используется в условном выражении или логической операции.

- Пример преобразования в логический тип:

var num = 42;

var boolValue = Boolean(num); // boolValue = true

**4. Базовые операторы:**

- Арифметические операторы: `+`, `-`, `\*`, `/`, `%`.

- Операторы присваивания: `=`, `+=`, `-=`, `\*=`, `/=`.

- Операторы инкремента и декремента: `++`, `--`.

- Операторы сравнения: `==`, `===`, `!=`, `!==`, `>`, `<`, `>=`, `<=`.

**5. Операторы сравнения:**

- Операторы сравнения используются для сравнения значений и возвращают булево значение `true` или `false`.

- Примеры:

var x = 5;

var y = 10;

console.log(x > y); // false

console.log(x === y); // false

console.log(x !== y); // true

**6. Циклы:**

В JavaScript циклы используются для выполнения повторяющихся операций. Они позволяют многократно выполнять блок кода до тех пор, пока заданное условие истинно. Вот некоторые из наиболее часто используемых циклов в JavaScript:

1. Цикл `for`:

- Цикл `for` позволяет выполнить блок кода заданное количество раз.

- Он состоит из трех частей: инициализация, условие и инкремент/декремент.

- Пример использования цикла `for`:

for (var i = 0; i < 5; i++) {

console.log(i);

}

2. Цикл `while`:

- Цикл `while` выполняет блок кода до тех пор, пока заданное условие истинно.

- Условие проверяется перед выполнением блока кода.

- Пример использования цикла `while`:

var i = 0;

while (i < 5) {

console.log(i);

i++;

}

3. Цикл `do...while`:

- Цикл `do...while` похож на цикл `while`, но проверка условия происходит после выполнения блока кода.

- Поэтому блок кода всегда выполняется хотя бы один раз.

- Пример использования цикла `do...while`:

var i = 0;

do {

console.log(i);

i++;

} while (i < 5);

4. Цикл `for...in`:

- Цикл `for...in` используется для перебора свойств объекта.

- Он выполняет итерацию по всем перечисляемым свойствам объекта.

- Пример использования цикла `for...in`:

var person = {

name: "John",

age: 30,

city: "New York"

};

for (var key in person) {

console.log(key + ": " + person[key]);

}

5. Цикл `for...of`:

- Цикл `for...of` используется для перебора элементов коллекций, таких как массивы или строки.

- Он выполняет итерацию по каждому элементу коллекции.

- Пример использования цикла `for...of`:

var fruits = ["apple", "banana", "orange"];

for (var fruit of fruits) {

console.log(fruit);

}

Циклы позволяют автоматизировать выполнение повторяющихся задач и обрабатывать большие объемы данных. Они играют важную роль в программировании и помогают сделать код более эффективным и

гибким.

**7. Условные операторы:**

В JavaScript условные операторы используются для выполнения различных блоков кода в зависимости от условий. Они позволяют программе принимать решения на основе истинности или ложности определенных выражений. Ниже приведены основные условные операторы в JavaScript:

1. Оператор `if`:

- Оператор `if` выполняет блок кода, если заданное условие истинно.

- Синтаксис:

if (условие) {

// выполняемый код, если условие истинно

}

- Пример использования оператора `if`:

var age = 20;

if (age >= 18) {

console.log("Вы совершеннолетний");

}

2. Оператор `if...else`:

- Оператор `if...else` выполняет блок кода, если условие истинно, и альтернативный блок кода, если условие ложно.

- Синтаксис:

if (условие) {

// выполняемый код, если условие истинно

} else {

// выполняемый код, если условие ложно

}

- Пример использования оператора `if...else`:

var age = 20;

if (age >= 18) {

console.log("Вы совершеннолетний");

} else {

console.log("Вы несовершеннолетний");

}

3. Оператор `if...else if...else`:

- Оператор `if...else if...else` позволяет проверить несколько условий и выполнить соответствующий блок кода.

- Синтаксис:

if (условие1) {

// выполняемый код, если условие1 истинно

} else if (условие2) {

// выполняемый код, если условие1 ложно и условие2 истинно

} else {

// выполняемый код, если все условия ложны

}

- Пример использования оператора `if...else if...else`:

var age = 20;

if (age < 13) {

console.log("Вы ребенок");

} else if (age < 18) {

console.log("Вы подросток");

} else {

console.log("Вы взрослый");

}

4. Оператор `switch`:

- Оператор `switch` позволяет проверить различные значения выражения и выполнить соответствующий блок кода.

- Синтаксис:

switch (выражение) {

case значение1:

// выполняемый код, если выражение

равно значению1

break;

case значение2:

// выполняемый код, если выражение равно значению2

break;

default:

// выполняемый код, если ни одно из значений не совпало

break;

}

- Пример использования оператора `switch`:

var day = "пятница";

switch (day) {

case "понедельник":

console.log("Сегодня понедельник");

break;

case "вторник":

case "среда":

case "четверг":

console.log("Сегодня будний день");

break;

case "пятница":

console.log("Сегодня пятница! Пятница - лучший день!");

break;

default:

console.log("Сегодня выходной");

break;

}