**Унарні оператори**

Унарні оператори в JavaScript - це особливий вид операторів, які застосовуються лише до одного операнда для зміни його стану або значення. Вони є важливими інструментами в мові програмування, оскільки дозволяють виконувати оперативні зміни, такі як збільшення чи зменшення числових значень, інвертування знаку числа, визначення типу даних змінної, а також приведення значень до числа.

Приклади унарних операторів в JavaScript включають:

Унарний оператор мінус **(**-**)** - змінює знак числа на протилежний.

var c = 3;

console.log(-c); // Виведе: -3

Унарний оператор плюс (+) - спробує перетворити операнд на число, якщо це можливо (детально розглянемо далі).

var e = "5";

console.log(+e); // Виведе: 5

var f = "hello";

console.log(+f); // Виведе: NaN, оскільки "hello" не може бути перетворено на число

Оператор декрементації (--) - автоматично зменшує значення змінної на одиницю.

var b = 5;

console.log(--b); // Виведе: 4

Оператор інкрементації (++) - автоматично збільшує значення змінної на одиницю.

var a = 5;

console.log(++a); // Виведе: 6

Оператор typeof - визначає тип даних операнда.

var d = true;

console.log(typeof d); // Виведе: "boolean"

Ці унарні оператори надають зручні механізми для виконання швидких операцій зі змінними або виразами, спрощуючи код та покращуючи його читабельність. Вони є фундаментальними будівельними блоками в JavaScript, які використовуються у широкому спектрі сценаріїв програмування.

## Префіксний та постфіксний унарні оператори

У JavaScript префіксний та постфіксний унарні оператори інкремента (++) та декремента (--) дозволяють нам збільшувати або зменшувати значення змінних на одиницю. Використовуються вони для спрощення коду, коли потрібно модифікувати числове значення змінної, наприклад, при переборі елементів масиву або в циклах.

### Префіксний інкремент та декремент

Префіксний інкремент (++var) або декремент (--var) збільшує або зменшує значення змінної на одиницю перед тим, як повернути значення. Це означає, що спочатку відбувається модифікація значення, а потім воно використовується у виразі.

var num1 = 5;

console.log(++num1); // 6

console.log(--num1); // 5

У цьому прикладі, спочатку num1 збільшується на 1, тому console.log() виводить 6. Потім num1 зменшується на 1, тому наступний console.log() виводить 5.

### Постфіксний інкремент та декремент

Постфіксний інкремент (var++) або декремент (var--) спочатку повертає поточне значення змінної, а потім збільшує або зменшує його на одиницю. Це означає, що використання значення відбувається до його модифікації.

var num2 = 5;

console.log(num2++); // 5

console.log(num2); // 6

console.log(num2--); // 6

console.log(num2); // 5

У цьому прикладі, коли виконується console.log(num2++), спочатку виводиться поточне значення num2 (5), а потім воно збільшується на 1. Наступний console.log(num2) виводить уже модифіковане значення (6). Аналогічно, console.log(num2--) спочатку виводить 6, а потім значення num2 зменшується на 1.

Ці оператори широко застосовуються для контролю ітерацій у циклах та при роботі з даними, які потребують поступового збільшення або зменшення.

**Бінарні оператори**

Бінарні оператори в JavaScript - це оператори, що вимагають два операнди для виконання операції. Вони застосовуються для виконання арифметичних обчислень, порівняння значень, логічних операцій та для присвоєння значень. Бінарні оператори дозволяють нам виконувати різноманітні операції між змінними та літералами, що є основою логіки багатьох програмних алгоритмів.

Приклади бінарних операторів:

**Арифметичні оператори** (наприклад, +, -, \*, /) використовуються для виконання математичних обчислень між числами.

var sum = 10 + 5; // 15

var difference = 10 - 5; // 5

var product = 10 \* 5; // 50

var quotient = 10 / 5; // 2

Оператори порівняння (наприклад, ==, !=, >, <) використовуються для порівняння двох значень і повертають булеве значення (true або false) в залежності від результату порівняння.

var isEqual = (10 == 5); // false

var isNotEqual = (10 != 5); // true

var isGreater = (10 > 5); // true

var isLess = (10 < 5); // false

Логічні оператори (&&, ||) використовуються для виконання логічних операцій між двома булевими значеннями або виразами.

var andOperation = (true && false); // false

var orOperation = (true || false); // true

Оператор присвоєння (=) використовується для присвоєння значення правого операнда змінній, що є лівим операндом.

var number = 10; // Змінній number присвоєно значення 10

Бінарні оператори є невід'ємною частиною мови програмування JavaScript, дозволяючи нам виконувати обчислення, порівняння та логічні операції, необхідні для розв'язання різноманітних програмних задач. Вони спрощують код, роблячи його більш читабельним та ефективним.

**Присвоєння з операцією**

Присвоєння з операцією в JavaScript є способом спрощення коду, який дозволяє нам комбінувати арифметичні, бітові або логічні операції з присвоєнням значення змінній. Використовуючи ці оператори, ми можемо зменшити кількість необхідного коду для виконання операцій та присвоєння результату цих операцій змінним. Це робить код більш лаконічним та покращує його читабельність.

Приклади присвоєння з операцією:

Оператор присвоєння з додаванням **(**+=**)** - додає правий операнд до лівого та присвоює результат лівому операнду.

var a = 10;

a += 5; // Еквівалентно a = a + 5; Тепер a дорівнює 15

Оператор присвоєння з відніманням (-=) - віднімає правий операнд з лівого та присвоює результат лівому операнду.

var b = 10;

b -= 5; // Еквівалентно b = b - 5; Тепер b дорівнює 5

Оператор присвоєння з множенням (\*=) - множить лівий операнд на правий та присвоює результат лівому операнду.

var c = 10;

c \*= 5; // Еквівалентно c = c \* 5; Тепер c дорівнює 50

Оператор присвоєння з діленням (/=) - ділить лівий операнд на правий та присвоює результат лівому операнду.

var d = 10;

d /= 5; // Еквівалентно d = d / 5; Тепер d дорівнює 2

Оператор присвоєння з остачею від ділення (%=) - знаходить остачу від ділення лівого операнда на правий та присвоює результат лівому операнду.

var e = 10;

e %= 3; // Еквівалентно e = e % 3; Тепер e дорівнює 1

Використання присвоєння з операцією дозволяє нам виконувати обчислення та одночасно оновлювати значення змінних, що ефективно спрощує код та робить його більш зрозумілим.

## Оператори відносин & Об'єкт Math

### Оператори порівнянь

Оператори порівнянь у JavaScript використовуються для порівняння двох значень чи виразів, результатом чого є булеве значення: true (істина) або false (хиба). Ці оператори дозволяють нам виконувати логічні порівняння, що є фундаментальною частиною умовних конструкцій та рішень, які приймаються під час виконання програми.

Основні оператори відносин:

* == (рівно) - перевіряє, чи рівні операнди за значенням.
* != (не рівно) - перевіряє, чи не рівні операнди за значенням.
* === (строго рівно) - перевіряє, чи рівні операнди за значенням та типом.
* !== (строго не рівно) - перевіряє, чи не рівні операнди за значенням та типом.
* > (більше) - перевіряє, чи лівий операнд більший за правий.
* < (менше) - перевіряє, чи лівий операнд менший за правий.
* >= (більше або рівно) - перевіряє, чи лівий операнд більший або рівний правому.
* <= (менше або рівно) - перевіряє, чи лівий операнд менший або рівний правому.

Приклад коду:

var numberOne = 5;

var numberTwo = 10;

// Перевіряємо, чи numberOne менше numberTwo

if (numberOne < numberTwo) {

console.log("numberOne менше за numberTwo"); // Виведе: "numberOne менше за numberTwo"

}

// Перевіряємо, чи рівні numberOne та numberTwo за значенням та типом

if (numberOne === numberTwo) {

console.log("numberOne і numberTwo рівні за значенням та типом");

} else {

console.log("numberOne і numberTwo не рівні за значенням та типом"); // Виведе: "numberOne і numberTwo не рівні за значенням та типом"

}

Використання операторів відносин дуже важливе для управління логікою програми, особливо при прийнятті рішень усередині умовних конструкцій, таких як if та циклах. Вони дозволяють програмі реагувати на різні умови та виконувати відповідні дії.

### Об'єкт Math

Об'єкт Math в JavaScript є вбудованим об'єктом, який надає велику кількість математичних функцій та констант. Ми використовуємо його для виконання складних математичних обчислень, таких як знаходження максимальних або мінімальних значень, округлення чисел, виконання тригонометричних обчислень та багато іншого без необхідності самостійного написання відповідних функцій з нуля.

Приклади використання об'єкта Math:

Отримання максимального та мінімального значення з набору чисел:

var maxNumber = Math.max(10, 20, 30, 40, 50);

console.log(maxNumber); // Виведе: 50

var minNumber = Math.min(10, 20, 30, 40, 50);

console.log(minNumber); // Виведе: 10

Округлення числа:

var roundedNumber = Math.round(4.7);

console.log(roundedNumber); // Виведе: 5

var floorNumber = Math.floor(4.7);

console.log(floorNumber); // Виведе: 4

Знаходження квадратного кореня числа:

var sqrtNumber = Math.sqrt(16);

console.log(sqrtNumber); // Виведе: 4

Піднесення числа до степеня:

var powerNumber = Math.pow(2, 4);

console.log(powerNumber); // Виведе: 16

Генерація випадкового числа:

var randomNumber = Math.random();

console.log(randomNumber); // Виведе випадкове число між 0 (включно) та 1 (не включно)

Об'єкт Math є незамінним інструментом у розробці, коли потрібно виконувати математичні операції. Він дозволяє значно спростити код та зробити його більш читабельним, уникаючи необхідності в реалізації базових математичних функцій власноруч.

## String

У JavaScript, String є примітивним типом даних, який використовується для представлення тексту. Ми використовуємо рядки для зберігання та маніпуляції текстом, таким як імена користувачів, повідомлення, тексти на веб-сторінках та багато іншого. Рядки можна створювати за допомогою одинарних, подвійних або зворотних лапок, що дозволяє включати в текст змінні та вирази.

Приклади використання рядків:

Створення рядків:

var greeting = 'Привіт, світе!';

console.log(greeting); // Виведе: Привіт, світе!

Об'єднання рядків:

var firstName = 'Іван';

var lastName = 'Іваненко';

var fullName = firstName + ' ' + lastName;

console.log(fullName); // Виведе: Іван Іваненко

Використання шаблонних літералів для вставки змінних та виразів:

var firstName = 'Іван';

var lastName = 'Іваненко';

var fullName = firstName + ' ' + lastName;

var age = 25;

var introduction = `Мене звати ${fullName} і мені ${age} років.`;

console.log(introduction); // Виведе: Мене звати Іван Іваненко і мені 25 років.

Зауважте, що для шаблонних літералів використовуються **зворотні лапки**.

Доступ до символів у рядку:

var greeting = 'Привіт, світе!';

var letter = greeting[8]; // Отримуємо дев'ятий символ з рядка greeting

console.log(letter); // Виведе: с

Довжина рядка:

var greeting = 'Привіт, світе!';

var length = greeting.length;

console.log(length); // Виведе кількість символів у рядку 'Привіт, світе!'

Рядки в JavaScript є надзвичайно гнучкими та потужними інструментами для роботи з текстом. Вони дозволяють з легкістю форматувати, об'єднувати та маніпулювати текстовими даними, що є важливою частиною майже будь-якої програми або веб-сайту.

## Методи рядків & Boolean

Методи рядків у JavaScript - це вбудовані функції, які ми використовуємо для маніпуляції рядками, наприклад, для пошуку підрядків, заміни фрагментів тексту, перетворення регістру та багато іншого. Вони дозволяють нам з легкістю обробляти та модифікувати текстові дані.

Приклади використання методів рядків:

* **Доступ до символу за індексом**:

Замість charAt(), ми часто використовуємо доступ через [] для отримання символу за певним індексом.

var text = 'Привіт, світе!';

console.log(text.charAt(8)); // Виведе: с

console.log(text[8]); // Виведе: с

* **substring(start, end)** - повертає підрядок з рядка між двома індексами.

var text = 'Привіт, світе!';

console.log(text.substring(0, 6)); // Виведе: Привіт

* **slice(start, end)** - подібно до substring(), але може приймати від'ємні індекси.

var text = 'Привіт, світе!';

console.log(text.slice(-6, -1)); // Виведе: світе

* **substr(start, length)** - повертає підрядок, починаючи з заданого індексу та заданої довжини.

var text = 'Привіт, світе!';

console.log(text.substr(1, 4)); // Виведе: риві

* **indexOf(substring)** - повертає індекс першого входження підрядка в рядок.

var text = 'Привіт, світе!';

console.log(text.indexOf('світе')); // Виведе: 8

* **lastIndexOf(substring)** - повертає індекс останнього входження підрядка.

var text = 'Привіт, світе!';

console.log(text.lastIndexOf('і')); // Виведе: 10

* **replace(oldSubstring, newSubstring)** - замінює перше входження підрядка на новий.

var text = 'Привіт, світе!';

console.log(text.replace('світе', 'друже')); // Виведе: Привіт, друже!

* **split(separator)** - розбиває рядок на масив рядків за допомогою роздільника.

console.log('a,b,c'.split(',')); // Виведе: ['a', 'b', 'c']

* **toUpperCase()** та **toLowerCase()** - змінюють регістр всіх символів рядка.

var text = 'Привіт, світе!';

console.log(text.toUpperCase()); // Виведе: ПРИВІТ, СВІТЕ!

console.log(text.toLowerCase()); // Виведе: привіт, світе!

* **includes(substring)** - перевіряє, чи містить рядок заданий підрядок.

var text = 'Привіт, світе!';

console.log(text.includes('світе')); // Виведе: true

Ці методи є ключовими інструментами для ефективної роботи з текстом у JavaScript, дозволяючи нам легко маніпулювати та обробляти рядки для різноманітних завдань програмування.

### Boolean

Логічний тип даних (Boolean) у JavaScript є фундаментальним типом, який може приймати лише два значення: true (істинно) або false (хибно). Ми використовуємо логічний тип даних для управління потоком програми через умовні конструкції, такі як if-else вирази, цикли та для контролю логіки булевих виразів у логічних операціях.

Логічний тип даних є невід'ємною частиною програмування, оскільки він дозволяє виконувати операції, засновані на умовах, та керувати поведінкою програми в залежності від істинності чи хибності деяких умов.

Приклад використання логічного типу даних:

var isRainy = true;

var isSunny = false;

// Використання логічних значень у умовному операторі

if (isRainy) {

console.log('Візьміть парасольку!');

} else {

console.log('Сьогодні парасолька не потрібна.');

}

// Логічне "І" та "АБО"

if (isRainy && isSunny) {

console.log('Можлива веселка!');

} else if (isRainy || isSunny) {

console.log('Погода змінюється.');

} else {

console.log('Незрозуміла погода.');

}

У цьому прикладі використовуються змінні isRainy та isSunny з логічними значеннями для демонстрації, як можна управляти потоком програми залежно від погодних умов. Логічні значення true та false допомагають визначити, які дії слід виконати в різних ситуаціях.

Логічний тип даних є ключовим для створення ефективних умовних конструкцій та управління логікою програм на всіх рівнях складності.

## Логічні оператори / Null / Undefined

### Логічні оператори

Логічні оператори в JavaScript використовуються для виконання операцій з логічними значеннями (true і false) та управління потоком виконання програми на основі умов. Вони дозволяють нам комбінувати, інвертувати або перевіряти істинність виразів та умов.

**Логічні оператори включають:**

* && (логічне "І") використовується для перевірки, чи істинні обидва операнди. Якщо обидва істинні, результат буде true; інакше - false.
* || (логічне "АБО") перевіряє, чи істинний хоча б один з операндів. Якщо хоча б один істинний, результат буде true.
* ! (логічне НЕ) інвертує булеве значення: перетворює true в false та навпаки.
* ?? (оператор нульового злиття) повертає лівий операнд, якщо він не null і не undefined; інакше повертає правий.

Приклади використання:

var isRainy = true;

var isSunny = false;

// Логічне "І"

if (isRainy && isSunny) {

console.log('Можлива веселка!');

} else {

console.log('Веселки не буде.');

}

// Логічне "АБО"

if (isRainy || isSunny) {

console.log('Погода змінлива.');

} else {

console.log('Погода одноманітна.');

}

// Логічне "НЕ"

if (!isSunny) {

console.log('Без сонячних променів.');

}

// Оператор нульового злиття

var guests = null;

var defaultGuests = 10;

var totalGuests = guests ?? defaultGuests;

console.log(totalGuests); // Виведе: 10

Ці оператори дуже важливі для розгалуження логіки програми, дозволяючи нам створювати більш складні умовні конструкції та ефективно управляти потоком виконання програми залежно від різних умов і ситуацій. Оператор нульового злиття (??) є особливо корисним для роботи з можливо відсутніми або не визначеними значеннями, дозволяючи задавати значення за замовчуванням.

### Null

У JavaScript, null є спеціальним значенням, яке представляє відсутність будь-якого значення або об'єкта. Ми використовуємо null для ініціалізації змінних, які можуть надалі отримати конкретне значення або об'єкт, але на початковому етапі явно не містять ніякого значення. Відповідно, null часто використовується для вказівки на те, що змінна повинна мати "порожнє" або "ніяке" значення.

\*\*null використовується для:\*\*

* Явної вказівки на відсутність значення змінної або об'єкта.
* Очищення посилань на об'єкти, щоб допомогти зі збіркою сміття (гарбідж колекцією).
* Відмінності від undefined, яке вказує на те, що змінна була оголошена, але не ініціалізована жодним значенням.

Приклад коду з описом:

var user = null;

if (user === null) {

console.log('Користувач не визначений.');

} else {

console.log(`Користувач: ${user}`);

}

// Пізніше в програмі, користувач отримує значення

user = 'Іван Петрович';

if (user !== null) {

console.log(`Користувач: ${user}`); // Виведе: Користувач: Іван Петрович

}

У цьому прикладі спочатку змінна user ініціалізована значенням null, що вказує на відсутність користувача або його невизначеність на даному етапі. Після цього, коли користувачу присвоюється конкретне значення, програма може виконувати різні дії, виходячи з цього. Використання null дозволяє нам явно керувати наявністю або відсутністю даних, покращуючи читабельність та надійність коду.

### Undefined

У JavaScript, undefined є типом даних, який вказує на відсутність визначеного значення. Змінна вважається undefined, якщо вона була оголошена, але їй не було присвоєно жодного значення. Це відрізняється від null, яке є способом явно вказати на відсутність будь-якого значення. Ми використовуємо undefined для ідентифікації змінних, які ще не мають присвоєного значення, а також для позначення випадків, коли об'єкт або функція не повертають значення.

\*\*undefined використовується для:\*\*

* Визначення стану змінних, яким ще не було присвоєно значення.
* Перевірки наявності властивостей в об'єктах або повернення значення з функцій, коли вони не мають явно визначеного результату.

Приклад коду з описом:

var user;

console.log(user); // Виведе: undefined

function getUser() {

// Функція без повернення значення явно повертає undefined

}

var result = getUser();

console.log(result); // Виведе: undefined

// Перевірка, чи змінна має визначене значення

if (user === undefined) {

console.log('Змінна user не має значення.');

} else {

console.log(`Змінна user має значення: ${user}`);

}

У цьому прикладі спочатку змінна user оголошена, але не ініціалізована, тому її значення - undefined. Також функція getUser() не має явно визначеного повернення, тому за замовчуванням вона повертає undefined. Використання undefined допомагає нам ідентифікувати та керувати змінними або поверненнями з функцій, які ще не були явно визначені або ініціалізовані.

**Наочна демонстрація null vs undefined:**

## Object

У JavaScript, об'єкт - це колекція властивостей, де кожна властивість має ім'я (або ключ) і відповідне їй значення, яке може бути даними або функцією. Об'єкти використовуються для зберігання складних структур даних, представлення сутностей з властивостями та методами та для організації коду у вигляді модулів і просторів імен.

**Створення об'єктів:**

Об'єктний літерал - найпростіший і найбільш поширений спосіб створення об'єктів.

var person = {

name: 'Іван',

age: 30

};

Конструктор new Object() - створення об'єкта за допомогою вбудованого конструктора Object.

var person = new Object();

person.name = 'Іван';

person.age = 30;

Object.create(proto) - створює новий об'єкт з вказаним прототипом і властивостями (прототипи детально розглядаються в наступних темах).

var proto = {

greet: function() {

console.log('Привіт, ' + this.name);

}

};

var person = Object.create(proto);

person.name = 'Іван';

person.greet(); // Виведе: Привіт, Іван

**Робота з об'єктами:**

delete - видаляє властивість з об'єкта.

delete person.age;

console.log(person.age); // Виведе: undefined

in - перевіряє, чи існує властивість у об'єкта.

console.log('name' in person); // Виведе: true

Object.freeze() - забороняє зміни в об'єкті, роблячи його незмінним.

Object.freeze(person);

person.name = 'Петро'; // Зміни не будуть застосовані

console.log(person.name); // Виведе: Іван

Object.assign() - копіює значення всіх властивостей від одного або кількох джерел до цільового об'єкта.

var personCopy = Object.assign({}, person);

console.log(personCopy.name); // Виведе: Іван

**Глибоке копіювання об'єкта:**

Для глибокого копіювання об'єкта, який може містити вкладені об'єкти, використовуються спеціалізовані функції або бібліотеки, оскільки Object.assign() виконує лише поверхневе копіювання.

var deepCopy = JSON.parse(JSON.stringify(person));

deepCopy.name = 'Микола';

console.log(person.name); // Виведе: Іван

console.log(deepCopy.name); // Виведе: Микола

Об'єкти є основою для багатьох аспектів програмування у JavaScript, включаючи створення складних структур даних, управління станом і організацію коду.