**JavaScript** – є синхронною і однопотоковою мовою програмування високого рівня, реалізованою згідно стандарту ECMA Script.

**Стандарт ECMA Script (ES)** – це специфікація, яка визначає синтаксис, типи даних, оператори, об'єкти та інші елементи JavaScript.

**Мова JavaScript використовується для:**

* для розробки клієнтської частини веб-додатків, що працюють у браузері.
* створення односторінкових та прогресивних вебзастосунків (React, AngularJS, Vue.js)
* програмування на боці сервера (Node.js(Express.js ))
* стаціонарних застосунків (Electron, NW.js)
* мобільних застосунків (React Native, Cordova)
* сценаріїв в прикладних програмах (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter )
* всередині PDF-документів тощо.

**Основні особливості JavaScript:**

* Динамічна типізація: Змінні в JavaScript не мають статичного типу даних і можуть змінювати свій тип під час виконання.
* Об'єктно-орієнтоване програмування: JavaScript підтримує об'єктно-орієнтовану парадигму, що дозволяє створювати та використовувати об'єкти, класи, успадкування та поліморфізм.
* Функціональне програмування: JavaScript також підтримує функціональне програмування, дозволяючи використовувати функції як значення, передавати їх як аргументи, повертати функції з інших функцій тощо.
* Маніпуляція DOM: JavaScript надає доступ до об'єктної моделі документа (DOM), що дозволяє змінювати структуру та вміст HTML-елементів на веб-сторінці.
* Асинхронне програмування: JavaScript має механізми асинхронного програмування, такі як колбеки, проміси та асинхронні функції, для ефективної обробки операцій.

**Основні складові частини виконання програми JavaScript**

Середовище у якому браузер виконує програми JavaScript - **Execution context** (контекст виконання).

Компоненти контексту виконання:

1. Компонент пам’яті - **Memory** (колекція пар ключ-значення : назва\_змінної : значення\_змінної).
2. Компонент коду - **Thread of Execution** (місце де виконуються команди).

|  |  |
| --- | --- |
| **Memory** | **Thread of Execution** |
| https://lh3.googleusercontent.com/WKNDus_TkwsP9G9FyJBVSG8PNV2ZVDXcyMMOczs3Kg6NNfFcS1ch8QgbyORxctz1BKVpsX09oBhws1JH-ybiDBNJuLIVp6GyLACJf5ENzssrYFMuwTPEfxDVsVVaNMCEFiOuIZdoPoz0b5eq7plc_w=s2048 | https://lh4.googleusercontent.com/eoxLV1YgMSoDn9whMnvCk4jLW_IfSdBrArKB6ltw0IIzMAPAWOW4937bzFx_T4X4sE0uXXou2br-gHfJ64nkbBzW_pTT6M2Ajpv4EhvkaVKfb3dEPwHmsOAsmsViGq_ixyMTLiwiT1BAKVCGaY-W6Q=s2048 |

**Фази виконання програм JavaScript:**

1. Фаза виділення пам’яті (виділення пам’яті для змінних і функцій).
2. Фаза виконання команд (поступове виконання команд по одній за раз).

|  |  |
| --- | --- |
| **Фаза виконання** | **Стан контексту виконання** |
| 1) Виділення пам’яті (змінні мають початкове значення - **undefined**) | |  |  | | --- | --- | | **Memory** | **Thread of Execution** | | https://lh4.googleusercontent.com/JXWDZ-35wAk6lkbEQyYxacyypZQwUdu4RddU6WkqM6M70ecHl54j9vDR4K3TvqASGzZpn4q8_5SXQNAjjQ5Gy9vSjDrYtV7gtUHm2QmUXw98iH98b_-mGW-L8amFHY5FlMX3EeiW-NoeG-NoeohAvw=s2048 |  | |
| 2) Виконання команд | |  |  | | --- | --- | | **Memory** | **Thread of Execution** | | https://lh3.googleusercontent.com/nwHMoNS6d7TVZyG2g7jXxxha8PnyqqK7Zo4ryoO6l9bawpEI4mdDT8qNi5lKJ6IauFMtuzuxnGzrvY15wPzBPWrWhc4mLXpzaz1vPlXD4bNMcErqg6bCOm81TBKyRXlCZlmPMwYbI3Qd8vDisX0S4A=s2048 | https://lh5.googleusercontent.com/5_ag_Ct0rryZDdgp3HmWWsnMQWrATDvi78YQrV_irCxbmZUSCk_gPxZTF2azGkuOH3at_pT9v_UeI30gUnzNC2LwB5SusxCby2lEoroBF8dF4oO15gU2yArO3Aq9mi5FgXBeMO0UMpYJZfoqeyR8Ag=s2048 | |

**Підключення (додавання) скриптів:**

**1) Безпосереднє включення у текст сторінки.**



**2) Підключення окремих файлів з скриптами (мають розширення «js»).**



**Підключення та порядок виконання скриптів:**

1. **Блокуючий обробку сторінки.**

|  |  |
| --- | --- |
| <script src=”script1.js”>  <script src=”script2.js”> | Сторінка **html** заванатжується. Як тільки черга доходить до  **<**script src=**”script1.js”>**, завантаження **html** призупиняється, браузер починає завантажувати і відразу здійснювати файл **”script1.js”,** потім завантажує і відразу здійснює **”script2.js”**, і тільки після цього браузер продовжить дозавантажувати **html**. |

1. **Неблокуючий обробку сторінки.**

|  |  |  |
| --- | --- | --- |
| <script **async** src=”script1.js”>  <script **async** src=”script2.js”> | Сторінка **html** заванатжується. Як тільки черга доходить до  **<**script **async** src=”script1.js”**>** він асинхронно завантажується, разом з завантаженням HTML-сторінки. Скрипт почне виконуватись відразу, незалежно від стану обробки сторінки або інших скриптів.  Скрипти виконуються у будь якому порядку (хто встиг швидше завантажитись), незалежно від їх розташування у структурі html. | |
| <script **defer** src=”script1.js”>  <script **defer** src=”script2.js”> | | Сторінка **html** заванатжується. Як тільки черга доходить до  **<**script **defer** src=”script1.js”**>** він асинхронно завантажується, разом з завантаженням HTML-сторінки. Скрипт почне виконуватись тільки після завершення завантаження HTML-сторінки.  Скрипти будуть виконуватись почерзі, згідно їх розташування у структурі html. |
| <script **defer** src=”script1.js”>  <script **async** src=”script2.js”> | | Спочатку завантажиться і виконається <script **async** src=”script2.js”>  Потім завантажиться <script **defer** src=”script2.js”>  Потім завантажиться HTML-сторінка  Потім виконається <script **defer** src=”script2.js”> |

**JavaScript має декілька типів даних:**

* Рядки (**Strings**): Використовуються для збереження текстової інформації (символи, слова, речення). Наприклад: "Привіт, світ!".
* Числа (**Numbers**): Використовуються для збереження числових значень як цілик так і дробових. Наприклад: 42, 3.14.
* Булеві значення (**Boolean**): Мають лише два можливих значення: true (істина) і false (хиба). Використовуються для логічних операцій та умовних перевірок.
* Об'єкти (**Objects**): Використовуються для збереження складних структур даних, які містять властивості і методи. Об'єкти створюються за допомогою фігурних дужок {}.

Наприклад: {name: "John", age: 30}.

* Масиви (**Arrays**): Використовуються для збереження колекцій значень впорядкованого списку. Масиви створюються за допомогою квадратних дужок []. Наприклад: [1, 2, 3, 4].
* **Null**: Використовується для вказання відсутності значення. Він є окремим типом даних і має єдине значення - null. Присвоюється змінній, коли потрібно показати, що вона не має значення або потребує очищення.
* **Undefined**: один з примітивних типів даних, який вказує на відсутність присвоєного значення змінній або властивості об'єкта.
* **Symbol**: Це новий тип даних, введений у стандарті ECMAScript 6. Використовується для створення унікальних ідентифікаторів. Начало формы

**Оператор "typeof"** використовується для визначення типу даних змінної, значення або виразу в JavaScript. Він дозволяє дізнатись, до якого саме типу даних належить певний об'єкт чи значення.

Оператор "typeof" повертає “рядок”, який відповідає типу даних. Наприклад, він може повернути "**number**" для числа, "**string**" для рядка, "**boolean**" для булевого значення, "**object**" для об'єкта, "**function**" для функції та так далі.

let x = 42;

console.log(typeof x); // Виведе "number"

let message = "Привіт, світ!";

console.log(typeof message); // Виведе "string"

let isActive = true;

console.log(typeof isActive); // Виведе "Boolean”

**Літерал** - це спосіб представлення фіксованого значення в коді. Літерали використовуються для створення конкретних значень для різних типів даних, таких як числа, рядки, булеві значення, масиви, об'єкти та інші.

* Літерал числа ( цілого або дійсного типу ): 42, 3.14, -10.
* Літерал рядка ( символьний ): "Привіт, світ!", 'JavaScript', `Рядок зі змінною ${variable}` (шаблонний рядок).
* Літерал булевого значення ( логічний ): true, false.
* Літерал масиву: [1, 2, 3], ['яблуко', 'банан', 'помаранч'].
* Літерал об'єкта: { name: 'John', age: 30 }.
* Літерал регулярного виразу: /pattern/, /[A-Za-z]+/.

**Спецсимволи JS**

**\b**: Використовується для представлення символу зворотнього виведення (backspace).

**\f**: Використовується для представлення символу переходу на нову сторінку (form feed).

**\r**: Використовується для представлення символу повернення каретки (carriage return).

**\n**: Використовується для представлення символу нового рядка (line feed).

**\v**: Використовується для представлення символу вертикальної табуляції (vertical tab).

**\t**: Використовується для представлення символу горизонтальної табуляції (horizontal tab).

**\'**: Використовується для екранування символу одинарної лапки всередині рядка, який вже містить одинарні лапки.

**\"**: Використовується для екранування символу подвійної лапки всередині рядка, який вже містить подвійні лапки.

**\XXX**: Восьмеричний спецсимвол. Використовується для представлення символу з восьмеричним кодом XXX.

**\xXX**: Шістнадцятковий спецсимвол. Використовується для представлення символу з шістнадцятковим кодом XX.

**\uXXXX**: Юнікод-символ. Використовується для представлення символу Юнікоду з кодом XXXX, де XXXX - код символу в шістнадцятковому форматі.

**\u{XXXXX}**: Розширений юнікод-символ. Використовується для представлення символу Юнікоду з розширеним кодом XXXXX, де XXXXX - код символу в шістнадцятковому форматі.

**Опис змінних**

У JavaScript **змінна** - це іменоване місце для зберігання даних. Змінні використовуються для зберігання та маніпулювання значеннями певних типів даних, таких як числа, рядки, булеві значення, об'єкти тощо.

**Оголошення змінної в JavaScript:**

var myVariable;

myVariable це ім'ям змінної. За допомогою оператора присвоєння (=) присвоїмо їй значення:

myVariable = 10;

Можна оголошувати змінні і присвоювати їм значення одночасно:

var myVariable = "Hello";

JavaScript є динамічно типізованою мовою, що означає, що тип даних змінної може змінюватися під час виконання програми. Ми також можемо змінювати значення змінної після її оголошення:

var myVariable = 5;

myVariable = "World";

Змінні в JavaScript можуть мати різні **області видимості**, такі як **глобальна** (видима в усій програмі) або **локальна** (видима лише у межах певної функції чи блоку).

**Функціональна область видимості**: Змінні, оголошені всередині функції (включаючи параметри функції), мають функціональну область видимості. Це означає, що ці змінні будуть доступні всередині цієї функції, а також всередині будь-яких вкладених функцій (якщо такі є). Змінні з функціональною областю видимості не будуть доступні поза функцією, в якій вони були оголошені.

**Блочна область видимості:** Змінні, оголошені всередині блока коду (наприклад, умовного оператора if, циклу for, блока {}), мають блочну область видимості. Це означає, що ці змінні будуть доступні лише всередині цього блока та будь-яких вкладених блоків. Вони не будуть доступні поза цим блоком. Це правило стосується змінних, оголошених з використанням let або const. Змінні, оголошені з використанням let, можуть змінювати свої значення, тоді як змінні, оголошені з використанням const, є незмінними (константами) і не можуть змінювати свої значення.

**function** exampleFunction() {

**var** firstVariable = "First functional scope"; // Змінна **з функціональною** областю видимості

**if** (true) {

**var** secondVariable = "Second functional scope"; // Змінна **з функціональною** областю видимості

**let** minLenght = " minLenght - have block scope"; // Змінна **з блочною** областю видимості

**const** MAX\_LENGHT = "MAX\_LENGHT – constant with block scope"; // Константа **з блочною** областю видимості

console.log(firstVariable); // + Виведе "First functional scope"

console.log(secondVariable); // + Виведе " Second functional scope "

console.log(minLenght); // + Виведе "minLenght - have block scope "

console.log(MAX\_LENGHT); // + Виведе MAX\_LENGHT – constant with block scope "

}

console.log(firstVariable); // + Виведе "First functional scope "

console.log(secondVariable); // + Виведе " Second functional scope "

console.log(minLenght); // - Помилка: minLenght не доступна за межами блока

console.log(MAX\_LENGHT); // - Помилка: MAX\_LENGHT не доступна за межами блока

}

console.log(firstVariable); // - Помилка: varVariable не доступна за межами функції

console.log(secondVariable); // - Помилка: varVariable не доступна за межами функції

**Способи виведення даних:**

* **Використання функції console.log()**: Цей метод дозволяє виводити дані у консоль браузера або середовища виконання JavaScript.

let a = 9;

let b = 7;

let sum = a + b;

**console.log**("Sum =”+sum);

* **Використання alert()**: Ця функція виводить повідомлення в модальному вікні браузера.

let a = 9;

let b = 7;

let sum = a + b;

**alert**("Sum =”+sum);

За допомогою цього діалогового вікна можна виводити не тільки текстові дані, а й дані інших типів. При цьому параметр (якщо він не текстового типу) приводиться до текстового типу.

* **Використання** документа для виведення даних: Ви можете використовувати властивості **document.write()** або **document.writeln()** для виведення даних прямо на HTML сторінці.   
  При цьому можна також викорстовувати розмітку HTML, наприклад тег <br>.

let a = 9;

let b = 7;

let sum = a + b;

**document.write**("Sum =”+sum);

**Способи введення даних (значення змінних):**

**prompt** - вспливаюче вікно з повідомленням для введення даних. Має текстове повідомлення і місце для введення даних. Функція prompt приймає два параметри: **message** і **default**.

* **message** - це текстове повідомлення, яке відображається користувачу.
* **default** - це необов'язкове значення, яке встановлюється в поле введення за замовчуванням.

Коли функція prompt викликається, воно блокує виконання коду, доки користувач не введе дані та не натисне кнопку "OK" або "Скасувати". Після цього функція повертає введене користувачем значення як рядок.

let name = prompt("message - Введіть ваше ім'я:", "default - Анонім");

console.log("Привіт, " + name + "!");

let age = prompt("Введіть ваш вік:");

console.log("Вам " + age + " років.");

У першому прикладі, користувачу буде показано спливаюче вікно з повідомленням "Введіть ваше ім'я:" та полем для введення даних. За замовчуванням у полі введення встановлено значення "Анонім". Після введення значення та натискання кнопки "OK", введене значення буде збережено у змінну name, і результат буде виведений у консолі.

У другому прикладі, користувачу буде показано спливаюче вікно з повідомленням "Введіть ваш вік:" та полем для введення даних. Користувач може ввести свій вік, а введене значення буде збережено у змінну age, і результат буде виведений у консолі.

Зауважте, що значення, отримане з функції prompt, завжди є рядком. Якщо вам потрібно використовувати це значення як число, ви можете скористатися функціями, такими як **parseInt** або **parseFloat**, для його перетворення.

**parseInt –** функція використовується для перетворення рядка (string) на **ціле число** (**integer**).

parseInt(string, radix)

* **string:** Рядок, який потрібно перетворити на ціле число.
* **radix** (необов'язковий параметр): Вказує у якій системі числення заданий рядок.

let num1 = parseInt("42");

console.log(num1); // Виведе “42”

let num2 = parseInt("10", 2);

console.log(num2); // Виведе “2” (перетворення з двійкової системи числення)

let num3 = parseInt("FF", 16);

console.log(num3); // Виведе “255” (перетворення з шістнадцяткової системи числення)

Якщо рядок не містить допустимого числа, функція parseInt поверне значення **NaN (Not a Number)**.

let num4 = parseInt("Hello");

console.log(num4); // Виведе “NaN”

Важливо зазначити, що функція parseInt також відкидає десяткову частину числа, якщо вона присутня.

let num5 = parseInt("3.14");

console.log(num5); // Виведе “3”

**parsFloat –** функція яка використовується для перетворення рядка (string) на число з плаваючою точкою **(floating-point number).**

Вона приймає один параметр - рядок, який потрібно перетворити.

let num1 = parseFloat("3.14");

console.log(num1); // Виведе “3.14”

let num2 = parseFloat("10.5");

console.log(num2); // Виведе “10.5”

let num3 = parseFloat("2.71828");

console.log(num3); // Виведе “2.71828”

У всіх трьох прикладах, рядки з числами з плаваючою точкою перетворюються на відповідні числа з плаваючою точкою.

Важливо зазначити, що якщо рядок не містить дійсного числа, то parseFloat поверне значення NaN (Not a Number). Наприклад:

let num4 = parseFloat("Hello");

console.log(num4); // Виведе “NaN”

Також варто знати, що parseFloat перетворює рядок до першого недопустимого символу або до кінця рядка, якщо немає допустимих символів для числа з плаваючою точкою. Наприклад:

let num5 = parseFloat("3.14abc");

console.log(num5); // Виведе “3.14”

**toFixed() –** є методом, який доступний для числових значень. Він використовується для форматування числа з плаваючою точкою та задання кількості десяткових знаків після коми.

number.toFixed(digits)

Тут **number** є числовим значенням, до якого застосовується метод **toFixed()**, а **digits** - необов'язковим параметром, що вказує кількість десяткових знаків, які мають бути виведені після коми. Він може бути цілим числом від 0 до 20.

let number = 3.14149;

let newNumber = number.toFixed(3);

console.log(newNumber); // Виведе строку з значенням "3.141"

У цьому прикладі число 3.14149 було відформатовано за допомогою toFixed(**digist**=3), що означає, що вивід буде мати три десяткових знаки. Результатом є рядок "3.141".

Важливо, **toFixed() повертає рядок, а не число**. Тому, якщо потрібно виконати подальші математичні операції з результатом, може знадобитись перетворити його на число, наприклад, за допомогою parseFloat() або Number().

let realNumber = parseFloat(newNumber);

console.log(realNumber); // Виведе 3.14

**Арифметичний вираз** - це вираз, що складається з операторів та операндів, в якому виконуються арифметичні операції.

**Бінарні операції** - це операції, які виконуються між двома операндами. Термін "бінарний" походить від латинського слова "binarius", що означає "парний" або "двоїстий", оскільки такі операції потребують двох операндів.

|  |  |  |  |
| --- | --- | --- | --- |
| Оператор |  | Опис | Приклад |
| + | 12 | Додавання | 5 + 3 |
| - | 12 | Віднімання | 8 - 2 |
| \* | 13 | Множення | 4 \* 6 |
| / | 13 | Ділення | 10 / 2 |
| % | 13 | Остача від ділення (модуль) | 7 % 3 |
| > | 10 | Більше ніж | 5 > 2 |
| < | 10 | Менше ніж | 3 < 7 |
| >= | 10 | Більше або дорівнює | 4 >= 4 |
| <= | 10 | Менше або дорівнює | 6 <= 9 |
| === | 9 | Рівне | 2 === 2 |
| !== | 9 | Нерівне | 5 !== 3 |
| && | 5 | Логічне І | true && false |
| || | 4 | Логічне АБО | true || false |

**Унарні оператори** - це оператори, які працюють з одним операндом, тобто вони виконують дію лише на одному значенні. Унарні оператори використовуються для здійснення різних операцій, таких як зміна знаку числа, інкремент або декремент числа, логічне заперечення та інші.

Унарні оператори можуть бути застосовані до чисел, рядків, змінних, об'єктів та інших типів даних, залежно від їх призначення.

| **Оператор** | **Опис** |
| --- | --- |
| + | Унарний плюс. Перетворює операнд на число. |
| - | Унарний мінус. Змінює знак числа. |
| ++ | Унарний оператор інкремента. Збільшує значення операнду на одиницю. |
| -- | Унарний оператор декремента. Зменшує значення операнду на одиницю. |
| ! | Логічне заперечення. Перетворює операнд на логічне значення та повертає його заперечення. |
| typeof | Повертає рядкове представлення типу операнду. |
| delete | Видаляє властивість об'єкту або елемент масиву. |
| void | Повертає undefined. |
| ~ | Побітове заперечення. Перетворює операнд на 32-бітне ціле число та повертає його заперечення. |



**Шаблонні рядки**, також відомі як **template strings**, є особливим типом рядків, які дозволяють вставляти змінні або вирази у середину рядка без необхідності використання конкатенації або екранування символів.

У шаблонних рядках використовується спеціальний синтаксис зворотніх апострофів (backticks), які огортають рядок.

const a = 5;

const b = 10;

const result = `The sum of ${a} and ${b} is ${a + b}.`;

console.log(result);

У цьому прикладі змінні **a** і **b** вставляються у шаблонний рядок за допомогою синтаксису ${...}. Під час виконання коду, значення змінних будуть автоматично підставлені відповідним чином. Результат виведення цього коду буде:

The sum of 5 and 10 is 15.

**Основні методи Math**

|  |  |
| --- | --- |
| Метод | Опис |
| Math.abs(x) | Повертає абсолютне значення числа **x**. |
| Math.ceil(x) | Округлює число **x** вгору до найближчого більшого цілого числа. |
| Math.floor(x) | Округлює число **x** вниз до найближчого меншого цілого числа. |
| Math.round(x) | Округлює число **x** до найближчого цілого числа (за правилами округлення). |
| Math.max(x1, x2, ...) | Повертає найбільше число серед **x1**, **x2**, ... |
| Math.min(x1, x2, ...) | Повертає найменше число серед **x1**, **x2**, ... |
| Math.pow(x, y) | Підносить число **x** до степеня **y**. |
| Math.sqrt(x) | Повертає квадратний корінь числа **x**. |
| Math.random() | Генерує випадкове дробове число в діапазоні від 0 (включно) до 1 (не включаючи). |
| Math.sin(x) | Повертає синус числа **x** (в радіанах). |
| Math.cos(x) | Повертає косинус числа **x** (в радіанах). |
| Math.tan(x) | Повертає тангенс числа **x** (в радіанах). |
| Math.log(x) | Повертає натуральний логарифм числа **x**. |
| Math.exp(x) | Повертає експоненту (число e, приблизно 2.71828) піднесену до степеня **x**. |
| Math.atan(x) | Повертає арктангенс числа **x**. |
| Math.acos(x) | Повертає арккосинус числа **x**. |
| Math.asin(x) | Повертає арксинус числа **x**. |
| Math.trunc(x) | Видаляє десяткову частину числа **x**, повертаючи тільки цілу частину без округлення. |
| Math.sign(x) | Повертає знак числа **x** як -1, 0 або 1, вказуючи на від'ємне, нульове або додатнє значення відповідно. |
| Math.log10(x) | Повертає логарифм числа **x** за основою 10. |
| Math.log2(x) | Повертає логарифм числа **x** за основою 2. |
| Math.cbrt(x) | Повертає кубічний корінь числа **x**. |
| Math.hypot(x1, x2, ...) | Повертає гіпотенузу для заданих сторін прямокутного трикутника зі сторонами **x1**, **x2**, ... (квадратний корінь суми квадратів). |

**Перетворення** різних типів даних **на рядок (стрінгу)** за допомогою різних методів та операцій

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Тип даних | toString() | toFixed() | toPrecision() | toLocaleString() | JSON.stringify() | String() | Конкатенація |
| Number | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ |
| Boolean | ✔️ |  |  |  | ✔️ | ✔️ | ✔️ |
| String | ✔️ |  |  | ✔️ | ✔️ | ✔️ | ✔️ |
| Array | ✔️ |  |  |  | ✔️ | ✔️ | ✔️ |
| Object | ✔️ |  |  |  | ✔️ | ✔️ | ✔️ |

**Конкатенація** - процес об'єднання двох або більше рядків в один (або одного рядка та іншого типу даних).

* **Використання оператора "+"**

var str1 = 'Hello';

var str2 = ' world!';

var result = str1 **+** str2; // Конкатенація рядків

console.log(result); // Виведе "Hello world!"

* **Використання методу concat()**

var str1 = 'Hello';

var str2 = ' world!';

var result = str1**.concat(str2)**; // Конкатенація рядків

console.log(result); // Виведе "Hello world!"

* **Використання шаблонних рядків** (Template literals)

var name = "John";

var age = 30;

var str = `My name is **${name}** and I am **${age}** years old.`;

console.log(str); // Виведе "My name is John and I am 30 years old."

* **Метод join():** Цей метод масиву об'єднує всі елементи масиву в один рядок, розділених заданим роздільником.

var arr = [1, 2, 3];

var str = arr**.join("-")**;

console.log(str); // Виведе "1-2-3"

* **Метод toString()**

var num = 42;

var str = num**.toString()**;

console.log(str); // Виведе "42"

var bool = true;

var str = bool**.toString()**;

console.log(str); // Виведе "true"

**Перетворення** різних типів даних **на число**

|  |  |  |
| --- | --- | --- |
| Тип даних | Приклад значення | Результат |
| Рядок | "42" | 42 |
| Рядок | "3.14" | 3.14 |
| Рядок | порожній рядок | 0 |
| Рядок | "ABC" | NaN |
| Булеве значення | true | 1 |
| Булеве значення | false | 0 |
| null | null | 0 |
| undefined | undefined | NaN |
| Об'єкт | { value: 42 } | NaN |
| Масив | [1, 2, 3] | NaN |

**Логічний тип**

Логічний тип даних **Boolean** має всього два значення:

* **true** («істина», «вірно» або ж «так»)
* **false** («неправда», «невірно» або ж «ні»)

**Перетворення до** логічного типу даних **(Boolean)**

Перетворення до логічного типу (Boolean) виконується для визначення, чи є значення істинним (true) або неправдивим (false)

|  |  |
| --- | --- |
| Значення | Результат |
| false | false |
| Число 0 | false |
| Порожній рядок " " | false |
| null | false |
| undefined | false |
| NaN | false |
| -------------------------------------------------------------------------------------------------------------------- |  |
| true | true |
| Будь-яке число, відмінне від 0 (включаючи від'ємні і десяткові числа) -1, 1, 13,5 | true |
| Непорожній рядок “Hello” | true |
| Об'єкт {} | true |
| Масив [] | true |
| Будь-яка функція function(){} | true |

Перетворення до логічного типу може бути здійснене за допомогою функції ‘Boolean()’ або використанням логічного оператора ’!!’

**Логічні вирази**

Логічний вираз в JavaScript є виразом, який містить **логічні оператори**, **оператори порівняння** та операнди і видає результат у вигляді логічного значення (true або false).

**Логічні вирази можуть містити:**

* **логічні константи (true/false**) - це фіксовані значення, які представляють логічний стан істинності або хибності.

var x = 5;

if (x > 0) {

console.log("x є додатним числом"); // true

} else {

console.log("x не є додатним числом"); // false

}

* **змінні логічного типу**

let isTrue = true;

let isFalse = false;

console.log(isTrue); // true

console.log(isFalse); // false

* **оператори порівняння -** використовуються для порівняння значень і повертають логічне значення true або false залежно від результату порівняння.

var a = 5;

var b = 10;

|  |  |  |  |
| --- | --- | --- | --- |
| console.log(a == b); | 9 | false | Рівність |
| console.log(a === b); | 9 | false | Строга рівність |
| console.log(a != b); | 9 | true | Нерівність |
| console.log(a !== b); | 9 | true | Строга нерівність |
| console.log(a > b); | 10 | false | Більше |
| console.log(a < b); | 10 | true | Менше |
| console.log(a >= b); | 10 | false | Більше або рівне |
| console.log(a <= b); | 10 | true | Менше або рівне |

**== (рівність)**: Порівнює два значення на рівність, перетворюючи типи даних, якщо необхідно. Наприклад - 1 == '1' буде true, оскільки значення рівні після перетворення типів.

**=== (строга рівність)**: Порівнює два значення на рівність, без перетворення типів. Воно порівнює значення і тип даних.

Наприклад - 1 === '1' буде false, оскільки значення рівні, але типи даних різні.

**!= (нерівність)**: Порівнює два значення на нерівність, перетворюючи типи даних, якщо необхідно. Наприклад - 1 != '1' буде false, оскільки значення рівні після перетворення типів, тому вони не є нерівними.

**!== (строга нерівність)**: Порівнює два значення на нерівність, без перетворення типів. Воно порівнює значення і тип даних.

Наприклад - 1 !== '1' буде true, оскільки значення рівні, але типи даних різні, тому вони є строго нерівними.

Строгі оператори рівності **===** та **!==**  часто вважаються безпечнішими, оскільки вони не здійснюють автоматичне перетворення типів, дозволяючи точнішу перевірку значень і типів даних

**Операції з логічними виразами**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y | x || y 4 | x && y 5 | ^ 7 | !x 15 |
| false | false | false | false | false | true |
| false | true | true | false | true | true |
| true | false | true | false | true | false |
| true | true | true | true | false | false |

* Логічне і (&&): Повертає true, якщо обидва операнди є true, інакше повертає false.

var x = 5;

var y = 10;

var z = 15;

console.log(x < y && y < z); // Виведе true - оскільки обидва умови виконуються

console.log(x < y && y > z); // Виведе false - оскільки друга умова не виконується

* Логічне АБО (||): Повертає true, якщо хоча б один з операндів є true, інакше повертає false.

var x = 5;

var y = 10;

var z = 15;

console.log(x < y || y < z); // Виведе true - оскільки хоча б одна з умов виконується

console.log(x > y || y > z); // Виведе false - оскільки обидві умови не виконуються

* Логічне НЕ (!): Інвертує значення операнду (true перетворюється на false і навпаки).

var x = 5;

var y = 10;

console.log(!(x < y)); // Виведе false - оскільки x < y є true, а !true = false

console.log(!(x > y)); // Виведе true - оскільки x > y є false, а !false = true

**Умовний оператор if-else** – керуюча конструкція в JS

var age = 20;

if (age >= 18) {

console.log("Ви повнолітній");

} else {

console.log("Ви неповнолітній");

}

Основна ідея умовного оператора полягає в перевірці заданої умови. Якщо умова є істинною, виконується блок коду після ключового слова if. У протилежному випадку, якщо умова є хибною, виконується блок коду після ключового слова else.

**Тернарний оператор** – керуюча конструкція в JS

var age = 20;

var message = **(**age >= 18**) ? "**Ви повнолітній**" : "**Ви неповнолітній**"**;

console.log(message); // Виведе “Ви повнолітній”

У цьому прикладі, змінна age містить значення 20. Умова (age >= 18) перевіряє, чи age більше або дорівнює 18. Якщо це вірно, то змінна message отримує значення "Ви повнолітній". У протилежному випадку, коли умова є хибною, message отримує значення "Ви неповнолітній".

**Оператор вибору** **switch** – керуюча конструкція в JS

Оператор вибору перевіряє значення виразу та порівнює його зі значеннями у різних випадках (**case**). Якщо знайдено відповідну відповідність, виконується код, пов'язаний з цим випадком. Якщо жоден з випадків не збігається, виконується блок коду, що розташований після **default**

var medal = 2;

var medalName;

switch (medal) {

case 1:

medalName; = "Золота";

break;

case 2:

medalName; = "Срібна";

break;

case 3:

medalName; = "Бронзова";

break;

default:

medalName; = "Хибні дані";

break;

}

console.log("Медаль " + medalName);

**Короткий цикл обчислень**

Короткий цикл обчислень дозволяє програмі пропустити зайві оцінки умов і покращити продуктивність, вибираючи перше "істинне" значення або зупиняючись на першій "істинній" умові.

**Логічний оператор** "АБО" **|| 4**

У цьому рішенні, використовуючи логічний оператор "АБО" (||), ми перевіряємо кожну змінну від a до d на "істинність" значення (**true**)

var a = ‘’;

var b = null;

var c = 'World';

var d = undefined;

var result = a || b || c || d || 'No value found';

console.log(result); // 'World'

Результатом обчислення буде значення виразу, якому еквівалентне **true** (усі наступні вирази не обчислюються).

У цьому прикладі, перша умова **a** перевіряє, яким є значення **a** при перетворенні до логічного типу

даних Boolean - **true** або **false**.

Значення **a** =’’ - це порожній рядок – воно означає що **а** – **false**, умова не виконується .

Таким же чином перевіряється кожна наступна умова b || c || d. Як тільки якась умова матиме "істинне" значення **true**, змінній result присвоїться її значення.

Значення **b** = null (**false) –** умова теж не виконується.

Значення **с** = 'World' – непорожній рядок – означає що **с** – **true.**

Наступна умова **d** не оцінюється.

**Логічний оператор** "І" **(&&) 5**

У цьому рішенні, використовуючи логічний оператор "І" (&&), ми перевіряємо кожну змінну від a до d на "хибне" значення (**false**).

var a = true;

var b = 'World';

var c = undefined;

var d = 0;

var result = a && b && c && d;

console.log(result); // undefined

Результатом обчислення буде значення виразу, якому еквівалентне **false** (усі наступні вирази не обчислюються).

Значення **a** = true - це означає що **а** присвоєно значення – **true**, умова не виконується .

Значення **b** = 'World' – непорожній рядок – означає що **b** – **true.**

Значення **с** = undefined – означає що змінна **с** не має визначеного значення, отже– **false.**

Наступна умова **d** не оцінюється.

**Оператор нульового злиття** (??) **Nullish Coalescing Operator**. 4

У цьому рішенні, використовується оператор ?? (Nullish Coalescing Operator)

var a = null;

var b = undefined;

var c = 'Hello';

var d = 'World';

var result = a ?? b ?? c ?? d;

console.log(result); // 'Hello'

Результатом обчислення буде значення виразу, який не є **null** або **undefined** (усі інші вирази праворуч не обчислюються).

Змінній **result** присвоїться значення змінної c = **'Hello'**, так як це перша умова яка не є **null** або **undefined**.

**Оператор умовного присвоєння ??= 2**

Оператор умовного присвоєння **??=** може бути використана в короткому циклі обчислень для швидкої перевірки та присвоєння значень змінним, якщо вони є null або undefined.

var a = null;

var b = 'Hello';

var c = undefined;

var d = 'World';

a ??= 'Default Value';

b ??= 'Default Value';

c ??= 'Default Value';

d ??= 'Default Value';

console.log(a); // 'Default Value'

console.log(b); // 'Hello' (значення не змінилося)

console.log(c); // 'Default Value'

console.log(d); // 'World' (значення не змінилося)

**Цикли** - керуюча конструкція в JS

**Цикл** - конструкція мови, яка дозволяє виконувати певний блок коду або набір інструкцій повторно. Цикли дозволяють автоматизувати повторні завдання, виконувати обробку масивів, працювати з колекціями даних і багато іншого.

Цикл з параметром **for**

**for** - використовується для повторення блоку коду певну кількість разів або до досягнення певної умови.

for (ініціалізація; умова; ітерація) {

// блок коду, який виконується

}

**ініціалізація** - вказує початкове значення змінної, яка контролює цикл.

**умова** - це булевий вираз, який перевіряється перед кожною ітерацією. Якщо умова є true, то блок коду виконується. Якщо умова є false, то цикл завершується.

**ітерація** - визначає дії, які виконуються після кожної ітерації циклу. Зазвичай вона використовується для зміни значення змінної, яка контролює цикл.

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

console.log(i);

}

У цьому прикладі цикл **for** виконується п'ять разів. Змінна **i** ініціалізується значенням 0. Умова **i < 5** перевіряється перед кожною ітерацією, і якщо вона є **true**, то блок коду виконується. Після кожної ітерації змінна **i** збільшується на 1 за допомогою оператора **i++**. Результатом будуть числа від 0 до 4, виведені в консоль.

Цикл з передумовою **while**

**while** - Використовується для виконання блоку коду, доки задана умова є істинною. Умова перевіряється перед кожною ітерацією циклу.

while (умова) {

// блок коду, який виконується

}

**умова** - це булевий вираз, який перевіряється перед кожною ітерацією. Якщо умова є true, то блок коду виконується. Якщо умова є false, то виконання циклу припиняється, і керування передається наступній інструкції після циклу.

let i = 0;

while (i < 5) {

console.log(i);

i++;

}

У цьому прикладі блок коду виконується, доки змінна **i** менше 5. Змінна **i** початково має значення 0. Після кожної ітерації значення i збільшується на 1. Результатом будуть числа від 0 до 4, виведені в консоль.

Цикл з післяумовою **do...while**

**do...while** - тип циклу, який виконує блок коду перед перевіркою умови. Після виконання блоку коду умова перевіряється, і якщо вона є істинною, цикл продовжується.

do {

// блок коду, який виконується

} while (умова);

let i = 0;

do {

console.log(i);

i++;

} while (i < 5);

У цьому прикладі блок коду виконується спочатку, незалежно від умови. Потім, після кожної ітерації, перевіряється умова **i < 5**. Якщо умова є **true**, цикл продовжується і блок коду виконується знову. Цикл завершується, коли умова стає **false**. Результатом будуть числа від 0 до 4, виведені в консоль.

Оператор **break**

Використовується для негайного виходу з циклу або переривання виконання блоку коду. Коли виконується оператор **break**, керування передається наступній інструкції після циклу або блоку коду, що містить break.

Оператор **break** може бути використаний у циклах (**for, while, do...while**) та блоках коду, таких як **switch**.

let i = 0;

while (i < 5) {

console.log(i);

i++;

if (i === 3) {

break;

}

}

У цьому прикладі, коли i досягає значення 3, виконується оператор **break**, і цикл **while** припиняє свою роботу. Результатом будуть числа 0, 1 і 2, виведені в консоль.

Оператор **break** з міткою

Використовується для виходу з зовнішнього циклу або блока коду, коли є вкладені цикли або блоки коду.

out: for (let i = 0; i < 3; i++) {

for (let j = 0; j < 3; j++) {

if (i === 1 && j === 1) {

break out;

}

console.log(`i: ${i}, j: ${j}`);

}

}

У цьому прикладі ми маємо вкладений цикл **for**, який виконується для значень **i** від 0 до 2 і **j** від 0 до 2. Умова **if** перевіряє, чи **i** дорівнює 1 і **j** дорівнює 1. Якщо ця умова є істинною, то за допомогою оператора **break out** виконання циклу припиняється і вихід здійснюється зовнішньої мітки **out**. Це означає, що весь вкладений цикл припиняється і керування переходить на наступну інструкцію після зовнішнього циклу.

Оператор **continue**

При використанні оператора **continue** в коді, виконання поточної ітерації циклу припиняється, і керування переходить до наступної ітерації циклу.

let i = 0;

while (i < 5) {

i++;

if (i % 2 === 0) {

continue;

}

console.log(i);

}

У цьому прикладі, якщо **i** є парним числом, оператор **continue** виконується, і поточна ітерація циклу припиняється. Керування переходить до наступної ітерації циклу, і парне число пропускається. Результатом будуть непарні числа 1, 3 і 5, виведені в консоль.

**Функція**

**Функція** є блоком коду, який виконує певну задачу або обчислення. Вона складається з ключового слова **function**, **назви функції**, **списку параметрів** у круглих дужках, **тіла функції** і можливого **return** повернення значення.

* **Оголошення функції**: Використовується ключове слово **function**, за яким йде назва функції.

**function myFunction**(a, b) {

// код функції

}

* **Параметри**: Функції можуть мати вхідні параметри, які приймають значення, коли функція викликається. Вони вказуються у круглих дужках, розділені комами.

function myFunction**(a, b)** {

sum = a + b;

return sum;

}

* **Тіло функції**: Це блок коду, який виконується при виклику функції. Він містить інструкції та обчислення, які виконуються при виклику функції.

function myFunction (a, b) {

**sum = a + b**;

return sum;

}

* **Повернення значення**: Функція може повертати значення, за допомогою ключового слова return. Це значення може бути використане у виразах або збережене в змінну.

function myFunction (a, b) {

sum = a + b;

**return sum;**

}

Функції можуть бути визначені і викликані в будь-якому місці програми. Їх можна передавати як аргументи в інші функції і повертати як значення. Функції є потужним інструментом для структурування та повторного використання коду.