**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.
* **Undefined**: Використовується для вказання, що змінна не має значення або не була ініціалізована.
* **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). Крім var, в JavaScript також є ключові слова let і const, які використовуються для оголошення змінних з областю видимості блоку.

**var:** Змінні, оголошені з використанням var, мають функціональну область видимості. Це означає, що такі змінні будуть видимими всередині функції, в якій вони оголошені. Якщо змінна оголошена за межами будь-якої функції, вона матиме глобальну область видимості і буде доступною в усій програмі.

**let** та **const**: Змінні, оголошені з використанням let або const, мають блочну область видимості. Це означає, що такі змінні будуть видимими лише всередині блока, в якому вони оголошені. Блок може бути функцією, циклом, умовним оператором або будь-яким іншим блоком коду, обмеженим фігурними дужками {}. 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 varVariable = "I have a functional scope"; // Змінна з функціональною областю видимості

if (true) {

let letVariable = "I have a block scope"; // Змінна з блочною областю видимості

const constVariable = "I am a constant with block scope"; // Константа з блочною областю видимості

console.log(varVariable); // Виведе "I have a functional scope"

console.log(letVariable); // Виведе "I have a block scope"

console.log(constVariable); // Виведе "I am a constant with block scope"

}

console.log(varVariable); // Виведе "I have a functional scope"

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

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

}

exampleFunction();

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