◈ What is Programming and Why JavaScript?

◈ **Definition of Programming and Its Importance:**

Programming is the process of writing instructions that computers execute to solve problems or perform specific tasks.

◈ **Why JavaScript?**

- JavaScript is the **most popular** programming language for web development.
- It runs directly in the browser, making it **fast and efficient**.
- Works on both **frontend** (client-side) and **backend** (server-side with Node.js).
- Has a vast ecosystem of libraries and frameworks (**React, Vue, Angular**).
- Used in a wide range of applications beyond web development, including **AI, mobile apps, and IoT**.

◈ **Uses of JavaScript:**

- **Web Development**: Creating dynamic and interactive websites.
- **Mobile Apps**: Using frameworks like **React Native**.
- **Game Development**: Creating browser-based games.
- **AI and Machine Learning**: Using libraries like **TensorFlow.js**.
- **Server-side Development**: With **Node.js**.
- **Data Visualization**: Using tools like **D3.js**.

◈ **JavaScript in Artificial Intelligence (AI):**

- **Machine Learning in the Browser**: JavaScript enables ML models to run in the browser using **TensorFlow.js**.

- **Natural Language Processing (NLP)**: Used for **text analysis, sentiment detection, and chatbots**.

- **Computer Vision**: Detecting and recognizing images, faces, and objects using **tracking.js** and **TensorFlow.js**.

- **Voice and Speech Recognition**: Converting speech to text using **Web Speech API**.

- **Recommendation Systems**: Implementing personalized content suggestions, similar to Netflix and YouTube algorithms.

- **Automated Data Processing**: Using AI to classify, sort, and analyze data efficiently.

- **Chatbots and Virtual Assistants**: Developing smart assistants like **Google Assistant, Siri, or Alexa**, but inside web apps.

- **AI-powered Automation**: Creating **smart bots** that automate repetitive tasks in web applications.

🚀 JavaScript is a **versatile and powerful** language, making it a must-learn for developers, especially in AI-driven applications!

Here is a formatted summary explaining how to set up a complete JavaScript development environment on any device, with offline portability.

🚀 Setting Up a JavaScript Development Environment in Visual Studio Code

◈ Includes: VS Code + Node.js + Essential Extensions + Offline Portability

1⃣ Download and Install Essential Software

📌 Download and Install Visual Studio Code

🔗 Download Link: https://code.visualstudio.com/

📌 Manually: Download the appropriate version for your OS (Windows, macOS, Linux) and install it.

📌 Direct Command (Linux - Ubuntu/Debian):

```sh
sudo apt update && sudo apt install -y code
```

📌 Download and Install Node.js

🔗 Download Link: https://nodejs.org/

📌 It is recommended to install the (LTS) version for stability.

📌 Verify installation after setup:

```sh
node -v   # Show Node.js version
npm -v    # Show npm version
```

2 Install Essential Extensions in VS Code

📌 **These extensions will make JavaScript development easier and faster.**

◈ 1- Essential JavaScript Extensions

- ✅ **JavaScript (ES6) Code Snippets – Ready-to-use code snippets**
- ✅ **ESLint – Automatically analyze and fix code errors**
- ✅ **Prettier – Auto-format code**
- ✅ **Path Intellisense – Auto-complete file paths**

◈ 2- HTML, CSS & JavaScript Extensions

- ✅ **Live Server – Run a local server for live preview**
- ✅ **HTML CSS Support – Auto-completion for HTML & CSS**
- ✅ **Auto Rename Tag – Automatically update HTML tags**
- ✅ **CSS Peek – View linked CSS files directly**

◈ 3- React, Node.js, Debugging Extensions

- ✅ **React Snippets – Ready-made React code snippets**
- ✅ **Node.js Intellisense – Auto-complete for Node.js libraries**
- ✅ **Debugger for Chrome – Debug directly in the browser**

4- Productivity Boosting Extensions

- ✅ GitLens – View code change history
- ✅ Bracket Pair Colorizer – Color matching brackets
- ✅ Todo Tree – Track all TODOs in your project

📌 To install all these extensions at once, open the VS Code Terminal and run the following command:

```sh
code --install-extension xabikos.JavaScriptSnippets \
&& code --install-extension dbaeumer.vscode-eslint \
&& code --install-extension esbenp.prettier-vscode \
&& code --install-extension christian-kohler.path-intellisense \
&& code --install-extension ritwickdey.LiveServer \
&& code --install-extension ecmel.vscode-html-css \
&& code --install-extension formulahendry.auto-rename-tag \
&& code --install-extension pranaygp.vscode-css-peek \
&& code --install-extension dsznajder.es7-react-js-snippets \
&& code --install-extension leizongmin.node-module-intellisense \
&& code --install-extension msjsdiag.debugger-for-chrome \
&& code --install-extension eamodio.gitlens \
&& code --install-extension coenraads.bracket-pair-colorizer \
&& code --install-extension gruntfuggly.todo-tree
```

```

## 3⃣ Transfer Your Development Environment to Another Device Without Internet

### 📌 1- Manually Transfer Extensions

On the first device, copy the following folder to a USB or external drive:

- **Windows:** %USERPROFILE%\.vscode\extensions

- **Linux/macOS:** ~/.vscode/extensions

Then, on the new device, paste the folder in the same location, restart VS Code, and all extensions will work instantly.

### 📌 2- Transfer Node.js and Installed Libraries

If you use npm in your project, you can transfer installed packages by:

1. On the first device, navigate to the project folder and run:

```sh
npm list --depth=0 > packages.txt
```

2. On the new device, install the same packages using:

```sh
npm install
```

🎯 Final Outcome:
```

✅ **Install VS Code + Node.js + Essential Extensions.**

✅ **Set up a complete development environment for JavaScript, React, and Node.js.**

✅ **Easily transfer the entire setup to another device without internet.**

🔥 **With this method, you can start developing web applications with ease and efficiency!** 🚀

In the video ** Douglas Crockford: The JavaScript Programming Language, he discussed some **disadvantages of JavaScript**, including:**

Douglas Crockford: The JavaScript Programming Language – Key Features

In the video you watched, ** Douglas Crockford: The JavaScript Programming Language, several key features were highlighted that distinguish JavaScript from other languages:**

---

1. JavaScript is a Dynamic Language

◈ **No need to specify variable types when declaring them**, allowing flexibility in data types during execution.

✍ **Example:**

`javascript

let data = 5; // Number

data = Hello; // String

console.log(data); // Hello

`

💡 **Comparison with Java:**

In Java, you must declare the type of a variable (int number = 5;), making it less flexible.

---

2. Functional Programming Support

◈ JavaScript **strongly supports functional programming**, making code cleaner and more reusable.

◈ Functions are **first-class citizens**, meaning they can be stored in variables, passed as arguments, and returned from other functions.

✍ **Example:**

`javascript

function greet(name) {

  return Hello, ;

}

const sayHello = greet; // Storing function in a variable

console.log(sayHello(Kareem)); // Hello, Kareem

`

💡 **Comparison with C++ or Java:**

- In these languages, passing functions requires additional structures like **function pointers (C++)** or **functional interfaces (Java)**, making it more complex.

---

3. Object Handling via Prototypes (Prototype-Based Inheritance)

🔷 **Unlike Java and C++, JavaScript does not rely on classical inheritance (Classes and OOP). Instead, it uses Prototypes.**

✍ **Example:**

`javascript

const car = {

  brand: Toyota,

  start() {

    console.log(Car started);

  }

};

// CREATING A NEW OBJECT FROM CAR

CONST MYCAR = OBJECT.CREATE(CAR);

CONSOLE.LOG(MYCAR.BRAND); // TOYOTA

MYCAR.START(); // CAR STARTED

`

💡 **COMPARISON WITH JAVA:**

IN JAVA, YOU MUST DEFINE A **CLASS** AND USE EXTENDS FOR INHERITANCE, MAKING THE CODE MORE RIGID AND VERBOSE.

---

4. LOOSE TYPING FOR DATA HANDLING

◈ JAVASCRIPT **IS NOT STRICTLY TYPED**, ALLOWING AUTOMATIC TYPE CONVERSION.

✍ **EXAMPLE:**

`JAVASCRIPT

FUNCTION SUM(A, B) {

  RETURN A + B;

}

CONSOLE.LOG(SUM(5, 10)); // 15 (ADDITION)

CONSOLE.LOG(SUM(5, 10)); // 510 (STRING CONCATENATION)

`

💡 **Comparison with Java or C++:**

- In Java, **strict type checking** prevents such implicit conversions.

- In JavaScript, implicit type conversion makes it more flexible but can sometimes lead to unexpected behavior.

---

5. Event-Driven Programming

🔷 JavaScript is **designed to handle events efficiently**, making it perfect for UI development.

✍ **Example:**

`javascript

document.getElementById(myButton).addEventListener(click, function() {

  alert(Button clicked!);

});

`

💡 **Comparison with Java:**

- Java requires additional setup with Listener Interfaces, making event handling more complex.

- JavaScript provides a simple built-in ddEventListener method for direct event handling.

---

6. Asynchronous Programming Support

◈ JavaScript supports **asynchronous programming** via **Callbacks, Promises, and async/await**.

✍ **Example:**

`javascript

function fetchData() {

  return new Promise(resolve => {

    setTimeout(() => resolve(Data loaded!), 2000);

  });

}

fetchData().then(console.log); // After 2 seconds: Data loaded!

`

💡 **Comparison with C++ or Java:**

- Java uses **Threads**, which add complexity.

- JavaScript provides sync/await, making asynchronous code more readable and manageable.

---

7. No Compilation Required – Runs Directly in Browsers

◈ Unlike Java or C++, **JavaScript does not need compilation**. It runs directly in web browsers.

✍ **Example:**

`html

<script>

console.log(Hello JavaScript!);

</script>

`

💡 **Comparison with C++ or Java:**

- In **Java**, you must compile using javac MyProgram.java and then execute java MyProgram.

- In **C++**, you must compile with g++ myProgram.cpp -o myProgram and then run ./myProgram.

- In **JavaScript**, you can run code directly in the browser with no additional setup!

---

8. Cross-Platform Compatibility

◈ JavaScript runs on **any web browser**, eliminating the need for platform-specific configurations.

💡 **Example:**

- **Java requires JVM (Java Virtual Machine).**

- **C++ requires a Compiler (GCC, MSVC).**

- **JavaScript runs directly in browsers (Chrome, Firefox, Edge, etc.).**

◈ Summary of JavaScript's Advantages Over Other Languages

Feature	**JavaScript**	**Java / C++**
Dynamic Typing	Supports implicit type conversion	Requires explicit type declaration
Event Handling	Uses ddEventListener for fast interaction	Requires Listener Interfaces
Direct Execution	Runs in the browser instantly	Requires compilation & execution
Functional Programming	Supports first-class functions	Requires additional structures
Prototype Inheritance	Uses Prototypes for object creation	Uses Class-based inheritance
Asynchronous Support	Simple sync/await and Promises	Requires Threads for async execution

---

💡 When to Use JavaScript?

✅ When developing **web applications**.

✅ When creating **interactive websites**.

✅ When working with **APIs and asynchronous data**.

✅ When handling **UI interactions in web or mobile applications**.

**Do you have any questions or need further clarification?

1. JavaScript is a Weakly Typed Language

🔷 One of the biggest issues with JavaScript is **the lack of a strong type system**, leading to unexpected behavior.

✍ **Example of Automatic Type Conversion (Type Coercion):**

`javascript

console.log(5 + 5); // 55 (Adding a number to a string converts it to a string)

console.log(5 - 1); // 4 (The string is converted to a number in subtraction)

console.log(true + true); // 2 (Boolean values are converted to numbers)

`

💡 **The Problem?**

- This behavior can cause **unexpected errors**, especially when handling user input or API responses.

---

2. Lack of a Strong Module System in Older Versions

◈ Before **ES6 (ECMAScript 2015)**, JavaScript did not have a built-in **module system**, making code organization difficult.

✍ **Before ES6, we used Global Variables:**

`javascript

var myLibrary = {}; // A global library that any code can modify

myLibrary.sayHello = function() {

  console.log(Hello!);

};

`

💡 **The Problem?**

- This can lead to **name conflicts (Global Namespace Pollution)** when multiple libraries use the same names.

☑ **The Solution in ES6?** Use import/export to organize code.

`javascript

// myModule.js

export function sayHello() {

  console.log(Hello!);

}

// main.js

import { sayHello } from ./myModule.js;

sayHello(); // Hello!

`

---

3. Slower Performance Compared to Other Languages

🔷 Since JavaScript is an **interpreted (not compiled) language**, it is slower than languages like **C++ and Java** for complex computations.

✍ **Example of Speed Comparison Between JavaScript and C++:**

- In C++, loops and calculations run faster due to direct machine code execution.

- In JavaScript, performance depends on the JavaScript engine in the browser.

✅ **Solution?**

- JavaScript engines like **V8 (Chrome) and SpiderMonkey (Firefox)** have improved performance significantly.

- **WebAssembly (WASM)** can be used for high-performance tasks.

---

4. No Native Multi-threading Support

◈ JavaScript **runs on a single thread**, making heavy operations like **data processing** difficult.

✍ **The Problem?**

`javascript

function longTask() {

  for (let i = 0; i < 1e9; i++) {} // A long loop that freezes the page

}

longTask(); // The browser becomes unresponsive during execution!

`

☑ **Solution?**

Use **Web Workers** to run tasks in the **background** without freezing the UI.

`javascript

// worker.js

self.onmessage = function(event) {

  let result = event.data * 2;

  self.postMessage(result);

};

// main.js

```
let worker = new Worker(worker.js);
worker.postMessage(10);
worker.onmessage = function(event) {
  console.log(event.data); // 20
};
`
```

---

5. Inconsistent this Behavior

◈ The this keyword in JavaScript can be confusing because it depends on **how a function is called**, leading to common mistakes.

✍ **The Problem?**

```javascript
const obj = {
  name: Kareem,
  sayHello: function() {
    console.log(this.name);
  }
};

setTimeout(obj.sayHello, 1000); // undefined (Loses context)
```

`

💡 **Why?**

- When passing sayHello to setTimeout, it loses its context (his no longer refers to obj).

✅ **Solution?** Useind() or arrow functions:

`javascript

setTimeout(obj.sayHello.bind(obj), 1000); // Kareem

`

---

6. Security Vulnerabilities

🔷 JavaScript runs in the browser, making it vulnerable to attacks like:

- **Cross-Site Scripting (XSS):** Injecting malicious JavaScript into a website.

- **Cross-Site Request Forgery (CSRF):** Exploiting active user sessions to make unauthorized requests.

✅ **Solution?**

- **Avoid eval() because it allows executing dangerous code.**

- **Use Content Security Policy (CSP)** to prevent loading untrusted external scripts.

✍ **Example of a Dangerous XSS Attack:**

`javascript

document.write(<script>alert Your site is hacked!

◈ **Functional Programming in JavaScript**

🎯 **What is Functional Programming?**

Functional Programming (FP) is a **programming paradigm** that focuses on **using functions as fundamental building blocks** while avoiding **changing state (Immutable State)** and relying on **Pure Functions** to improve performance, readability, and reduce errors.

JavaScript naturally supports functional programming, making it **different from other languages** like Java or C++, which rely more on Object-Oriented Programming (OOP).

🎯 **Key Principles of Functional Programming in JavaScript**

◈ **1. First-Class Functions**

In JavaScript, **functions can be assigned to variables, passed as arguments, and returned from other functions**.

✍ **EXAMPLE:**

`JAVASCRIPT

FUNCTION SAYHELLO(NAME) {

  RETURN HELLO, !;

}

// ASSIGNING THE FUNCTION TO A VARIABLE

CONST GREET = SAYHELLO;

CONSOLE.LOG(GREET(KAREEM)); // HELLO, KAREEM!

`

✍ **PASSING A FUNCTION AS AN ARGUMENT (CALLBACK FUNCTION):**

`JAVASCRIPT

FUNCTION PROCESSUSERINPUT(CALLBACK) {

  LET NAME = AHMED;

  CONSOLE.LOG(CALLBACK(NAME));

}

PROCESSUSERINPUT(SAYHELLO); // HELLO, AHMED!

`

💡 **Difference from other languages:**

In Java, functions cannot be passed as variables this easily; instead, methods are used inside objects, making JavaScript more flexible.

◈ **2. Pure Functions**

A pure function **does not modify external variables**, always returning the same output for the same inputs.

✍ **Example of a pure function:**

`javascript

function add(a, b) {

  return a + b; // No external state modification

}

console.log(add(3, 4)); // 7

console.log(add(3, 4)); // 7 (Same input, same output)

`

✍ **Example of an impure function (because it modifies an external variable):**

`javascript

let total = 0;

```
function addToTotal(value) {
   total += value;  // Modifies external state
   return total;
}

console.log(addToTotal(5)); // 5
console.log(addToTotal(3)); // 8 (Output depends on external state)
`
```

💡 **JavaScript Advantage:**

Languages like Java and C++ often modify global variables, whereas JavaScript makes it easy to apply functional programming with pure functions.

◈ **3. Avoiding State Mutation (Immutable State)**

Functional programming encourages **not modifying data directly** but instead **creating new copies of data**.

✍ **Example: Creating a new array instead of modifying the original:**

`javascript

const numbers = [1, 2, 3, 4];

// INSTEAD OF MODIFYING THE ORIGINAL ARRAY, CREATE A NEW ONE

CONST DOUBLED = NUMBERS.MAP(NUM => NUM * 2);

CONSOLE.LOG(DOUBLED); // [2, 4, 6, 8]

CONSOLE.LOG(NUMBERS); // [1, 2, 3, 4] (UNCHANGED)

`

💡 **DIFFERENCE FROM OTHER LANGUAGES:**

IN JAVA OR C++, ARRAYS ARE MODIFIED DIRECTLY USING LOOPS (FOR), WHICH CAN CAUSE DATA CONSISTENCY ISSUES IN CONCURRENT PROGRAMMING.

🔷 **4. HIGHER-ORDER FUNCTIONS**

HIGHER-ORDER FUNCTIONS **ACCEPT OTHER FUNCTIONS AS ARGUMENTS OR RETURN FUNCTIONS**. THIS MAKES THE CODE MORE CONCISE AND EFFICIENT.

✍️ **EXAMPLE: USING MAP TO CREATE A NEW LIST WITHOUT MODIFYING THE ORIGINAL:**

`JAVASCRIPT

CONST NUMBERS = [1, 2, 3, 4];

CONST SQUARED = NUMBERS.MAP(NUM => NUM * NUM);

CONSOLE.LOG(SQUARED); // [1, 4, 9, 16]
`

✍ **EXAMPLE: A FUNCTION THAT RETURNS ANOTHER FUNCTION (CURRYING):**

`JAVASCRIPT
FUNCTION MULTIPLYBY(FACTOR) {
  RETURN FUNCTION(NUMBER) {
    RETURN NUMBER * FACTOR;
  };
}

CONST DOUBLE = MULTIPLYBY(2);
CONSOLE.LOG(DOUBLE(5)); // 10
CONSOLE.LOG(DOUBLE(10)); // 20
`

💡 **DIFFERENCE FROM OTHER LANGUAGES:**

IN JAVA, RETURNING FUNCTIONS FROM FUNCTIONS IS NOT AS STRAIGHTFORWARD AND REQUIRES **OBJECTS**.

◈ **5. Asynchronous Functional Programming**

JavaScript supports **Promises and sync/await** for handling asynchronous tasks functionally.

✍ **Example:**

`javascript

function fetchData() {

  return new Promise(resolve => {

    setTimeout(() => resolve(Data Loaded!), 2000);

  });

}

fetchData().then(console.log); // After 2 seconds: Data Loaded!

`

💡 **JavaScript Advantage:**

In Java or C++, **Threads** are used for asynchronous execution, whereas JavaScript provides sync/await for a simpler approach.

🎯 **Comparison: Functional vs Object-Oriented Programming**

Feature	**Functional Programming**	**Object-Oriented Programming (OOP)**
Main Focus	Functions and transformations	Objects and their interactions
State Management	Immutable data	Objects store mutable state
Key Concepts	Pure functions, no state mutation	Encapsulation, Inheritance
Performance	Faster in some cases (no modification)	Slower with complex objects
Concurrency	Easier for asynchronous programming	More complex with Threads

---

🎯 **💡 Why Use Functional Programming in JavaScript?**

- ✅ **Cleaner and more reusable code**
- ✅ **Improved performance and fewer unpredictable errors**
- ✅ **Easier asynchronous programming with Promises and sync/await**
- ✅ **Ideal for front-end development using frameworks like React**

🔥 **💡 When to Use Functional Programming?**

- When **clean and readable** code is needed

- When handling **large datasets without modifying them directly**

- When building **dynamic UI interfaces that require state updates**

🚀 **Do you have any questions or need more examples? 😊**

🛠️ Understanding Developer Tools in the Browser (Developer Console) 💻

🔷 What are Developer Tools?

Developer Tools are a set of built-in tools in modern browsers such as **Google Chrome, Firefox, Edge**, and others that allow you to inspect pages, debug errors, and analyze performance.

🔷 How to Open Developer Tools?

You can open Developer Tools in several ways:

✅ **Quick Shortcut:**

- **Windows/Linux:** Press F12 or Ctrl + Shift + I

- **Mac:** Press Cmd + Option + I

✅ **Or from the menu:**

1⃣ Click on the **menu button** (⋮ in Chrome).

2⃣ Select **More Tools**.

3⃣ Choose **Developer Tools**.

🔷 Key Features of Developer Console

🔵 **1- Console (Command Line)**

- Used to execute **JavaScript** code directly in the browser.
- Displays errors and warnings on the page.
- You can write and test any **JavaScript** code instantly!

```js
console.log('Hello Developer Console!');
```

🔵 **2- Elements (Page Structure)**

- Displays the **HTML and CSS** structure of the page.
- Allows you to **modify elements** and see changes immediately.
- Very useful for **CSS debugging**.

🔘 **3- Network (Network Analysis)**

- Shows all **HTTP requests** sent and received from the server.

- Used to analyze **page load times** and improve website performance.

🔘 **4- Sources (File Sources)**

- Displays **JavaScript, CSS, and image** files of the website.

- Allows you to set **breakpoints** for debugging JavaScript.

🔘 **5- Application (Storage and Cookies)**

- Displays stored data in **Local Storage, Session Storage, IndexedDB, Cookies**.

- Useful for inspecting and managing **user data stored in the browser**.

🔘 **6- Performance (Performance Analysis)**

- Helps **analyze page speed** and optimize performance.

🎯 Summary: Why Use Developer Console?

✅ **Debug JavaScript errors** in real-time.

✅ **Analyze website performance** and improve speed.

✅ **Modify HTML and CSS instantly** without changing the actual code.

✅ **Test and experiment with code** easily inside the browser.

🛠️ **Developer Tools are your secret weapon for understanding and optimizing any website! 🚀**

```
console.log('Hello World!'); // هذا الكود يطبع Hello World! في وحدة التحكم (Console)
```

```
// 🟣 تعريف متغير باستخدام var
var userName = 'Ali';  // يمكن إعادة تعريفه وتغيير قيمته
console.log('User Name:', userName); // Ali

var userName = 'Omar'; // ✅ إعادة تعريف المتغير مسموحة
console.log('User Name:', userName); // Omar

userName = 'Ahmed'; // ✅ تغيير القيمة مسموح
console.log('User Name:', userName); // Ahmed

// 🟣 تعريف متغير باستخدام let
let userAge = 25; // let لا يسمح بإعادة التعريف لكنه يسمح بتغيير القيمة
console.log('User Age:', userAge); // 25
```

```
// let userAge = 30;  ❌ خطأ: لا يمكن إعادة تعريف متغير let بنفس الاسم

userAge = 30; // ✅ تغيير القيمة مسموح
console.log('User Age:', userAge); // 30

// 🔵 تعريف ثابت باستخدام const
const PI = 3.14; // const لا يمكن إعادة تعريفه أو تغيير قيمته
console.log('PI Value:', PI); // 3.14

// PI = 3.1415;  ❌ خطأ: لا يمكن تغيير قيمة ثابت
// console.log(PI);

// 🔵 الفرق في نطاق المتغيرات (Scope)
if (true) {
   var x = 10;  // var لديه نطاق Function Scope
   let y = 20;  // let لديه نطاق Block Scope
}
console.log('Var x:', x); // ✅ يعمل لأن var متاح خارج if
// console.log(y);  ❌ خطأ: y غير متاحة خارج الـ if لأنها معرفة بـ let

// 🔵 قواعد تسمية المتغيرات
let camelCaseExample = '✔ Valid Variable Name';  // ✅ اسم متغير صحيح
```

```
LET DOLLARSIGN = 100;  // ✅ يمكن استخدام $ في الأسماء
LET _SCORE = 50;  // ✅ يمكن استخدام _ في الأسماء
// LET 1NUMBER = 10; ❌ خطأ: لا يمكن أن يبدأ برقم
// LET LET = 'ERROR'; ❌ خطأ: لا يمكن استخدام الكلمات المحجوزة

CONSOLE.LOG('EXAMPLES:', CAMELCASEEXAMPLE, DOLLARSIGN, _SCORE);

// 🔵 **1- النصوص (STRINGS)**
LET STUDENTNAME = 'ALI'; // نص يمثل اسم الطالب
CONSOLE.LOG('STUDENT NAME:', STUDENTNAME); // ✅ طباعة الاسم في CONSOLE

// 🔵 **2- الأرقام (NUMBERS)**
LET STUDENTAGE = 20; // رقم يمثل عمر الطالب
LET STUDENTGRADE = 95.5; // يمكن أن تكون الأرقام صحيحة أو عشرية
CONSOLE.LOG('STUDENT AGE:', STUDENTAGE);
CONSOLE.LOG('STUDENT GRADE:', STUDENTGRADE);

// 🔵 **3- القيم المنطقية (BOOLEAN)**
LET ISPASSED = TRUE; // قيمة منطقية تعبر عن النجاح
CONSOLE.LOG('DID THE STUDENT PASS?', ISPASSED);

// 🔵 **4- المصفوفات (ARRAYS)**
```

```
LET STUDENTS = ['ALI', 'OMAR', 'SARA', 'NADA']; // مصفوفة تحتوي على أسماء الطلاب
CONSOLE.LOG('STUDENTS LIST:', STUDENTS); // ✅ طباعة المصفوفة

CONSOLE.LOG('FIRST STUDENT:', STUDENTS[0]); // ✅ طباعة أول عنصر
CONSOLE.LOG('TOTAL STUDENTS:', STUDENTS.LENGTH); // ✅ طباعة عدد العناصر

// 🔘 **5- الكائنات (OBJECTS)**
LET STUDENTINFO = {
  NAME: 'ALI',
  AGE: 20,
  JOB: 'STUDENT',
  GRADE: 95.5
}; // كائن يحتوي على معلومات الطالب

CONSOLE.LOG('STUDENT INFORMATION:', STUDENTINFO); // ✅ طباعة الكائن بالكامل
CONSOLE.LOG('STUDENT NAME:', STUDENTINFO.NAME); // ✅ طباعة قيمة محددة من الكائن

// 💡 **تمرين عملي إضافي: إضافة طالب جديد إلى المصفوفة**
STUDENTS.PUSH('KHALED'); // إضافة طالب جديد
CONSOLE.LOG('UPDATED STUDENTS LIST:', STUDENTS);

// 💡 **تمرين عملي إضافي: تعديل عمر الطالب داخل الكائن**
```

```
STUDENTINFO.AGE = 21; // تعديل قيمة العمر
CONSOLE.LOG('UPDATED STUDENT INFO:', STUDENTINFO);

// العمليات الحسابية بين عددين
LET NUM1 = 10;
LET NUM2 = 5;

// جمع العددين
CONSOLE.LOG('SUM:', NUM1 + NUM2);

// طرح العددين
CONSOLE.LOG('SUBTRACTION:', NUM1 - NUM2);

// ضرب العددين
CONSOLE.LOG('MULTIPLICATION:', NUM1 * NUM2);

// قسمة العدد الأول على العدد الثاني
CONSOLE.LOG('DIVISION:', NUM1 / NUM2);

// باقي القسمة بين العددين
CONSOLE.LOG('MODULUS:', NUM1 % NUM2);
```

```
// عمليات المقارنة بين العددين
CONSOLE.LOG('IS NUM1 GREATER THAN NUM2?', NUM1 > NUM2);
CONSOLE.LOG('IS NUM1 SMALLER THAN NUM2?', NUM1 < NUM2);
CONSOLE.LOG('IS NUM1 EQUAL TO NUM2 (WITHOUT TYPE CHECK)?', NUM1 == NUM2);
CONSOLE.LOG('IS NUM1 STRICTLY EQUAL TO NUM2?', NUM1 === NUM2);

// العمليات المنطقية للتحقق من الشروط
LET ISTRUE = (NUM1 > 5) && (NUM2 < 10); // تحقق أن كلا الشرطين صحيحان معاً
CONSOLE.LOG('BOTH CONDITIONS ARE TRUE:', ISTRUE);

LET ISEITHERTRUE = (NUM1 > 5) || (NUM2 > 10); // تحقق إن كان أحد الشرطين صحيحاً
CONSOLE.LOG('AT LEAST ONE CONDITION IS TRUE:', ISEITHERTRUE);

// تعريف دالة تقوم بجمع رقمين وإرجاع الناتج
FUNCTION ADDNUMBERS(A, B) {
   RETURN A + B;
}

// استدعاء الدالة مع تمرير القيم 7 و 3 وتخزين الناتج
LET RESULT = ADDNUMBERS(7, 3);

// طباعة ناتج الجمع
```

```
CONSOLE.LOG('SUM RESULT:', RESULT);


// إشارات وعلامات الجافا سكريبت مع شرح بالعربية


// 1. الفاصلة المنقوطة (;) - تستخدم لإنهاء الأوامر في JAVASCRIPT
LET X = 10;
LET Y = 20; // يمكن الاستغناء عنها في بعض الحالات


// 2. الأقواس العادية () - تستخدم لاستدعاء الدوال وتمرير القيم
FUNCTION SAYHELLO(NAME) {
   RETURN 'HELLO ' + NAME;
}
CONSOLE.LOG(SAYHELLO('KAREEM'));


// 3. الأقواس المعقوفة {} - تستخدم لتعريف كتل الكود مثل الدوال والحلقات
IF (X > Y) {
   CONSOLE.LOG('X IS GREATER THAN Y');
} ELSE {
   CONSOLE.LOG('Y IS GREATER THAN X');
}


// 4. الأقواس المربعة [] - تستخدم لإنشاء المصفوفات أو الوصول إلى القيم داخلها
```

```
LET NUMBERS = [1, 2, 3, 4, 5];
CONSOLE.LOG('FIRST ELEMENT:', NUMBERS[0]);

// 5. علامة المساواة (=) - تستخدم للإسناد (تعيين قيمة لمتغير)
LET NAME = 'KAREEM'; // تعيين قيمة النص 'KAREEM' للمتغير NAME

// 6. علامات المقارنة (== و ===) - تستخدم للمقارنة بين القيم
CONSOLE.LOG('COMPARISON:', 5 == '5');  // TRUE (مقارنة القيم فقط)
CONSOLE.LOG('STRICT COMPARISON:', 5 === '5'); // FALSE (مقارنة القيم والنوع)

// 7. العمليات الحسابية (+, -, *, /, %) - تستخدم لإجراء العمليات الرياضية
CONSOLE.LOG('ADDITION:', 10 + 5); // 15
CONSOLE.LOG('SUBTRACTION:', 10 - 5); // 5
CONSOLE.LOG('MULTIPLICATION:', 10 * 5); // 50
CONSOLE.LOG('DIVISION:', 10 / 5); // 2
CONSOLE.LOG('MODULO:', 10 % 3); // 1 (الباقي)

// 8. العمليات المنطقية (&&, ||, !) - تستخدم للتحقق من الشروط المنطقية
CONSOLE.LOG('AND OPERATION:', TRUE && FALSE); // FALSE
CONSOLE.LOG('OR OPERATION:', TRUE || FALSE); // TRUE
CONSOLE.LOG('NOT OPERATION:', !TRUE); // FALSE
```

```
// 9. علامة التعجب (!) - تستخدم لنفي القيم المنطقية
LET ISACTIVE = FALSE;
CONSOLE.LOG('NEGATION:', !ISACTIVE); // TRUE

// 10. الفاصلة (,) - تستخدم لفصل العناصر في المصفوفات أو تمرير المعاملات للدوال
LET COLORS = ['RED', 'GREEN', 'BLUE'];
CONSOLE.LOG('SECOND COLOR:', COLORS[1]);

// 11. علامة الاستفهام (؟) وعلامة النقطتين (:) - المشغل الثلاثي لاختصار الشروط
LET AGE = 18;
LET STATUS = (AGE >= 18) ? 'ADULT' : 'MINOR';
CONSOLE.LOG('STATUS:', STATUS);

// 12. النقطة (.) - تستخدم للوصول إلى خصائص الكائنات أو استدعاء الدوال
LET PERSON = { NAME: 'KAREEM', AGE: 30 };
CONSOLE.LOG('PERSON NAME:', PERSON.NAME);

// 13. العلامة النجمية (*) - تستخدم في تعريف الدوال التوليدية أو عمليات الضرب
FUNCTION* GENERATOR() {
  YIELD 1;
  YIELD 2;
  YIELD 3;
```

```
}
LET GEN = GENERATOR();
CONSOLE.LOG('GENERATOR OUTPUT:', GEN.NEXT().VALUE); // 1

// 14. علامة السهم (=>) - تستخدم في تعريف الدوال السهمية
CONST ADD = (A, B) => A + B;
CONSOLE.LOG('ARROW FUNCTION RESULT:', ADD(3, 4));

// 15. علامات BACKTICKS (` `) - تستخدم للقوالب النصية وإدراج المتغيرات
LET MESSAGE = `HELLO, YOU ARE AGE YEARS OLD`;
CONSOLE.LOG(MESSAGE);

// 16. علامة @ - تُستخدم في بعض الإطارات مثل ANGULAR للديكورات
// @COMPONENT({ SELECTOR: 'APP-ROOT' })

// 17. الشرطة السفلية (_) - يمكن استخدامها في تسمية المتغيرات
LET _PRIVATEVAR = 'THIS IS A PRIVATE VARIABLE';

// 18. الشرطة (-) - تُستخدم في الطرح وفي أسماء الخصائص في CSS
CONSOLE.LOG('SUBTRACTION:', 10 - 3); // 7

// 19. علامتا الإزاحة (<<, >>) - تستخدم لإزاحة القيم الثنائية
```

```
CONSOLE.LOG('LEFT SHIFT:', 4 << 1); // 8
CONSOLE.LOG('RIGHT SHIFT:', 4 >> 1); // 2

// 20. علامة& و | - تُستخدم في العمليات الثنائية
CONSOLE.LOG('BITWISE AND:', 5 & 1); // 1
CONSOLE.LOG('BITWISE OR:', 5 | 1); // 5

// 21. علامة ^ - XOR في العمليات الثنائية (الإقصاء الحصري)
CONSOLE.LOG('BITWISE XOR:', 5 ^ 1); // 4


// دالة تعادل NOW() في EXCEL وتعيد التاريخ والوقت الحالي بتنسيق واضح
FUNCTION NOW() {
  RETURN NEW DATE().TOLOCALESTRING('EN-US'); // إرجاع التاريخ والوقت الحالي بتنسيق إنجليزي
}

CONSOLE.LOG('📅 CURRENT DATE & TIME:', NOW());

// دالة لحساب مجموع ثلاثة أرقام مثل SUM(A1 + B1 + C1) في EXCEL
FUNCTION SUMCELLS(A, B, C) {
  RETURN A + B + C; // إرجاع مجموع القيم المدخلة
}
```

```
// مثال عملي
LET A1 = 10, B1 = 20, C1 = 30;
CONSOLE.LOG('SUM IS:', SUMCELLS(A1, B1, C1)); // 60 يجب أن يكون الناتج

// دالة تعادل IF(A1 > B1, 'YES', 'NO') في EXCEL
FUNCTION CHECKCONDITION(A, B) {
   RETURN A > B ? 'YES' : 'NO';
}

LET A1 = 10, B1 = 5;
CONSOLE.LOG('RESULT:', CHECKCONDITION(A1, B1));

// دالة تعادل IF(AND(A1 > B1, A1 > C1), 'YES', 'NO') في EXCEL
FUNCTION CHECKMULTIPLECONDITIONS(A, B, C) {
   RETURN (A > B && A > C) ? 'YES' : 'NO';
}

LET A1 = 10, B1 = 5, C1 = 3;
CONSOLE.LOG('RESULT:', CHECKMULTIPLECONDITIONS(A1, B1, C1));

// دالة تعادل IF(OR(A1 > B1, A1 > C1), 'YES', 'NO') في EXCEL
```

```
FUNCTION CHECKORCONDITION(A, B, C) {
   RETURN (A > B || A > C) ? 'YES' : 'NO';
}

LET A1 = 10, B1 = 20, C1 = 5;
CONSOLE.LOG('RESULT:', CHECKORCONDITION(A1, B1, C1));

// دالة تعادل IF(ISODD(MID(A1,3,2)), 'MALE', 'FEMALE') في EXCEL
FUNCTION CHECKGENDER(ID) {
   LET EXTRACTEDNUMBER = PARSEINT(ID.SUBSTRING(2, 4));
   RETURN EXTRACTEDNUMBER % 2 !== 0 ? 'MALE' : 'FEMALE';
}

LET A1 = '123456789';
CONSOLE.LOG('GENDER:', CHECKGENDER(A1));

FUNCTION CHECKGENDER(VALUE) {
   LET EXTRACTEDNUMBER = PARSEINT(VALUE.SUBSTRING(2, 4)); // استخراج الرقم من الموقع
الثالث والرابع

   IF (EXTRACTEDNUMBER % 2 !== 0) {
      RETURN "MALE";
   } ELSE {
```

```
        RETURN "FEMALE";
    }
}

// مثال عملي
LET A1 = "123456"; // مثال على قيمة خلية EXCEL
CONSOLE.LOG("GENDER:", CHECKGENDER(A1)); // إخراج النتيجة في الكونسول

// مصفوفة تحتوي على قيم متعددة مثل القيم في عدة خلايا
LET VALUES = ["123456", "987654", "456789", "135790"];

FUNCTION CHECKGENDER(VALUE) {
    LET EXTRACTEDNUMBER = PARSEINT(VALUE.SUBSTRING(2, 4)); // استخراج الرقم من الموقع
الثالث والرابع
    RETURN EXTRACTEDNUMBER % 2 !== 0 ? "MALE" : "FEMALE";
}

// استخدام حلقة FOR لمعالجة كل قيمة في المصفوفة
FOR (LET I = 0; I < VALUES.LENGTH; I++) {
    CONSOLE.LOG("A:", VALUES[I], "GENDER:", CHECKGENDER(VALUES[I]));

}
```

** 📌 What is Artificial Intelligence (AI) and How Does It Work?**

** 📌 Definition of Artificial Intelligence (AI)**

Artificial Intelligence (AI) is a branch of computer science that aims to **design systems and programs capable of simulating human intelligence**. These systems can learn, think, make decisions, and even interact with humans intelligently.

💡 **Examples of AI in Daily Life:**

- **Google Assistant / Siri / Alexa** 🗣 → Recognize your voice and respond to your queries.

- **Smart Camera Filters** 📷 → Detect faces and enhance images automatically.

- **Movie and Series Recommendations** 🎬 → Platforms like **Netflix and YouTube** suggest content based on your interests.

- **Self-Driving Cars** 🚗 → Like Tesla, which uses AI to make driving decisions.

🧠 How Does AI Work?

AI follows several steps to process data and make intelligent decisions. Let's break down each step:

** 📌 1. Data Collection (📊)**

◈ Data is the **fuel of AI**, enabling it to learn and make decisions.

◈ Data is collected from various sources, such as:

- Images 📷 (e.g., facial recognition systems).
- Texts 📝 (e.g., chatbot interactions).
- Numbers 🔢 (e.g., sales data in e-commerce).
- Videos 🎥 (e.g., self-driving car monitoring systems).

💡 **Real-Life Example:**

When developing an AI system for recognizing handwritten numbers, we collect **thousands of images of handwritten digits** to train the system.

** 📌 2. Data Processing (⚙️)**

◈ After data collection, the system **cleans and analyzes** the data to identify patterns.

◈ Some key steps in this phase include:

- **Data Cleaning:** Removing unnecessary or duplicate data.

- **Data Analysis:** Extracting patterns and relationships.

- **Data Transformation:** Converting data into a format that the system can process efficiently.

💡 **Real-Life Example:**

In an AI image recognition system, each image is converted into **a set of numbers** representing colors, making it easier for the computer to analyze.

📌 3. Decision Making (✅)

🔷 The system uses **algorithms and mathematical models** to make decisions based on the learned data.

🔷 Various types of algorithms assist in decision-making, such as:

- **Classification Algorithms:** Determine the category of an item (e.g., classifying an email as important or spam).

- **Prediction Algorithms:** Forecast future values (e.g., predicting stock prices).

- **Search Algorithms:** Find the best solution (e.g., GPS algorithms finding the shortest route).

💡 **Real-Life Example:**

When you search on Google, AI analyzes your keywords and presents **the most relevant results** using smart search algorithms.

** 📌 4. Self-Learning & Optimization (🤖)**

🔹 **This is where AI gets smarter!** The system analyzes its errors and improves itself based on experience.

🔹 This is done through **Machine Learning**, where the system learns automatically from new data without additional programming.

🔹 Two main types of Machine Learning:

- **Supervised Learning:** Uses labeled data, such as training a model to recognize cats and dogs using pre-labeled images.
- **Unsupervised Learning:** Learns from data without labels, such as customer data analysis to find hidden patterns.

💡 **Real-Life Example:**

When you use **YouTube**, it learns from **your watch history** and suggests similar videos based on your preferences!

** 📌 Practical Example: AI in JavaScript**

Let's create a simple AI function that predicts whether a user likes coffee based on input:

`JAVASCRIPT

```
function lovesCoffee(timeOfDay, isTired) {
  if (timeOfDay === morning || isTired) {
    return يبدو أنك تحتاج إلى فنجان قهوة! ☕;
  } else {
    return أنت نشيط، لا تحتاج إلى قهوة الآن! 🚀;
  }
}

console.log(lovesCoffee(morning, true)); // يبدو أنك تحتاج إلى فنجان قهوة! ☕
console.log(lovesCoffee(evening, false)); // أنت نشيط، لا تحتاج إلى قهوة الآن! 🚀
```
`

👆 This is a simple example demonstrating **how AI makes decisions based on input data**.

💡 Summary of How AI Works

1️⃣ **Data Collection** 📊 → Input data from various sources (images, text, numbers, videos).

2️⃣ **Data Processing** ⚙️ → Cleaning, analyzing, and extracting patterns.

3️⃣ **Decision Making** ✅ → Using algorithms to choose **the best possible outcome**.

4️⃣**Self-Learning** 🤖 → Improving performance over time based on experience.

🚀 How Can You Learn AI?

If you're interested in diving into AI, here are some key areas to study:

✅ **Machine Learning**

✅ **Neural Networks**

✅ **Natural Language Processing (NLP)**

✅ **Data Science**

✅ **AI Development with JavaScript and Python**

🔷 **Popular AI Libraries:**

- **TensorFlow.js** → AI development with JavaScript.

- **Brain.js** → Easy neural network creation.

- **Python (NumPy, Pandas, Scikit-learn, TensorFlow)** → Widely used in AI development.

📌 Do you have any other questions about AI? 😊🚀

Why Use JavaScript for Artificial Intelligence?

◈ Why Use JavaScript in AI?

✅ **Runs Directly in the Browser** 🌐:

JavaScript runs directly in the browser, allowing AI models to be executed **without requiring powerful servers or complex runtime environments**.

✅ **High Performance with WebGL & WebGPU** 🚀:

JavaScript can leverage **WebGL** and **WebGPU** to accelerate computations using **GPU processing**, significantly boosting AI model performance in the browser.

✅ **Seamless Integration with Web Applications** 📱:

AI can be easily integrated into web applications for **image and speech recognition, real-time translation, text analysis, and smart recommendations** using libraries like **TensorFlow.js and Brain.js**.

✅ **Easy Learning Curve for Web Developers** 👨‍💻:

If you're a web developer, using JavaScript for AI **gives you a major advantage**, as you can work within your existing environment without learning a new language.

✅ **Run Models Locally** 💻:

Instead of sending data to **external servers** for processing, JavaScript enables AI models to run **locally on the user's device**, enhancing privacy and reducing latency.

🔷 JavaScript AI Features

Feature	JavaScript
Easy to Learn	✅ Ideal for web developers
Performance	✅ Supports WebGPU for better speed
AI Support	✅ Integrated with **TensorFlow.js & Brain.js**
Model Execution	✅ Runs **directly in the browser**
Web Compatibility	✅ Best for web applications
Use Cases	🎯 Perfect for **interactive AI and fast applications**

🔷 When to Use JavaScript for AI?

✅ **When developing web applications that use AI directly in the browser** (e.g., speech or image recognition).

✅ **When processing data locally on the user's device** without external servers.

✅ **When you want to integrate AI into your web apps effortlessly** as a web developer.

✅ **When building interactive AI-powered applications**, such as games and smart chatbots.

🔷 Top JavaScript Libraries for AI

📌 **TensorFlow.js** → The most powerful library for training and running AI models in the browser.

📌 **Brain.js** → A simple library for creating neural networks in JavaScript.

📌 **Synaptic.js** → A flexible neural network library.

📌 **ml5.js** → A user-friendly library built on TensorFlow.js.

🔷 Practical Example: Simple AI Model Using TensorFlow.js

`javascript

// Import TensorFlow.js

import * as tf from '@tensorflow/tfjs';

```
// Create a simple prediction model
const model = tf.sequential();
model.add(tf.layers.dense({ units: 1, inputShape: [1] }));

// Compile the model
model.compile({ loss: 'meanSquaredError', optimizer: 'sgd' });

// Training data
const xs = tf.tensor2d([1, 2, 3, 4], [4, 1]);
const ys = tf.tensor2d([1, 3, 5, 7], [4, 1]);

// Train the model
model.fit(xs, ys, { epochs: 250 }).then(() => {
  // Make a prediction
  model.predict(tf.tensor2d([5], [1, 1])).print();
});


class DecisionTree {
  constructor(data) {
    // عند إنشاء كائن من الفئة، يتم استدعاء هذه الدالة تلقائيًا
    // يتم تخزين الشجرة الناتجة من البيانات المدخلة في المتغير this.tree
    this.tree = this.buildTree(data);
```

```
}

BUILDTREE(DATA) {
  // تصفية البيانات لتحديد عدد الذكور والإناث في بيانات التدريب
  LET MALES = DATA.FILTER(D => D.GENDER === 'MALE'); // استخراج الذكور فقط
  LET FEMALES = DATA.FILTER(D => D.GENDER === 'FEMALE'); // استخراج الإناث فقط

  RETURN {
    // تعريف قاعدة لتحديد الجنس بناءً على رقم الهوية
    RULE: (ID) => {
      // استخراج الرقمين الثالث والرابع من رقم الهوية
      LET EXTRACTEDNUMBER = PARSEINT(ID.SUBSTRING(2, 4));
      // إذا كان الرقم فرديًا، يتم توجيهه إلى LEFT، وإذا كان زوجيًا، يتم توجيهه إلى RIGHT
      RETURN EXTRACTEDNUMBER % 2 !== 0;
    },

    // تحديد الجنس في الفرع الأيسر بناءً على عدد الذكور والإناث
    LEFT: MALES.LENGTH > FEMALES.LENGTH ? 'MALE' : 'FEMALE',

    // تحديد الجنس في الفرع الأيمن بناءً على عدد الذكور والإناث
    RIGHT: FEMALES.LENGTH >= MALES.LENGTH ? 'FEMALE' : 'MALE'
  };
```

```
    }

    PREDICT(ID) {
        // لمعرفة ما إذا كان الرقم فرديًا أو زوجيًا RULE استدعاء القاعدة
        // RIGHT ، وإذا كان زوجيًا يتم إرجاع قيمة LEFT إذا كان فرديًا، يتم إرجاع قيمة
        RETURN THIS.TREE.RULE(ID) ? THIS.TREE.LEFT : THIS.TREE.RIGHT;
    }
}

// بيانات التدريب المستخدمة لتعليم النموذج
LET TRAININGDATA = [
    { ID:  123456789, GENDER: MALE },
    { ID: 987654321, GENDER: FEMALE },
    { ID: 135792468, GENDER: MALE },
    { ID: 246813579, GENDER: FEMALE },
];

// مع تمرير بيانات التدريب إليه DECISIONTREE إنشاء كائن من الفئة
LET MODEL = NEW DECISIONTREE(TRAININGDATA);

// اختبار النموذج بتوقع الجنس لرقم هوية معين
CONSOLE.LOG(MODEL.PREDICT(314159265)); // طباعة التوقع الناتج بناءً على رقم الهوية المدخل
```

21. **Decision Tree**

1. **Logistic Regression**

2. **Linear Regression**

3. **Random Forest**

4. **K-Nearest Neighbors (KNN)**

5. **Naïve Bayes**

6. **Artificial Neural Networks (ANN)**

7. **Gradient Boosting (XGBoost, LightGBM, CatBoost)**

8. **K-Means Clustering**

9. **Principal Component Analysis (PCA)**

10. **Support Vector Machines (SVM)**

11. **Recurrent Neural Networks (RNN)**

12. **Long Short-Term Memory (LSTM)**

13. **Convolutional Neural Networks (CNN)**

14. **Bayesian Networks**

15. **Genetic Algorithms**

16. **Deep Q-Networks (DQN)**

17. **Autoencoders**

18. **Self-Organizing Maps (SOM)**

19. **Restricted Boltzmann Machines (RBM)**

20. **Markov Chains**