\*\*🔹 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

1️⃣ 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

```

2️⃣ 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

```

3️⃣ 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    his Behavior\*\*

🔹 The   his 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 (♀or), 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:\*\*

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

2️⃣ Select \*\*More Tools\*\*.

3️⃣ 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 (الإقصاء الحصري) في العمليات الثنائية

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)); // يجب أن يكون الناتج 60

// 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) {

        // استدعاء القاعدة rule لمعرفة ما إذا كان الرقم فرديًا أو زوجيًا

        // إذا كان فرديًا، يتم إرجاع قيمة left، وإذا كان زوجيًا يتم إرجاع قيمة right

        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 },

];

// إنشاء كائن من الفئة DecisionTree مع تمرير بيانات التدريب إليه

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\*\*