

JavaScript

CRASHCOURSE

in **3** Hours
with Notes



[Video Link](#)

KG Coding



Some Other One shot Video Links:

- Complete Web Development
- Complete Backend
- Complete Java
- Complete C Programming
- One shot University Exam Series

<http://www.kgcoding.in/>

Our YouTube Channels

[KG Coding Android App](#)



[KG Coding](#)



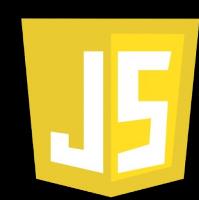
[Knowledge GATE](#)



[KG Placement Prep](#)



[Sanchit Socket](#)



Introduction to JavaScript

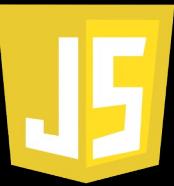
1. History of JavaScript
2. What is JavaScript
3. Popularity of JavaScript
4. Applications of JavaScript
5. Runtime Environment
6. JavaScript vs ECMA
7. JavaScript vs TypeScript
8. JavaScript in Console
9. JavaScript in Webpage
10. DOM Manipulation
11. JavaScript with Node



1. History of JavaScript



1. JavaScript was originally named Mocha, then renamed to LiveScript, and finally JavaScript to capitalize on the popularity of Java at the time.
2. JavaScript was created by Brendan Eich in 1995 while he was working at Netscape Communications Corporation.
3. JavaScript is an interpreted language, meaning it is executed line by line.



2. What is JavaScript

1. **JavaScript** is a high-level, **dynamic** programming language commonly used for creating interactive effects within web browsers.
2. **Actions:** Enables **interactivity**.
3. **Updates:** Alters page **without** reloading.
4. **Events:** Responds to user actions.
5. **Data:** Fetches and sends info to server.



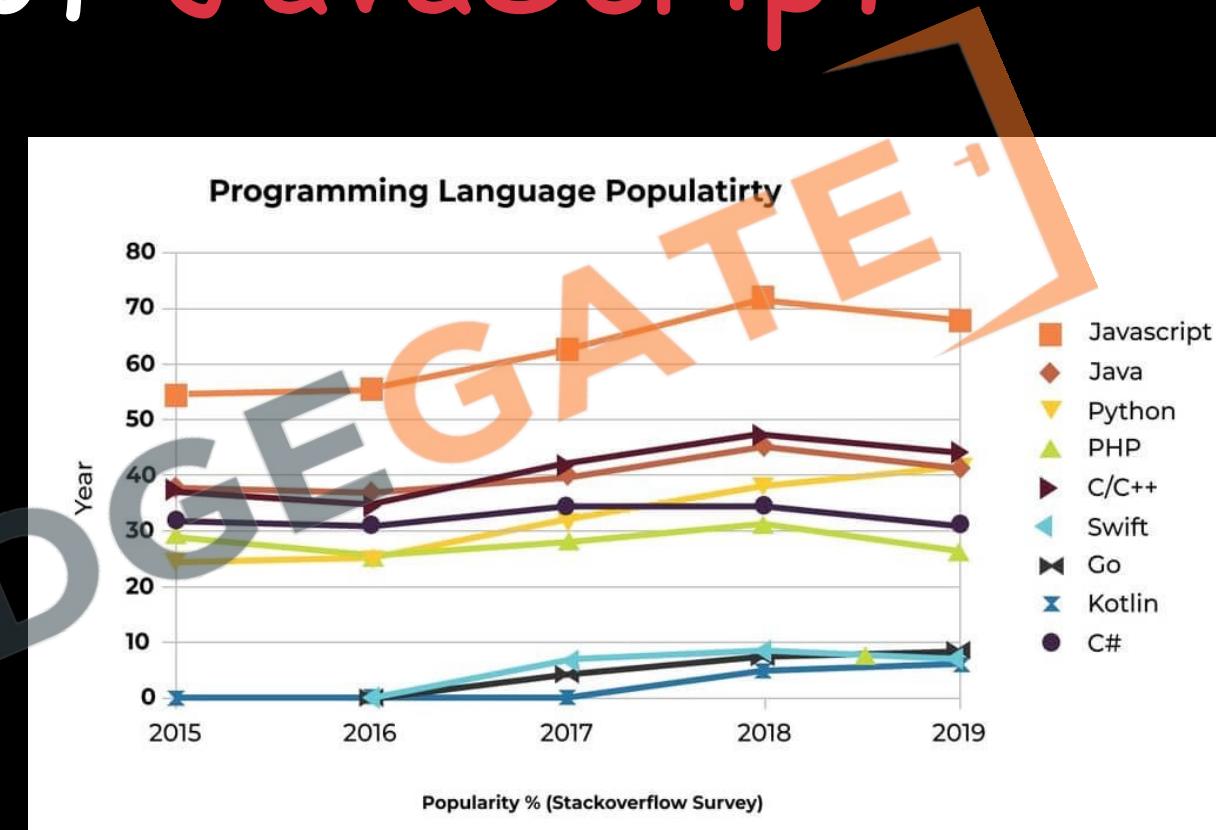


3. Popularity of JavaScript

1. **JavaScript** is one of the most popular programming languages in the world, consistently ranking at the top in surveys and job listings.

2. Average **JavaScript** Dev Salary in India:

- Entry-Level (0-1 year): Around ₹3,50,000 per annum.
- Mid-Level (2-5 years): Approximately ₹6,00,000 to ₹10,00,000 per annum.
- Experienced (5+ years): Can exceed ₹10,00,000 per annum, potentially reaching up to ₹20,37,500.





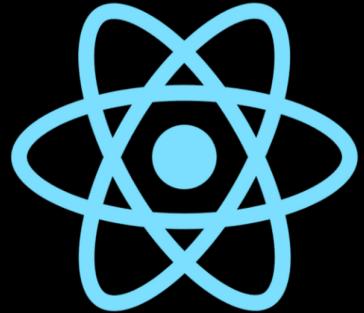
4. Applications of JavaScript



- KNOWLEDGEABLE**
1. **HTML**: Defines the **structure and content** of the **website**.
 2. **CSS**: Specifies the **appearance and layout** of the **website**.
 3. **JavaScript**: Adds **interactivity and dynamic behavior** to the **website**.



4. Applications of JavaScript



React JS



Web Applications:

- **React**: A library for **building user interfaces**, maintained by Facebook.
- **Angular**: A platform for building **mobile and desktop web applications**, maintained by Google.
- **Vue.js**: A progressive framework for building user interfaces.



4. Applications of JavaScript

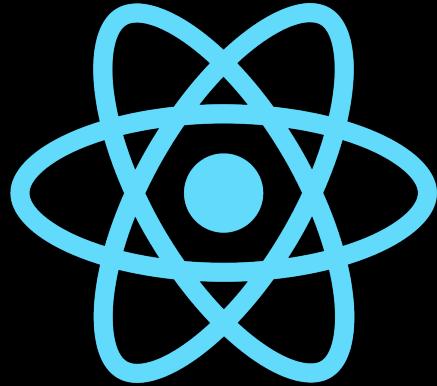


Server-Side:

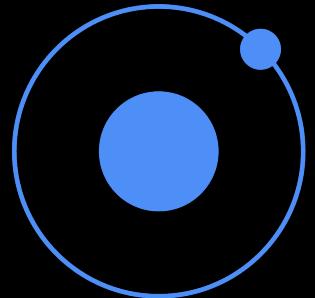
- **Node.js:** Allows JavaScript to run on the server, used for building scalable network applications.
- **Express.js:** A minimal and flexible Node.js web application framework.



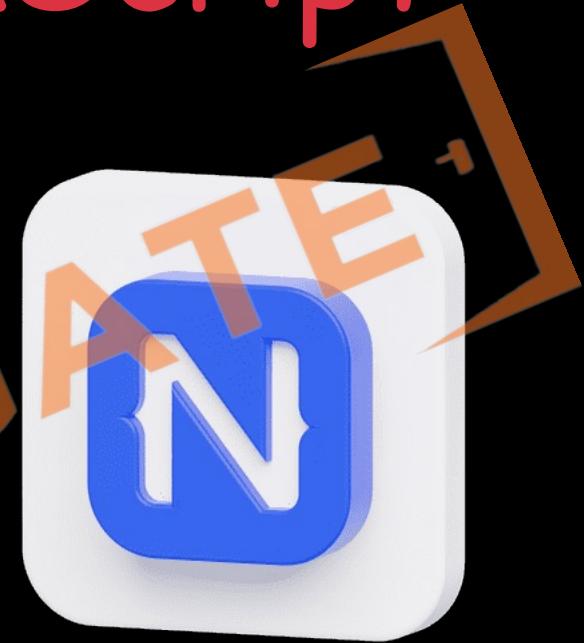
4. Applications of JavaScript



React Native

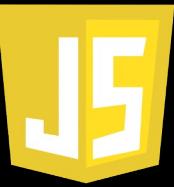


ionic

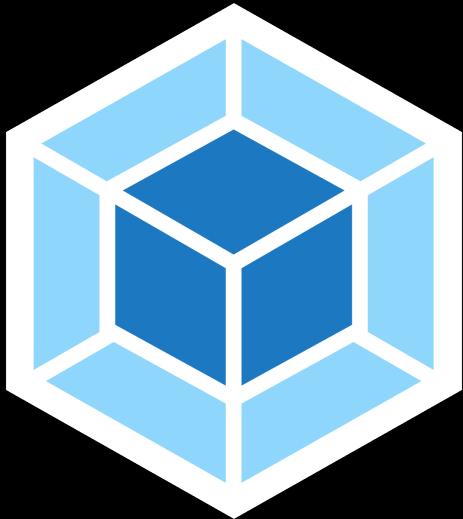


Mobile Applications:

- **React Native:** Builds **mobile apps** using **JavaScript** and **React**.
- **Ionic:** A framework for building **cross-platform mobile apps** with web technologies like **HTML**, **CSS**, and **JavaScript**.
- **NativeScript:** Allows building **native iOS** and **Android apps** using **JavaScript** or **TypeScript**.

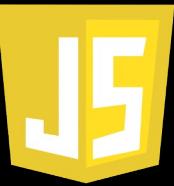


4. Applications of JavaScript

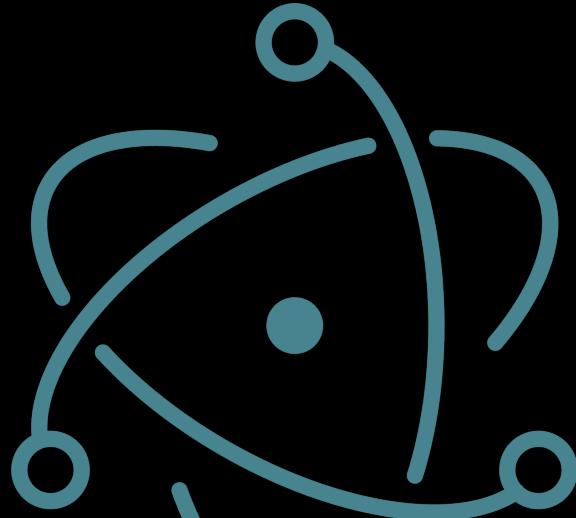


BuildTools:

- **Webpack:** A module bundler for JavaScript applications.
- **Parcel:** A fast, zero-configuration web application bundler.
- **Gulp:** A toolkit to automate tasks in your development workflow.



4. Applications of JavaScript

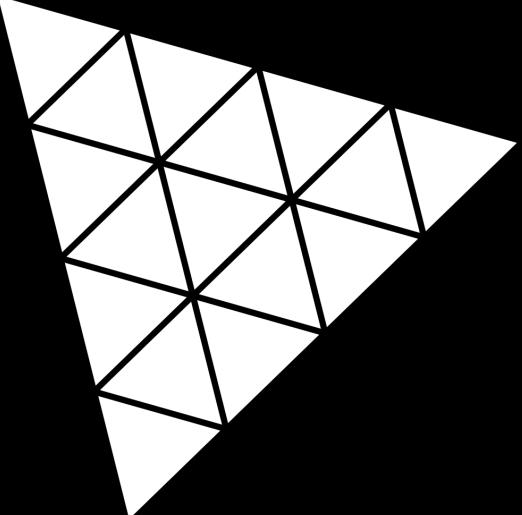


Desktop Applications:

- **Electron:** Allows building cross-platform desktop applications using HTML, CSS, and JavaScript.
- **NW.js:** A framework for building native applications with web technologies.

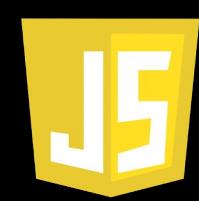


4. Applications of JavaScript



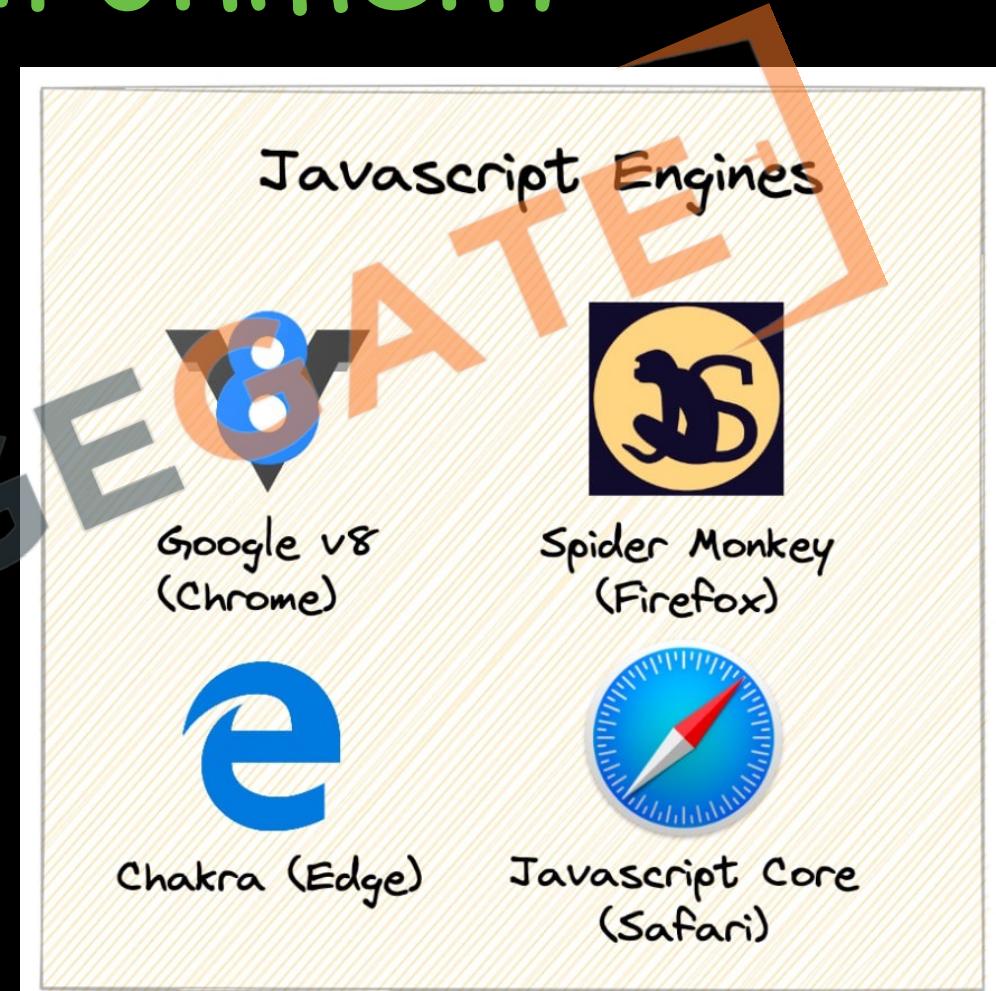
Cameras and Speakers:

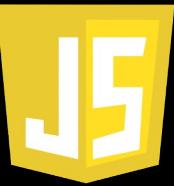
- **Three.js**: A library that makes **WebGL** - 3D programming for the web - easier to use.
- **WebRTC**: A technology that enables peer-to-peer audio, video, and data sharing.
- **Howler.js**: A JavaScript **audio library** for the modern web.



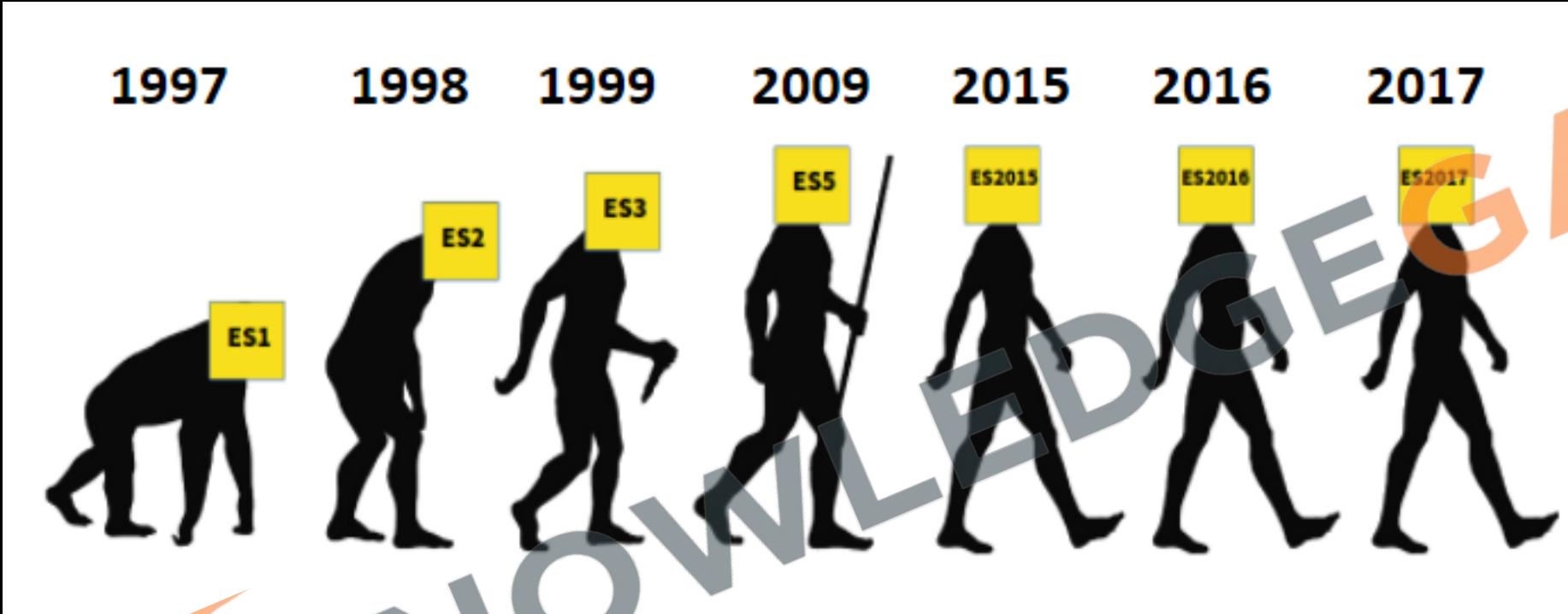
5. Runtime Environment

1. Provides infrastructure to execute JavaScript code.
2. Core: Includes a JavaScript engine (e.g., V8, SpiderMonkey).
3. Browser Environment: Offers APIs for DOM manipulation, events, and network requests.
4. Node.js: Extends JavaScript capabilities to server-side programming.
5. Asynchronous Support: Handles non-blocking operations with event loops, callbacks, and promises.





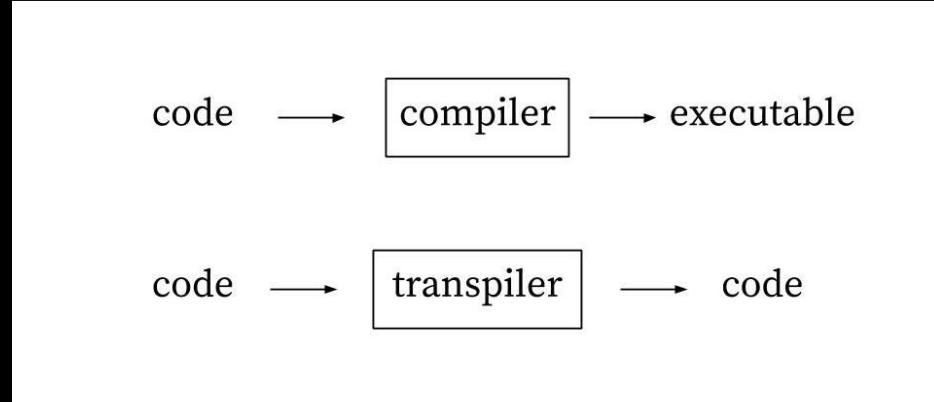
6. JavaScript vs ECMA



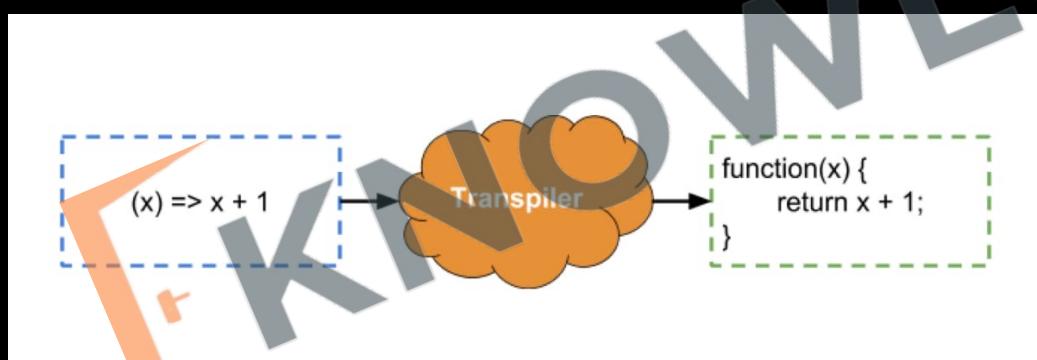
1. ECMAScript is the **standardized specification** developed by **ECMA International** that defines the **core features, syntax, and functionalities** of JavaScript and similar scripting languages.
2. JavaScript is the **actual language implementation**.

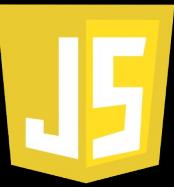


7. JavaScript vs TypeScript



1. JavaScript runs at the client side in the browser.
2. Coffee Script / TypeScript are transpiled to JavaScript.

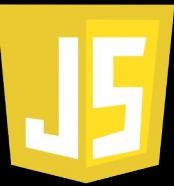




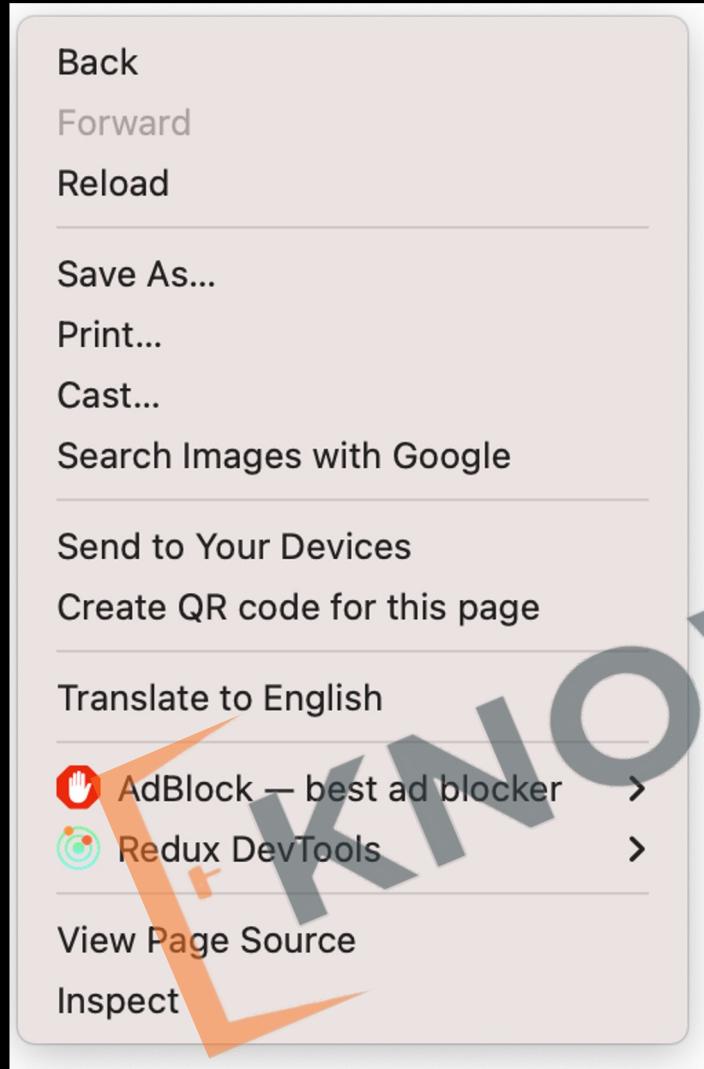
7. JavaScript vs TypeScript

Feature	JavaScript (JS)	TypeScript (TS)
Definition	A dynamic, high-level scripting language.	A statically typed superset of JavaScript.
Typing	Dynamically typed.	Statically typed with optional type annotations.
Compilation	Interpreted by browsers.	Transpiles to JavaScript before execution.
Error Detection	Errors detected at runtime.	Errors caught at compile-time.
Tooling Support	Basic tooling, less support for large-scale projects.	Enhanced tooling support with features like IntelliSense.
Learning Curve	Easier to learn for beginners.	Slightly steeper learning curve due to static typing.
Code Maintenance	Can be harder to maintain and debug in large codebases.	Easier to maintain and refactor due to static types.
Development Speed	Faster for small projects and prototyping.	Potentially slower initial development but saves time in the long run with fewer bugs.
Community and Usage	Widely used, especially in web development.	Growing rapidly, especially in large-scale applications.
Example Usage	<code>var x = 10;</code>	<code>let x: number = 10;</code>





8. JavaScript in Console (Inspect)



1. Allows **real-time editing** of **HTML/CSS/JS**
2. **Run Scripts**: Test code in console.
3. **Debug**: Locate and fix errors.
4. **Modify DOM**: Change webpage elements.
Errors: View error messages.



8. JavaScript in Console (Alert)

A screenshot of a Google search page in a browser. A large watermark reading "KNOWLEDGE" diagonally across the page is highlighted with orange arrows pointing towards the top right corner. An alert box is displayed in the center of the screen with the text "www.google.com says" and "JavaScript is Magical". In the bottom right corner of the browser window, the developer tools' "Console" tab is active, showing the command `alert('JavaScript is Magical');` and its execution result. The "Sources" tab is also visible in the tools.

Google

google.com

Gmail Images

www.google.com says
JavaScript is Magical

OK

Sources Console >>>

Filter

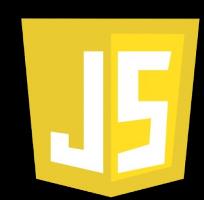
Default levels ▾ | No Issues

> alert('JavaScript is Magical');

>

Google Search I'm Feeling Lucky

Google offered in: हिन्दी ବାଂଗା ତେଲୁଗୁ ମରାଠୀ ତமିଳ் ଗୁଜରାତୀ କନ୍ନୁଡ଼ ମଲଯା�ളଂ ପੰਜਾਬੀ



8. JavaScript in Console (Math)

A screenshot of the browser developer tools' Console tab. The tab bar above shows 'Sources', 'Elements', 'Console' (which is underlined in blue), and 'More'. Below the tab bar are icons for refresh, zoom, and other developer tools, followed by 'top', a dropdown menu, and a 'Filter' input field. At the bottom left are 'Default levels' and 'No Issues' buttons. The main area of the console contains the following history of calculations:

```
> 2+3  
< 5  
> 5*9  
< 45  
> 8/4  
< 2  
> 99-9  
< 90  
>
```

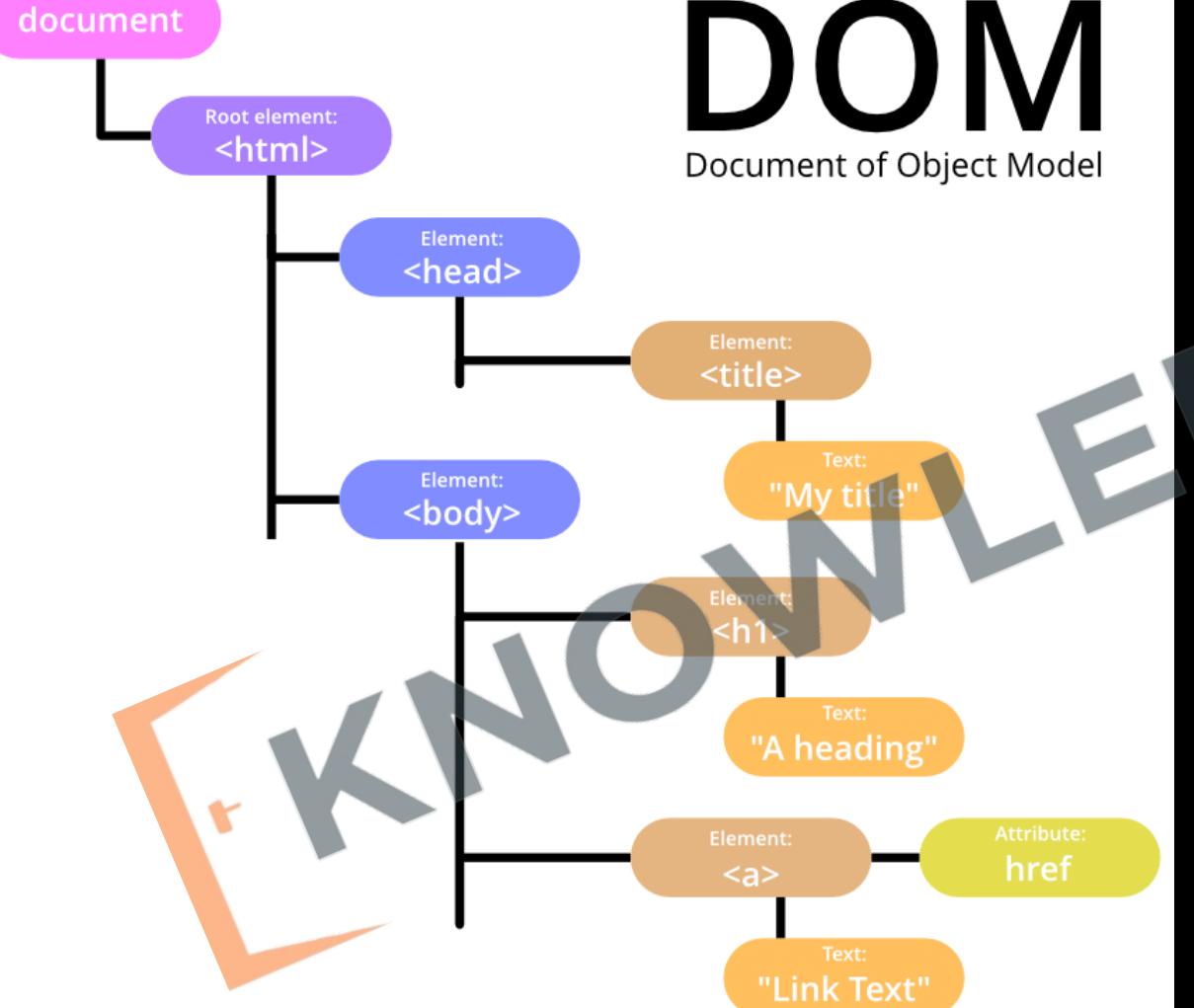
Console can be used
as a **Calculator**



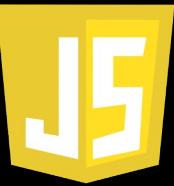
9. JavaScript in Webpage

```
<!DOCTYPE html>           Defines the HTML Version  
  
<html lang="en">          Parent of all HTML tags / Root element  
  
    <head>                  Parent of meta data tags  
        <title>My First Webpage</title> Title of the web page  
    </head>  
  
    <body>                  Parent of content tags  
        <h1>Hello World!</h1> Heading tag  
    </body>  
</html>
```

9. JavaScript in Webpage



- 1. Structure Understanding:** Helps in understanding the **hierarchical structure** of a webpage, crucial for applying **targeted CSS styles**.
- 2. Dynamic Styling:** Enables learning about **dynamic styling**, allowing for **real-time changes** and interactivity through CSS.

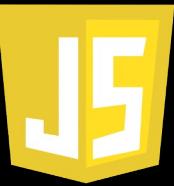


9. JavaScript in Webpage

(Script Tag)

1. **Embed Code:** Incorporates JavaScript into an HTML file, either **directly** or **via external files**.
2. **Placement:** Commonly placed in the **<head>** or just before the closing **</body>** tag to control when the script runs.
3. **External Files:** Use **src** attribute to link external JavaScript files, like **<script src="script.js"></script>**.
4. **Console Methods:** **log**, **warn**, **error**, **clear**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Basic JavaScript Example</title>
    <script>
      // This function changes the content of the paragraph with id="demo"
      function changeContent() {
        document.getElementById("demo").innerHTML =
          "Content changed by JavaScript!";
      }
    </script>
  </head>
  <body>
    <h1>Welcome to My Web Page</h1>
    <p id="demo">JavaScript can change HTML content.</p>
    <button onclick="changeContent()">Click me to change content</button>
  </body>
</html>
```

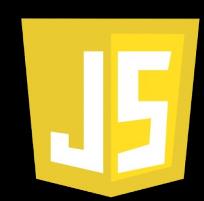


9. JavaScript in Webpage

(Comments)

```
/**  
 *  * This is a important comment  
 * ! This is warning comment  
 * ? This is a question comment  
 * TODO: This is a todo comment  
 */
```

1. Used to add notes in source code in JavaScript or CSS.
2. Not displayed on the web page
3. Syntax: /* comment here */
4. Helpful for code organization
5. Can be multi-line or single-line

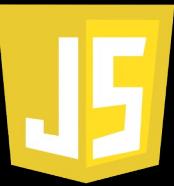


10. DOM Manipulation

A screenshot of a browser's developer tools console. The browser window shows a page titled "Welcome to KG Coding". The developer tools are open, with the "Console" tab selected. The console displays the following JavaScript code:

```
> document.body.innerHTML = '<b>Welcome to KG Coding</b>'  
< 'Welcome to KG Coding'  
> document.getElementsByTagName('b')[0].style.fontSize = '40px'  
< '40px'
```

-
1. Change HTML
 2. Change CSS
 3. Perform Actions



11 JavaScript with Node



1. **JavaScript Runtime:** Node.js is an **open-source**, **cross-platform** runtime environment for executing **JavaScript** code outside of a browser.
2. **NodeJs** is a **JavaScript** in a **different environment** means **Running JS on the server or any computer.**
3. **Built on Chrome's V8 Engine:** It runs on the **V8 engine**, which **compiles JavaScript directly to native machine code**, enhancing performance.
4. **V8** is written in **C++** for speed.
5. **V8 + Backend Features = NodeJs**



11 JavaScript with Node

JS math.js X

```
1 // Basic arithmetic operations and console output
2 console.log("5 + 3 =", 5 + 3);
3 console.log("10 - 6 =", 10 - 6);
4 console.log("7 + 2 =", 7 + 2);
5 console.log("20 - 4 =", 20 - 4);
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

● prashantjain@Mac-mini test % node math.js

5 + 3 = 8

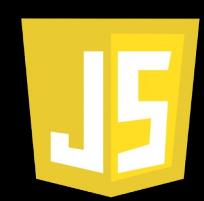
10 - 6 = 4

7 + 2 = 9

20 - 4 = 16

○ prashantjain@Mac-mini test % █

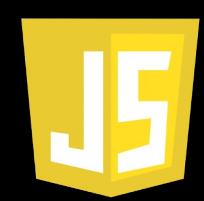
ACKNOWLEDGEMENT



Revision

1. History of JavaScript
2. What is JavaScript
3. Popularity of JavaScript
4. Applications of JavaScript
5. Runtime Environment
6. JavaScript vs ECMA
7. JavaScript vs TypeScript
8. JavaScript in Console
9. JavaScript in Webpage
10. DOM Manipulation
11. JavaScript with Node





KNOWLEDGEGATE

KG Coding



Some Other One shot Video Links:

- Complete Web Development
- Complete Backend
- Complete Java
- Complete C Programming
- One shot University Exam Series

<http://www.kgcoding.in/>

Our YouTube Channels

[KG Coding Android App](#)



[KG Coding](#)



[Knowledge GATE](#)



[KG Placement Prep](#)



[Sanchit Socket](#)



Core Concepts of JavaScript

- 12. Arithmetic Operators
- 13. Variables
- 14. Ways to Create Variables
- 15. Primitive Types
- 16. `typeof` Operator
- 17. Comparison Operators
- 18. `if-else`
- 19. Logical Operators
- 20. Functions
- 21. Loops
- 22. For Loop
- 23. Callbacks
- 24. Anonymous Functions as Values





12. Arithmetic Operators

Operators	Meaning	Example	Result
+	Addition	$4+2$	6
-	Subtraction	$4-2$	2
*	Multiplication	$4*2$	8
/	Division	$4/2$	2
%	Modulus operator to get remainder in integer division	$5\%2$	1

13 Variables

Variable is used to Store Data



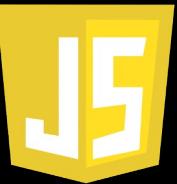
Variables are like containers
used for storing data values.



13 Variables (Syntax Rules)

```
1 // Defining a number variable  
2 let noOfStudents = 5;  
3 // Defining a String variable  
4 let welcomeMessage = "Hello Beta"
```

1. Can't use **keywords** or reserved words
2. Can't start with a **number**
3. No special characters other than **\$** and **_**
4. **=** is for **assignment**
5. **;** means end of **instruction**



13 Variables (Updating Values)

```
let noOfStudents = 5;  
noOfStudents = noOfStudents + 1;  
  
let money = 1;  
money += 5; // money = 6  
money -= 2; // money = 4  
money *= 3; // money = 12  
money /= 4; // money = 3  
money++; // money = 4
```

1. Do not need to use **let** again.
2. Syntax: **variable = variable + 1**
3. **Assignment** Operator is used **=**
4. Short Hand Assignment Operators:
+ =, - =, * =, / =, ++



14. Ways to Create Variables

var

var apple = a thing in a box named "apple"

apple = you can swap item later

let

let apple = a thing in a box named "apple" w/ protection shield

apple = OK! apple = NG
you can swap item only if you ask inside of the shield

const

const apple = a thing in LOCKED cage named "apple"

apple = NG
you can't swap item later.
apple.multiply(3) OK!
... but you can ask the item to change itself (if the item has method to do that)

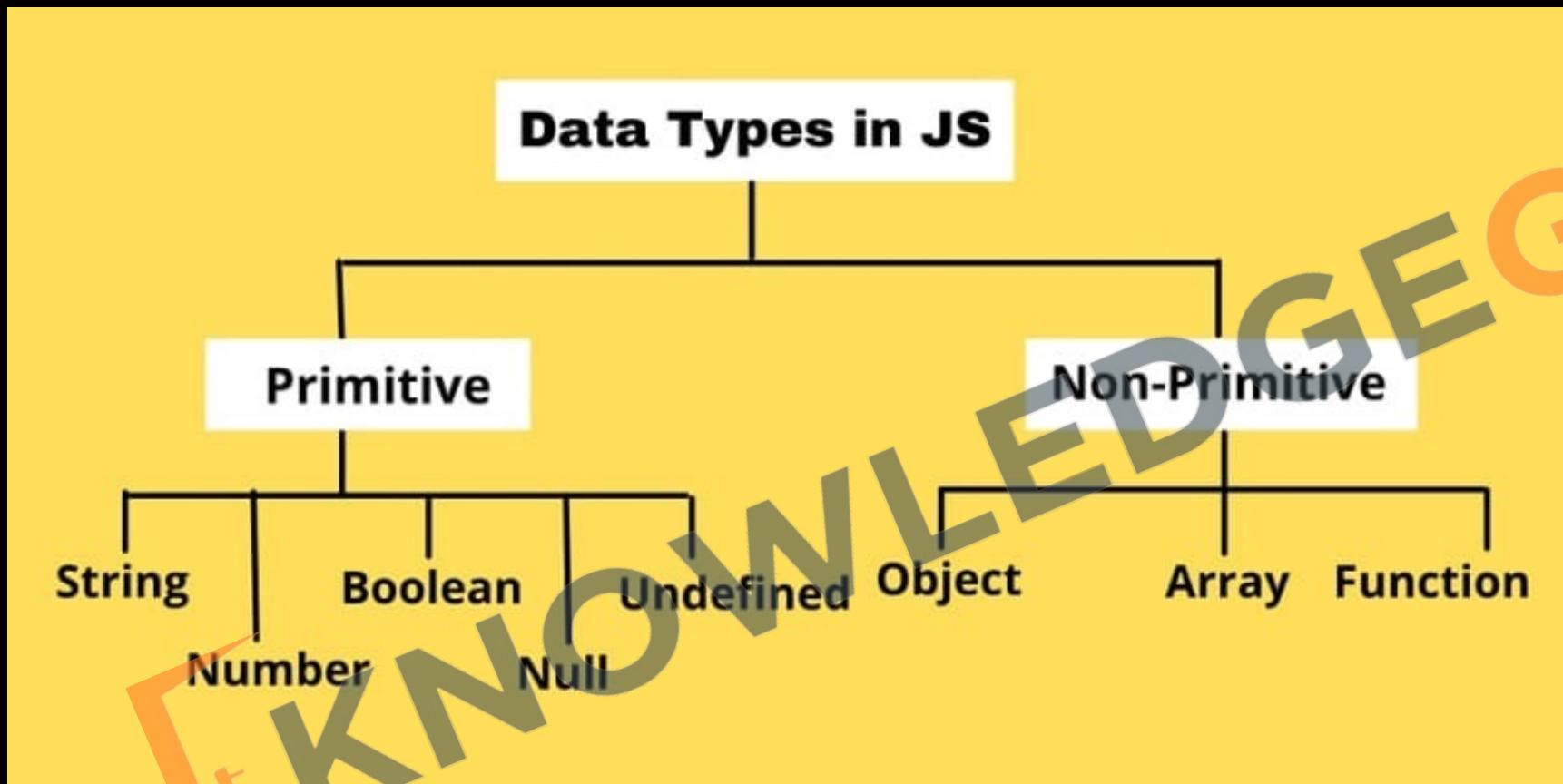


const let var

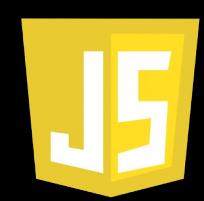
AGATE

15. Primitive Types

(What are Data Types)

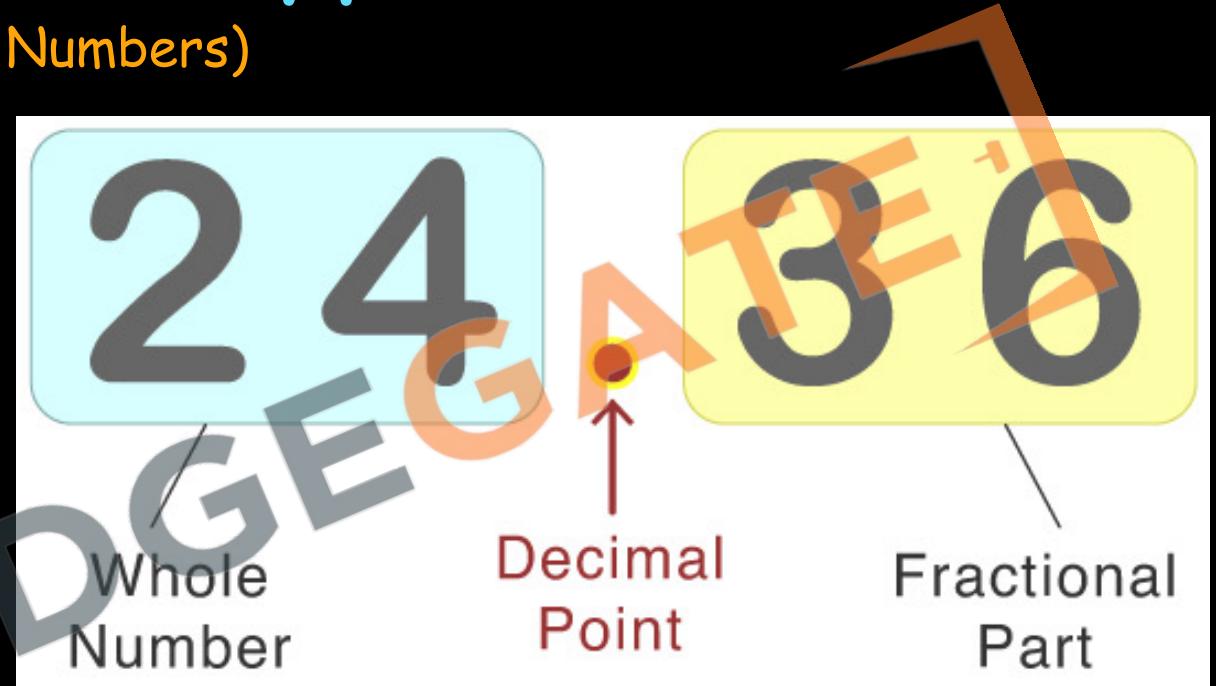
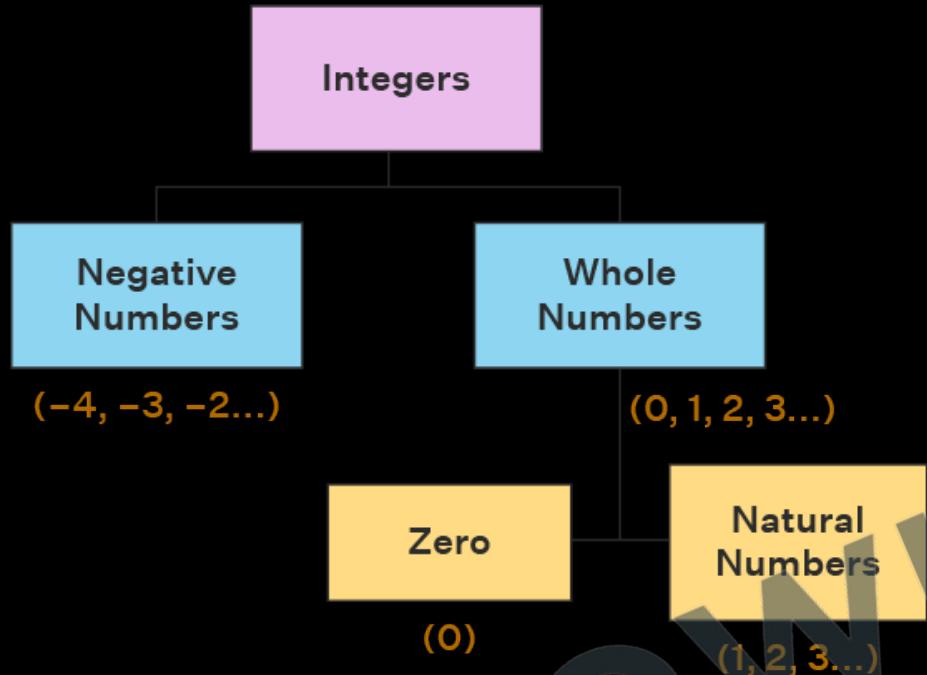


Primitive types in JavaScript are the most basic data types that are not objects and have no methods. They are immutable, meaning their values cannot be changed.



15. Primitive Types

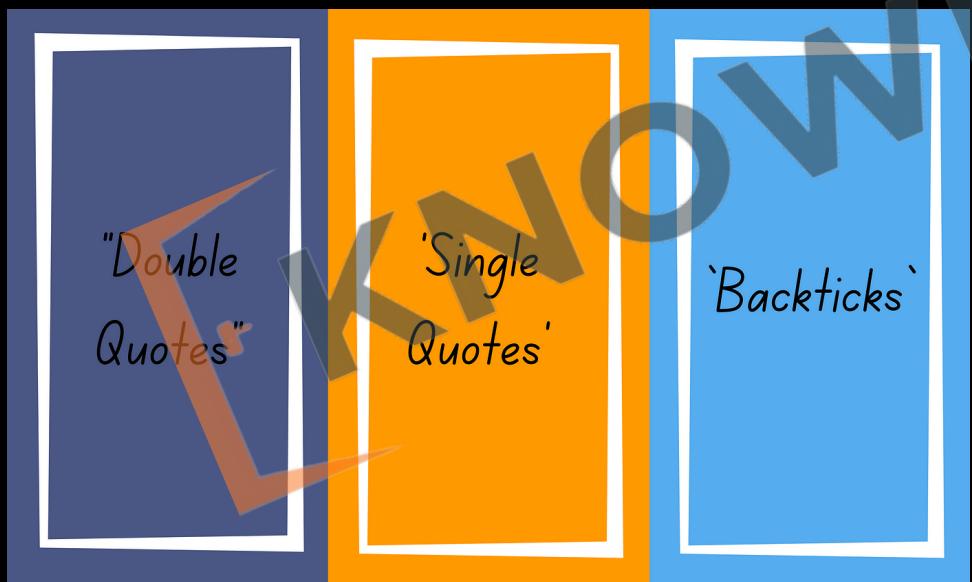
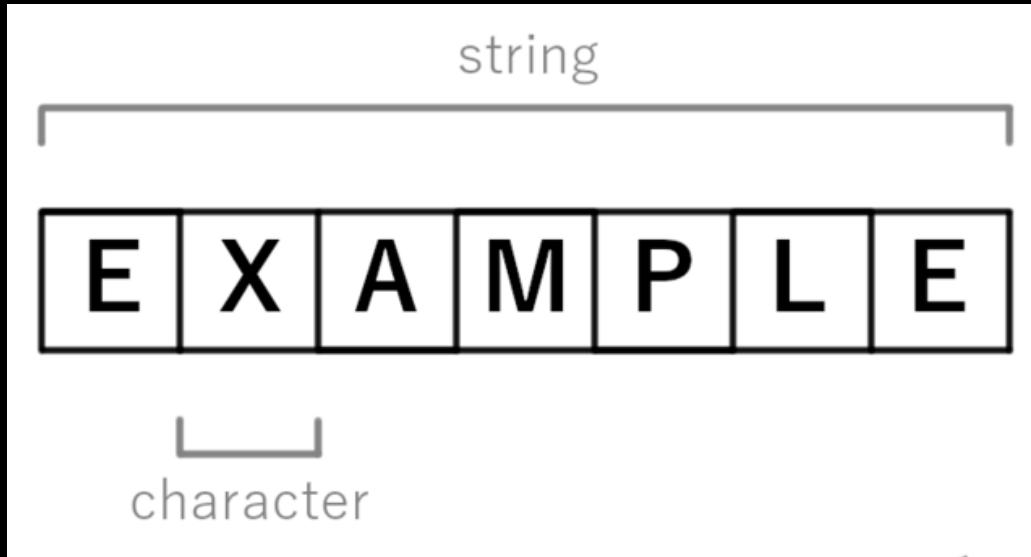
(Types of Numbers)





15. Primitive Types

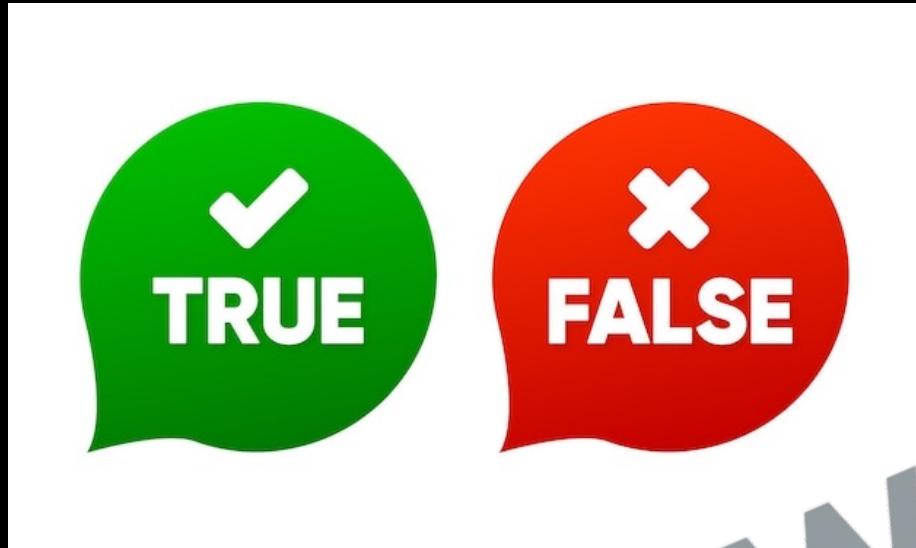
(Strings)



1. Strings hold **textual data**, anything from a single character to paragraph.
2. Strings can be defined using **single quotes '**', **double quotes " "**, or **backticks ` `**.
Backticks allow for template literals, which can include variables.
3. You can combine (**concatenate**) strings using the + operator. For example, "Hello" + " World" will produce "Hello World".

15. Primitive Types

(Boolean)



1. **Data Type:** Booleans are a basic data type in JavaScript.
2. **Two Values:** Can only be `true` or `false`.
3. '`true`' is a String not a Boolean



15. Primitive Types

(Null vs Undefined)

⇒ NULL vs UNDEFINED

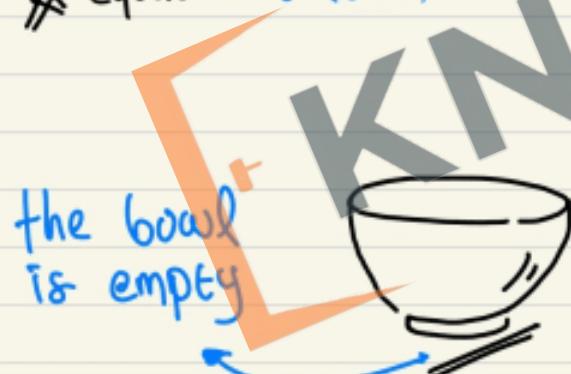
THOUGH BOTH ARE NULLISH & FALSEY VALUE

null !== undefined

no value, on purpose

the type is object

equal to 0 (zero)



declared, but not yet defined

the type is undefined

equal to NaN (Not A Number)

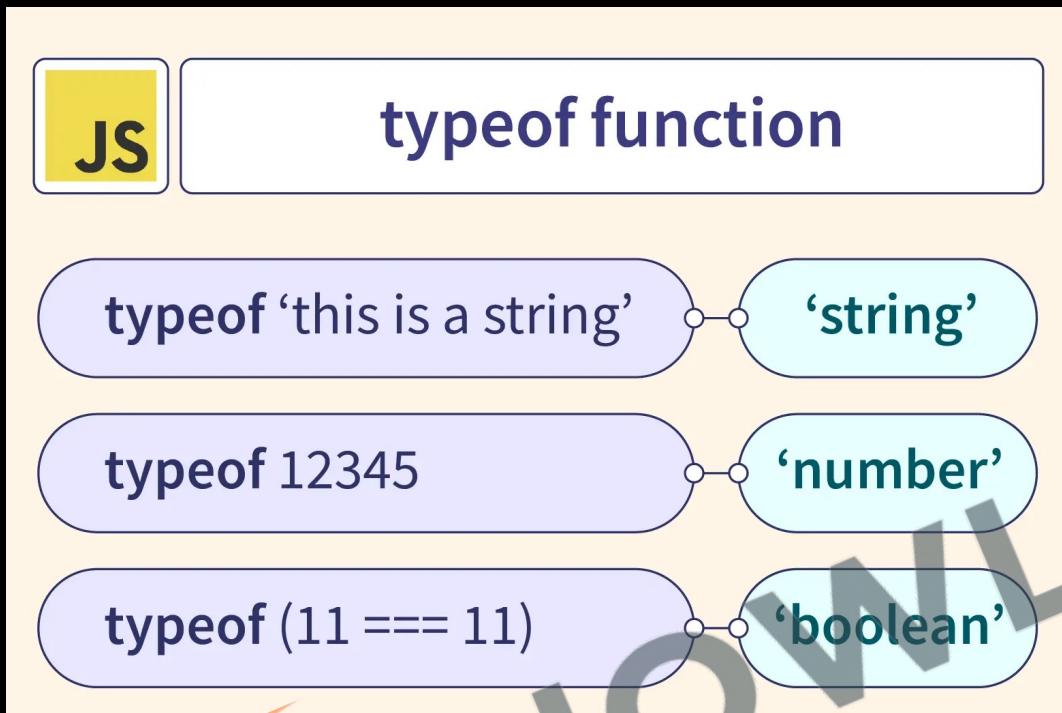
even, the bowl
is not exist



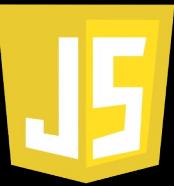
KNOWLEDGE GATE'



16. typeof Operator



1. **Check Type:** Tells you the data type of a variable.
2. **Syntax:** Use it like `typeof` variable.
3. **Common Types:** Returns "number," "string," "boolean," etc.



17. Comparison Operators

<, >, <=, >=, ==, !=

- Equality

- == Checks value equality.
- === Checks value and type equality.

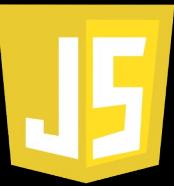
- Inequality

- != Checks value inequality.
- !== Checks value and type inequality.

- Relational

- > Greater than.
- < Less than.
- >= Greater than or equal to.
- <= Less than or equal to.

Order of comparison operators is less than arithmetic operators



18. if-else

```
// Use of if-else
let age = 18;
if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are a minor.");
}
```

if thirsty {

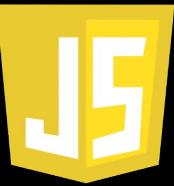


} else {



}

1. Syntax: Uses `if () {}` to check a condition.
2. What is `if`: Executes block if condition is `true`, skips if `false`.
3. What is `else`: Executes a block when the `if` condition is `false`.
4. Curly Braces can be omitted for single statements, but not recommended.
5. Use Variables: Can store conditions in variables for use in `if` statements.



18. if-else

```
// Use of if-else ladder
let score = 85;
if (score >= 90) {
    console.log("Grade: A");
} else if (score >= 80) {
    console.log("Grade: B");
} else if (score >= 70) {
    console.log("Grade: C");
} else if (score >= 60) {
    console.log("Grade: D");
} else {
    console.log("Grade: F");
}
```

```
// Use of nested if-else
let number = 10;
if (number > 0) {
    if (number % 2 === 0) {
        console.log("The number is positive and even.");
    } else {
        console.log("The number is positive and odd.");
    }
} else if (number < 0) {
    console.log("The number is negative.");
} else {
    console.log("The number is zero.");
}
```

If-else Ladder: Multiple `if` and `else if` blocks; only one executes.

19. Logical Operators

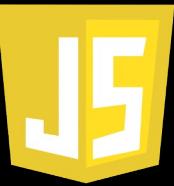


AND

OR

not

1. Types: **&&** (AND), **||** (OR), **!** (NOT)
2. AND (**&&**): All conditions **must be true** for the result to be true.
3. OR (**||**): Only **one condition** must be true for the result to be true.
4. NOT (**!**): **Inverts** the Boolean value of a condition.
5. Lower Priority than **Math** and **Comparison operators**



19. Logical Operators

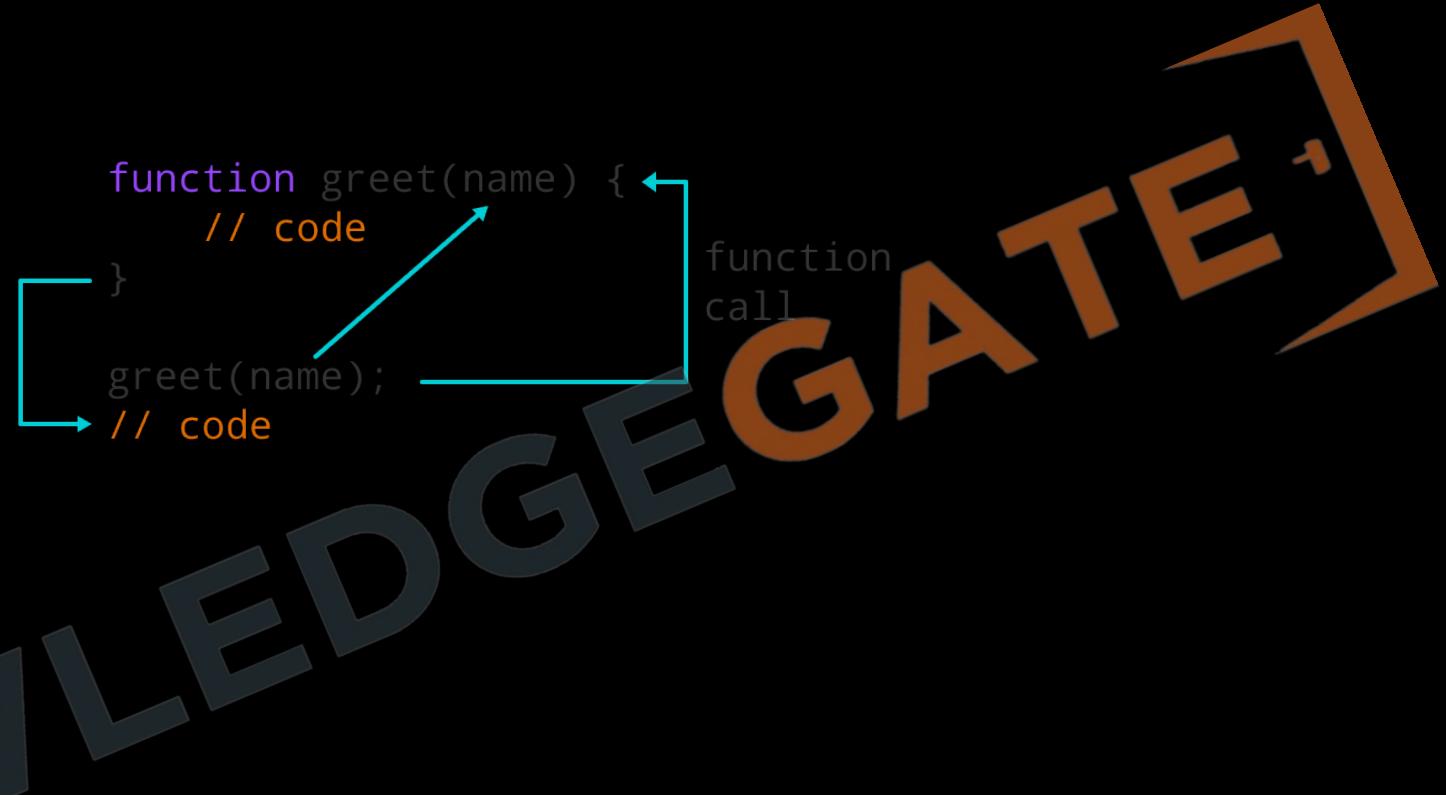
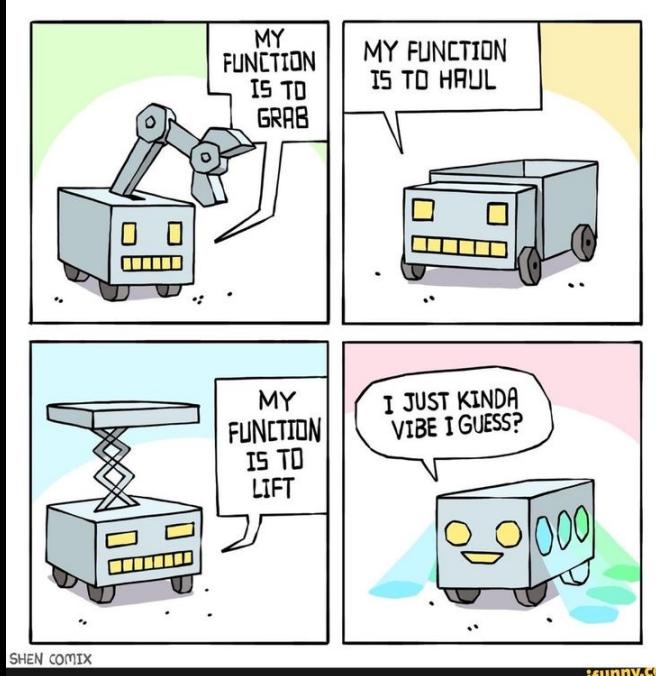
```
// Use of || OR
let day = "Saturday";
if (day === "Saturday" || day === "Sunday") {
  console.log("It's a weekend!");
} else {
  console.log("It's a weekday.");
}
```

```
// Use of ! NOT
let isRaining = false;
if (!isRaining) {
  console.log("You don't need an umbrella.");
} else {
  console.log("You need an umbrella.");
}
```

```
// Use of && AND
let age = 25;
let hasDrivingLicense = true;
if (age >= 18 && hasDrivingLicense) {
  console.log("You can drive.");
} else {
  console.log("You cannot drive.");
}
```



20. Functions



1. **Definition:** Blocks of reusable code.
2. **DRY Principle:** "Don't Repeat Yourself" it Encourages code reusability.
3. **Usage:** Organizes code and performs specific tasks.
4. **Naming Rules:** Same as variable names: camelCase
5. **Example:** "Beta Gas band kar de"

20. Functions (Return Statement)



1. Sends a value back from a function.
2. Example: "Ek glass paani laao"
3. What Can Be Returned: Value, variable, calculation, etc.
4. Return ends the function immediately.
5. Function calls make code jump around.
6. Prefer returning values over using global variables.

20. Functions (Parameters)



Parameter

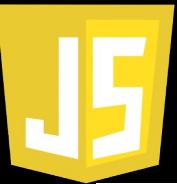


Function

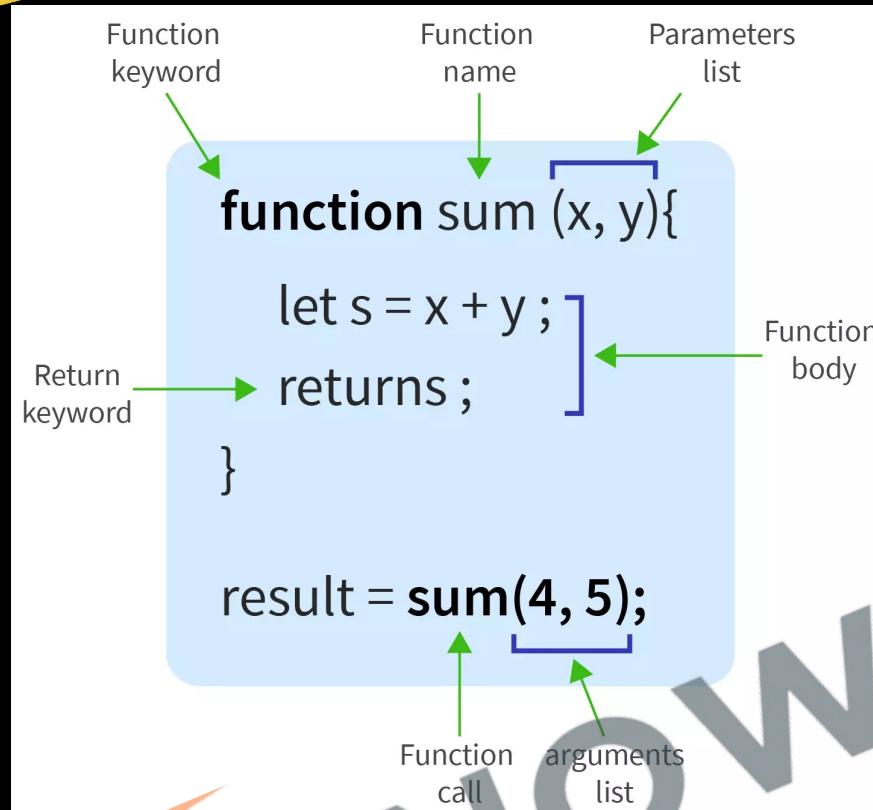


Return

1. Input values that a function takes.
2. Parameters put value into function, while return gets value out.
3. Example: "Ek packet dahi laao"
4. Naming Convention: Same as variable names.
5. Parameter vs Argument
6. Examples: alert, Math.round, console.log are functions we have already used
7. Multiple Parameters: Functions can take more than one.
8. Default Value: Can set a default value for a parameter.

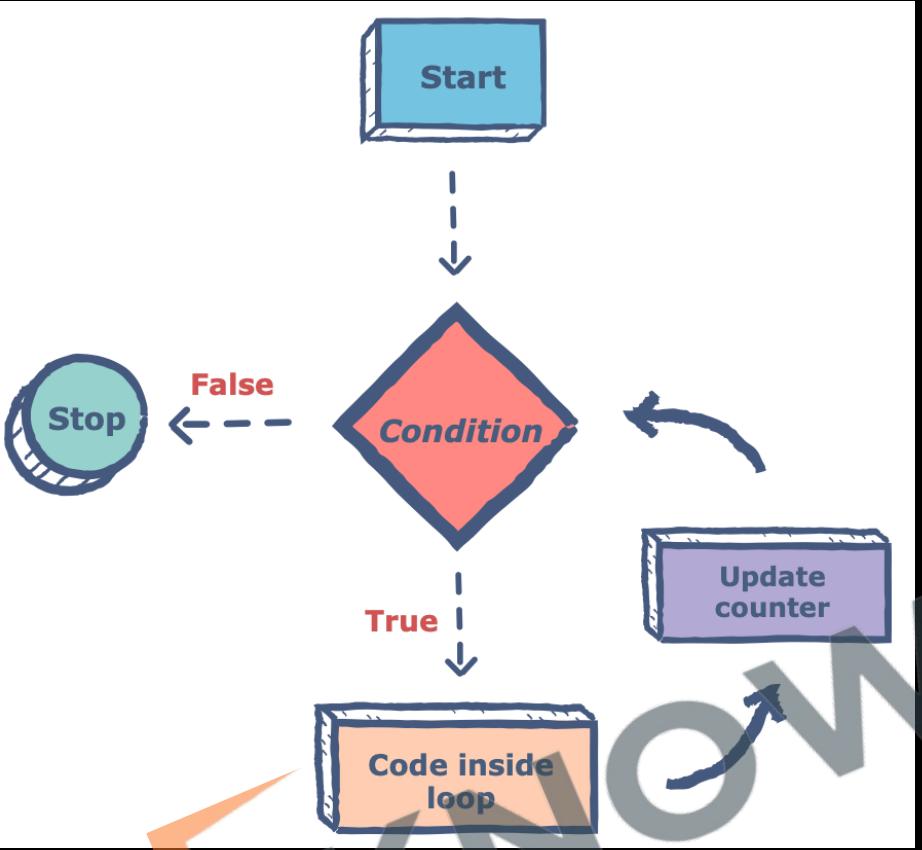


20. Functions (Syntax)



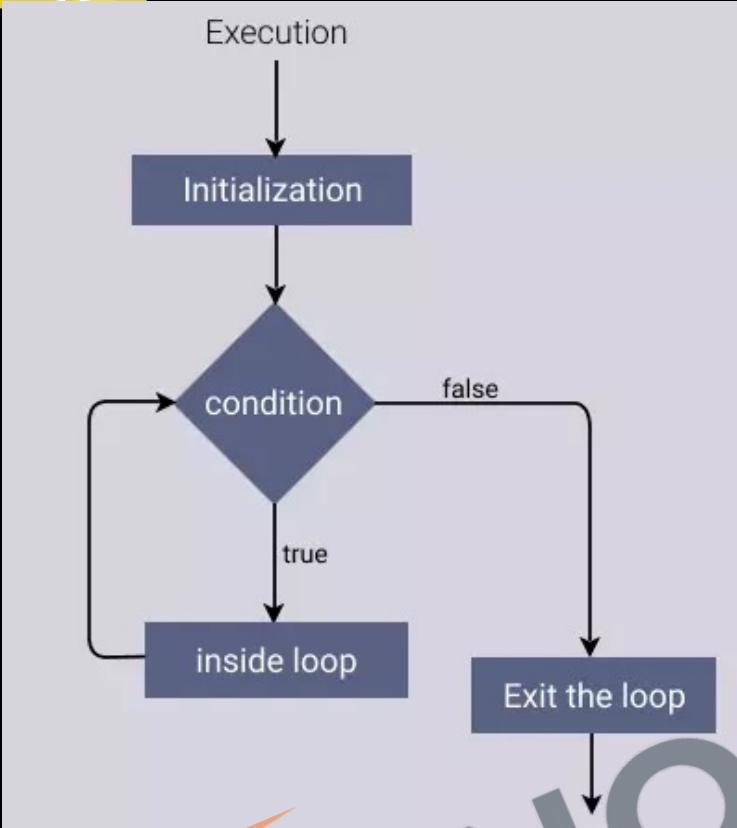
1. Use **function keyword** to declare.
2. Follows same rules as **variable names**.
3. Use **()** to contain parameters.
4. Invoke by using the **function name** followed by **()**.
5. Fundamental for **code organization** and **reusability**.

21. What is a Loop?



1. Code that runs **multiple times** based on a **condition**.
2. Loops also **alter the flow of execution**, similar to functions.
 - Functions: **Reusable** blocks of code.
 - Loops: **Repeated execution** of code.
3. Loops automate **repetitive tasks**.
4. Types of Loops: **for, while, do-while**.
5. Iterations: Number of **times** the loop runs.

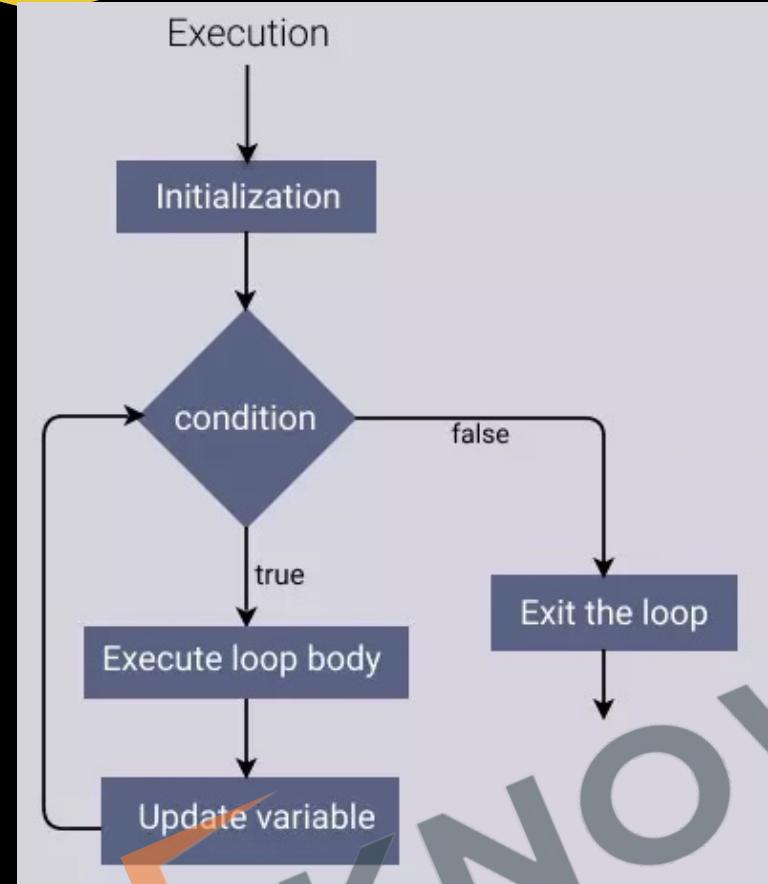
21. While Loop



```
while (condition) {  
    // Body of the loop  
}
```

1. Iterations: Number of **times** the loop **runs**.
2. Used for **non-standard** conditions.
3. Repeating a block of code **while** a condition is **true**.
4. Remember: Always include an update to **avoid infinite loops**.

22. For Loop



```
for (initialisation; condition; update) {  
    // Body of the loop  
}
```

1. Standard loop for running code multiple times.
2. Generally preferred for counting iterations.



23. Callbacks

```
// Define a callback function
function greeting(name) {
  console.log('Hello, ' + name);
}

// Define a function that takes a callback
function processUserInput(callback) {
  var name = prompt('Please enter your name.');
  callback(name);
}

// Call the function with the callback
processUserInput(greeting);
```

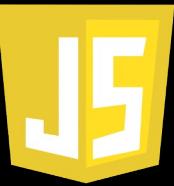
1. A **callback** is a **function** passed as an **argument** to another **function**, which is then **invoked inside the outer function** to complete some kind of routine or action.
2. **Usage:** Callbacks are **commonly used in asynchronous programming** to **execute code after an asynchronous operation has completed**.



24. Anonymous Functions as Values

```
1 // syntax
2 (function() {
3     // function body
4 });
5
6 // Example as a callback
7 setTimeout(function() {
8     console.log("This is anonymous");
9 }, 1000);
10
11 // Assigned to a variable
12 const add = function(a, b) {
13     return a + b;
14 };
15
16 console.log(add(2, 3)); // Outputs: 5
```

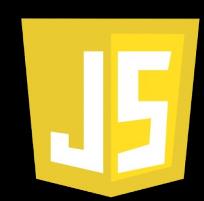
1. Anonymous functions are **functions** without a name.
2. They are often used as arguments to other **functions** or assigned to a **variable**.
3. Useful for **creating function scopes** and **avoiding global variables**.



Revision

- 12. Arithmetic Operators
- 13. Variables
- 14. Ways to Create Variables
- 15. Primitive Types
- 16. `typeof` Operator
- 17. Comparison Operators
- 18. `if-else`
- 19. Logical Operators
- 20. Functions
- 21. Loops
- 22. For Loop
- 23. Callbacks
- 24. Anonymous Functions as Values





KNOWLEDGEGATE

KG Coding



Some Other One shot Video Links:

- Complete Web Development
- Complete Backend
- Complete Java
- Complete C Programming
- One shot University Exam Series

<http://www.kgcoding.in/>

Our YouTube Channels

[KG Coding Android App](#)



[KG Coding](#)



[Knowledge GATE](#)



[KG Placement Prep](#)



[Sanchit Socket](#)



Advanced JavaScript

25. Object Oriented Language

26. Working with Objects

27. Reference Types

28. Arrays

29. for-each Loop

30. Array Methods

31. Arrow Functions

32. De-structuring

33. Spread & Rest Operator

34. Promises

35. Fetch API

36. Async / Await



24. Object Oriented Language



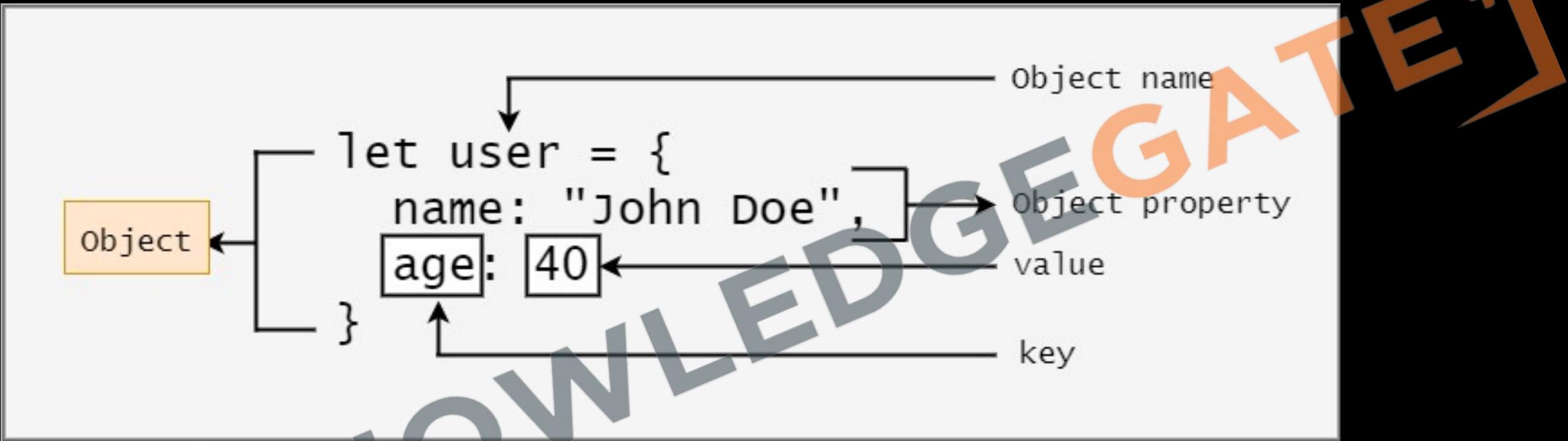
```
let product = {  
    company: 'Mango',  
    item_name: 'Cotton striped t-shirt',  
    price: 861  
};
```

1. Groups multiple values together in key-value pairs.
2. How to Define: Use {} to enclose properties.
3. Example: product {name, price}
4. Dot Notation: Use . operator to access values.
5. Key Benefit: Organizes related data under a single name.



24. Object Oriented Language

(Object Syntax)



1. **Basic Structure:** Uses `{}` to enclose **data**.
2. **Rules:** **Property** and **value** separated by a `:`
3. **Comma:** Separates different **property-value** pairs.
4. **Example:** `{ name: "Laptop", price: 1000 }`

25. Working with Objects

(Accessing Objects)

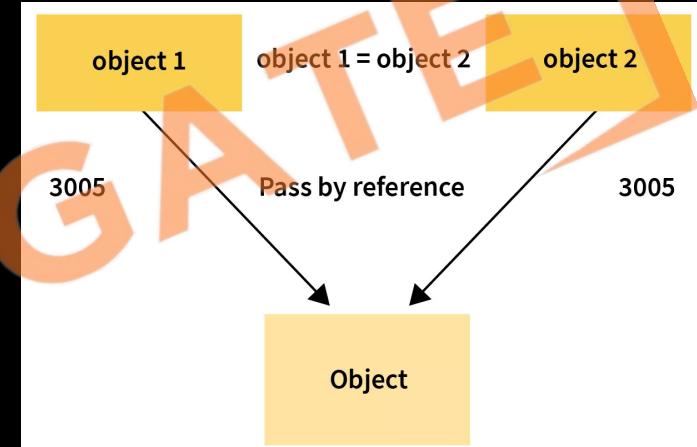
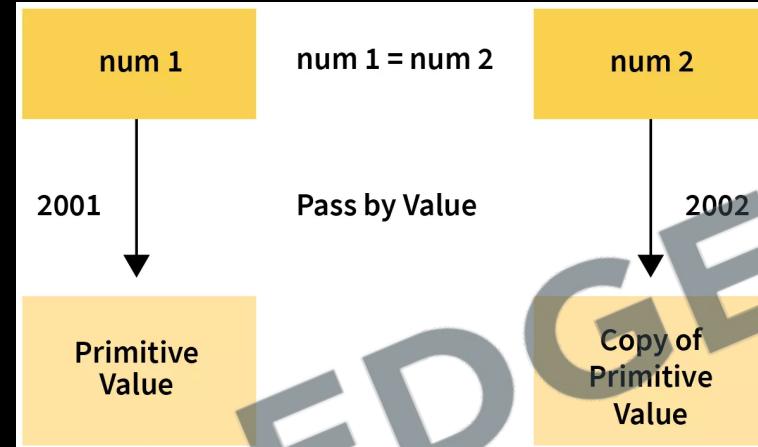
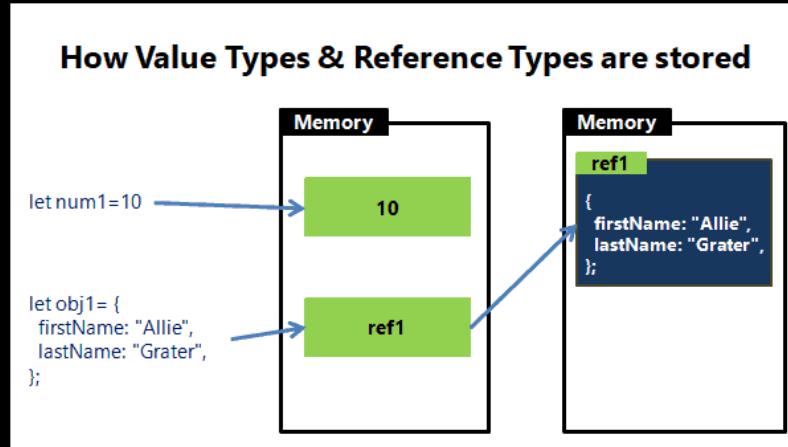


1. Dot Notation: Access **properties** using **. Operator** like `product.price`
2. Bracket Notation: Useful for **properties with special characters** `product["nick-name"]`. Variables can be used to access properties
3. `typeof` returns **object**.
4. Values can be **added** or **removed** to an object
5. Delete Values using **delete**

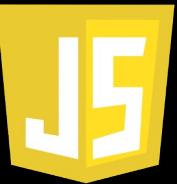


26. Reference Types

(Primitive vs Reference Types)

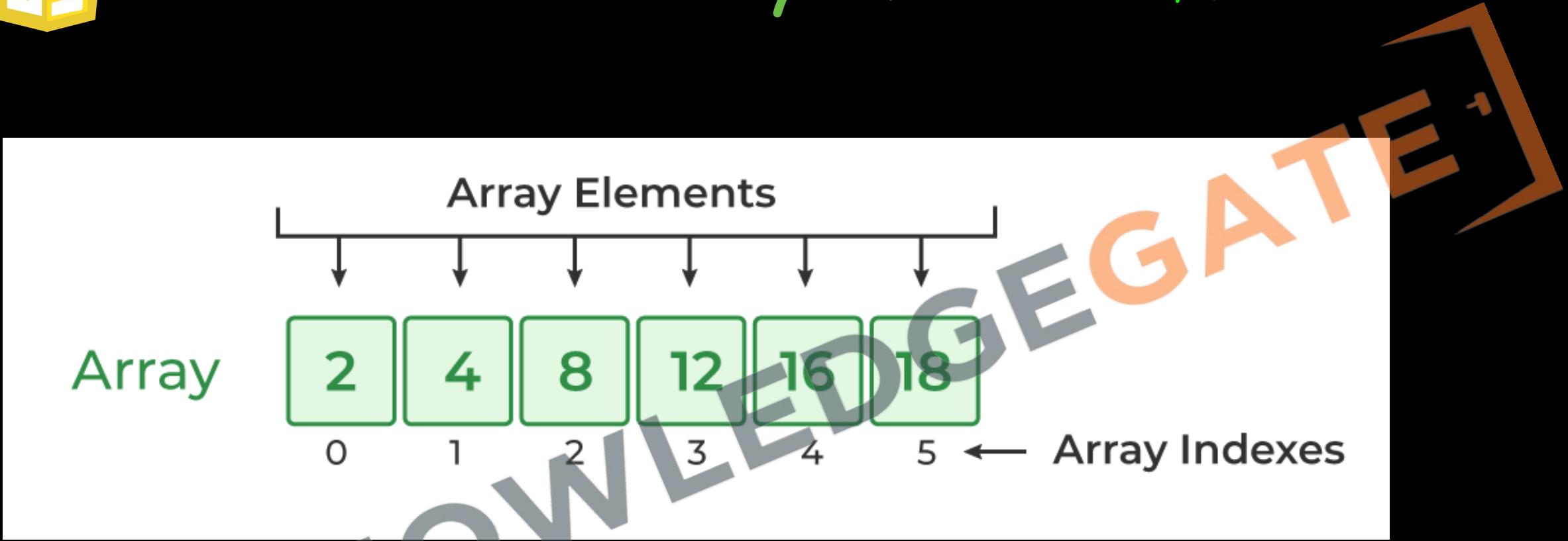


1. Objects work based on **references**, not actual data.
2. **Copying** an object copies the reference, not the actual object.
3. When comparing with `==`, you're comparing references, not content.
4. Changes to one reference **affects all copies**.



27. Arrays

(What is an Array?)



1. An Array is just a **list** of values.
2. **Index:** Starts with **0**.
3. Arrays are used for **storing multiple values** in a single variable.



27. Array (Syntax & Values)

```
let myArray = [1, 'KG Coding', null, true,  
 {likes: '1 Million'}];
```

1. Use [] to create a new array, [] brackets enclose list of values
2. Arrays can be saved to a variable.
3. Accessing Values: Use [] with index.
4. Syntax Rules:
 - Brackets start and end the array.
 - Values separated by commas.
 - Can span multiple lines.
5. Arrays can hold any value, including arrays.
6. typeof operator on Array Returns Object.



28. for-each Loop

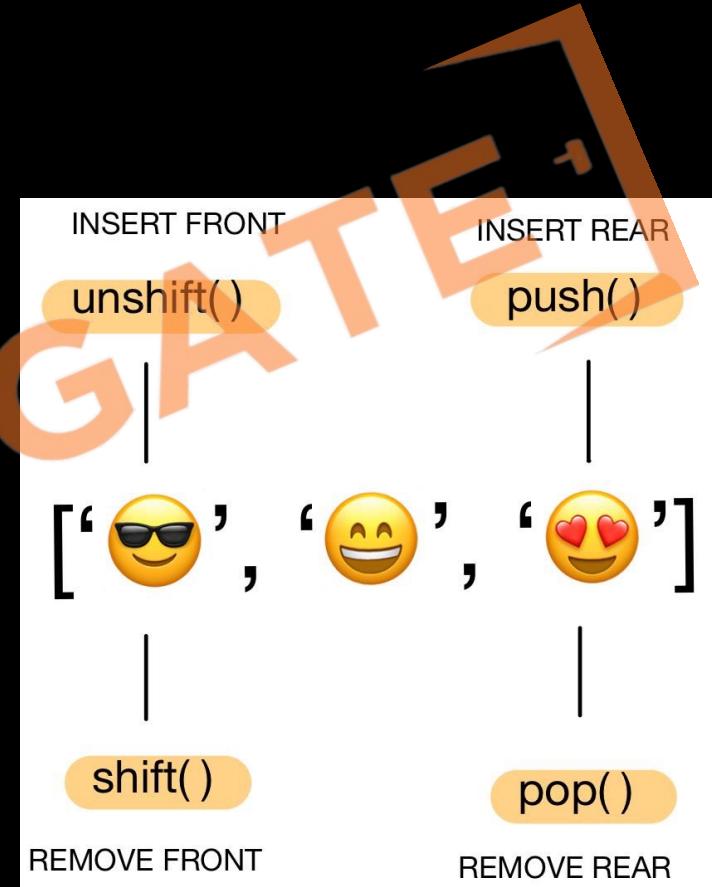
```
let foods = ['bread', 'rice', 'meat', 'pizza'];

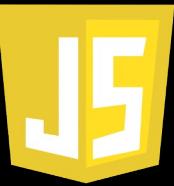
foods.forEach(function(food) {
  console.log(food);
})
```

- 1. A method for **array iteration**, often preferred for readability.
- 2. Parameters: One for **item**, optional second for **index**.
- 3. Using **return** is similar to **continue** in traditional loops.
- 4. Not straightforward to **break** out of a forEach loop.
- 5. When you need to perform an action on each array element and don't need to break early.

29. Array Methods

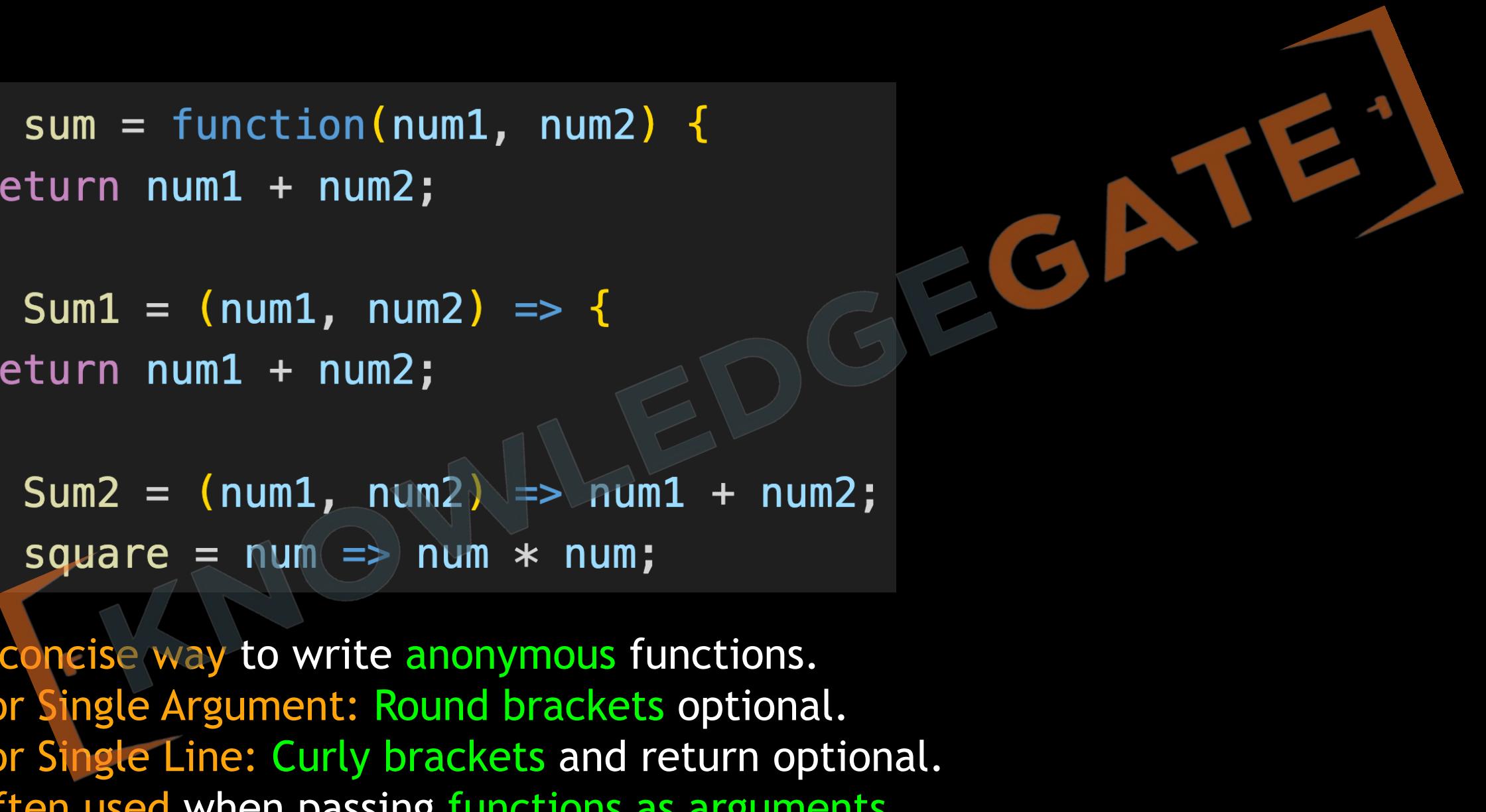
1. `Array.isArray()` checks if a variable is an array.
2. `Length` property holds the size of the array.
3. Common Methods:
 - `push/pop`: Add or remove to end.
 - `shift/unshift`: Add or remove from front.
 - `splice`: Add or remove elements.
 - `toString`: Convert to string.
 - `sort`: Sort elements.
 - `valueOf`: Get array itself.
4. Arrays also use `reference` like objects.
5. De-structuring also `works` for Arrays.

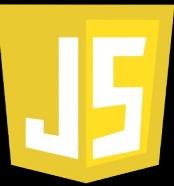




30. Arrow Functions

```
let sum = function(num1, num2) {  
    return num1 + num2;  
}  
  
let Sum1 = (num1, num2) => {  
    return num1 + num2;  
}  
  
let Sum2 = (num1, num2) => num1 + num2;  
let square = num => num * num;
```

- 
1. A concise way to write anonymous functions.
 2. For Single Argument: Round brackets optional.
 3. For Single Line: Curly brackets and return optional.
 4. Often used when passing functions as arguments.



30. Arrow Functions

(Anonymous & Arrow Callbacks)

```
// Anonymous Callback Function  
fetchData(function(data) {  
  console.log('Received:', data);  
});  
  
// Arrow Function as Callback  
fetchData(data => {  
  console.log('Received:', data);  
});
```

1. Instead of naming the callback function, you can **define it directly** within the argument list.
2. ES6 **arrow functions** can also be used as **callbacks** for a more concise syntax.



32. De-structuring

```
let product = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    price: 861  
};  
  
// Destructuring  
let company = product.company  
// is same as  
let { company } = product;
```

```
// Property shorthand  
let price = 861;  
let product = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    price: price  
};  
  
// is same as  
let product1 = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    price  
};
```

```
// Method shorthand  
let product = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    displayPrice: function() {  
        return `$$\{this.price.toFixed(2)\}`;  
    }  
};  
  
// is same as  
let product1 = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    displayPrice() {  
        return `$$\{this.price.toFixed(2)\}`;  
    }  
};
```

1. De-structuring: Extract **properties** from objects easily.
2. We can extract more than one property at once.
3. Shorthand Property: **{message: message}** simplifies to just **message**.
4. Shorthand Method: Define **methods** directly inside the object without the **function** keyword.



33. Spread & Rest Operator

(Spread)

```
1 // Array Expansion
2 const arr1 = [1, 2, 3];
3 const arr2 = [...arr1]; // [1, 2, 3]
4 // [1, 2, 3, 4, 5]
5 const arr3 = [...arr1, 4, 5];
6
7 // Object Expansion
8 const obj1 = { a: 1, b: 2 };
9 // { a: 1, b: 2, c: 3 }
10 const obj2 = { ...obj1, c: 3 };
11
12 // Function Arguments
13 function sum(a, b, c) {
14     return a + b + c;
15 }
16 const numbers = [1, 2, 3];
17 console.log(sum(...numbers)); // 6
```

1. Represented by three dots (...), the spread operator is used to expand elements of an iterable (like an array or string) into individual elements.
2. Useful for copying arrays and objects without modifying the original.
3. Ensures immutability in functions where modification of inputs is not desired.



33. Spread & Rest Operator

(Rest)

```
1 // Function Parameters
2 function sum(...numbers) {
3   return numbers.reduce((acc, curr) => acc + curr, 0);
4 }
5 console.log(sum(1, 2, 3, 4)); // 10
6
7 // Array Destructuring
8 const [first, second, ...rest] = [1, 2, 3, 4, 5];
9 console.log(rest); // [3, 4, 5]
10
11 // Object destructuring
12 const { a, b, ...rest } = { a: 1, b: 2, c: 3, d: 4 };
13 console.log(rest); // { c: 3, d: 4 }
```

- Used to collect the remaining elements of an array after extracting some elements.
- Used to collect the remaining properties of an object after extracting some properties.

1. Represented by three dots (...), the rest operator is used to collect multiple elements into a single array or object.
2. Allows a function to accept an indefinite number of arguments as an array.



34. Promises

(Need: Callback Hell)

```
function step1(callback) {  
    setTimeout(() => {  
        console.log('Step 1');  
        callback();  
    }, 1000);  
}
```

```
function step2(callback) {  
    setTimeout(() => {  
        console.log('Step 2');  
        callback();  
    }, 1000);  
}
```

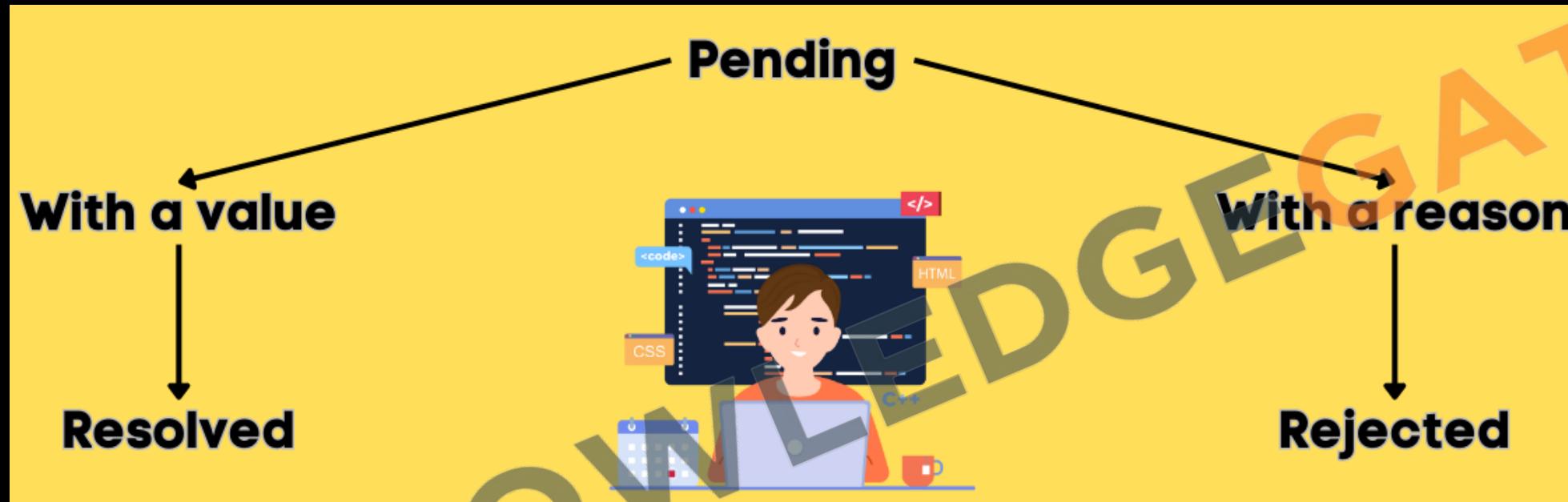
```
function step3(callback) {  
    setTimeout(() => {  
        console.log('Step 3');  
        callback();  
    }, 1000);  
}
```

```
step1(() => {  
    step2(() => {  
        step3(() => {  
            console.log('All steps completed');  
        });  
    });  
});  
});
```

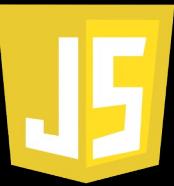
When multiple asynchronous operations need to be performed in sequence, callbacks can lead to deeply nested and hard-to-read code, often referred to as “callback hell.”

34. Promises

(States of Promise)



1. Definition: A promise is an object representing the eventual completion or failure of an asynchronous operation.
2. States of a Promise:
 - Pending: Initial state, neither fulfilled nor rejected.
 - Fulfilled: Operation completed successfully.
 - Rejected: Operation failed.

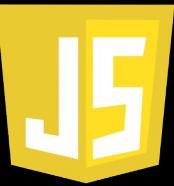


34. Promises

(Creation of Promise)

```
// Creating a Promise
let promise = new Promise(resolve, reject) => {
  // Asynchronous operation
  if (result()) {
    | resolve('Success');
  } else {
    | reject('Error');
  }
});
```

Promises are created using the `Promise` constructor, which takes an `executor` function with two arguments: `resolve` and `reject`.



34. Promises

(Handling of Promise)

```
// Handling a Promise: handle value
promise.then(value => {
  console.log(value); // 'Success'
});

// Handling a Promise: handle rejection
promise.catch(error => {
  console.error(error); // 'Error'
};

/* Handling a promise: Executes a block of
code regardless of the promise's outcome.*/
promise.finally(() => {
  console.log('Operation completed');
});
```

Promises have `then`, `catch`, and `finally` methods for handling the results of the asynchronous operation.

- `then()`: Used to handle fulfillment.
- `catch()`: Used to handle rejection.
- `finally()`: Executes a block of code regardless of the promise's outcome.



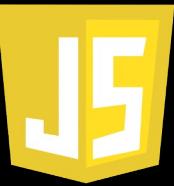
34. Promises

(Solving Callback Hell)

```
function step1() {  
  return new Promise((resolve) => {  
    setTimeout(() => {  
      console.log('Step 1');  
      resolve();  
    }, 1000);  
  });  
  
function step2() {  
  return new Promise((resolve) => {  
    setTimeout(() => {  
      console.log('Step 2');  
      resolve();  
    }, 1000);  
  });  
}
```

```
function step3() {  
  return new Promise((resolve) => {  
    setTimeout(() => {  
      console.log('Step 3');  
      resolve();  
    }, 1000);  
  });  
  
step1()  
.then(() => step2())  
.then(() => step3())  
.then(() => {  
  console.log('All steps completed');  
});
```

In this version, each step **returns** a Promise that resolves after a timeout. The steps are **chained** together using `.then()`, making the code more **readable** and easier to maintain.



35. Fetch API

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok ' + response.statusText);
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

- The Fetch API provides a modern way to make HTTP requests in JavaScript.
- It is a promise-based API, making it easier to handle asynchronous requests.

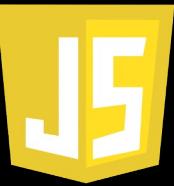


36. Async / Await

```
// using async  
async function myFunction() {  
  return 'Hello';  
}  
  
// using await  
async function fetchData() {  
  let response = await fetch('https://api.example.com/data');  
  let data = await response.json();  
  return data;  
}
```

KNOWLEDGE GATE

1. Syntax Sugar for Promises: `async/await` is built **on top of promises**, providing a cleaner and more readable way to work with asynchronous code.
2. Defining Async Functions: An `async` function is declared using the `async` keyword **before the function** definition. This function always returns a promise.
3. The `await` keyword is used to **pause the execution** of an `async` function until a **promise** is resolved. It can **only** be used inside an `async` function.

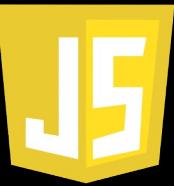


36. Async / Await

(Handling Exceptions)

```
async function getData() {  
  try {  
    let response = await fetch('https://api.example.com/data');  
    let data = await response.json();  
    return data;  
  } catch (error) {  
    console.error('Error:', error);  
  }  
}
```

Errors in **async** functions can be handled using **try...catch** blocks, making **error management straightforward** and **consistent with synchronous code**.



36. Async / Await

(Fetch API using `async/await`)

```
async function fetchData(url) {  
  try {  
    const response = await fetch(url);  
    if (!response.ok) {  
      throw new Error('Network response was not ok ' + response.statusText);  
    }  
    const data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.log(error);  
  }  
}  
  
fetchData('https://jsonplaceholder.typicode.com/posts');
```



Revision

24. Object Oriented Language

25. Working with Objects

26. Reference Types

27. Arrays

28. for-each Loop

29. Array Methods

30. Arrow Functions

31. De-structuring

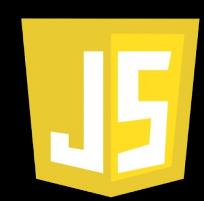
32. Spread & Rest Operator

33. Promises

34. Fetch API

35. Async / Await





KNOWLEDGEGATE