The typeof Keyword in Javascript

The **typeof** keyword in JavaScript is a **unary operator** used to **determine the data type** of its single operand, returning the result as a string. It's especially useful in JavaScript, a dynamically typed language, for type checking and data validation at runtime.

Usage:
The syntax is straightforward: typeof operand or typeof(operand), as parentheses are optional.

```
let count = 10;
console.log(typeof count); // Output: "number"

let name = "John";
console.log(typeof name);  // Output: "string"

let isActive = true;
console.log(typeof isActive); // Output: "boolean"
```

Return Values and Quirks:

While generally reliable for most primitive types, typeof has some specific behaviors and historical quirks you should be aware of:

Return Values of typeof:

string "string"

number "number"
Returns "number" even for NaN (Not a Number).

boolean "boolean"

undefined "undefined"
Works for both declared but unassigned variables and undeclared variables without throwing a ReferenceError.

symbol "symbol" Introduced in ES6.

bigint "bigint"

function "function"
Functions are a special type of object, but typeof returns "function" for them and classes.

object "object"
The generic return for objects, arrays, dates, and null.

null "object"
This is a well-known, historical bug in JavaScript and a key reason to use the strict equality operator (=== null) to specifically check for null values.
Practical Considerations

Arrays and Dates:
typeof returns "object" for arrays, dates, and regular expressions. To differentiate them, you often need additional methods like Array.isArray() or the instanceof operator.

Checking for Objects: A robust way to check if a value is a true object (and not null) is typeof myVar === 'object' && myVar !== null.

Error Handling: Using typeof to check for an undefined variable (e.g., if (typeof someVar === 'undefined')) is an effective way to prevent a ReferenceError if the variable might not be declared.