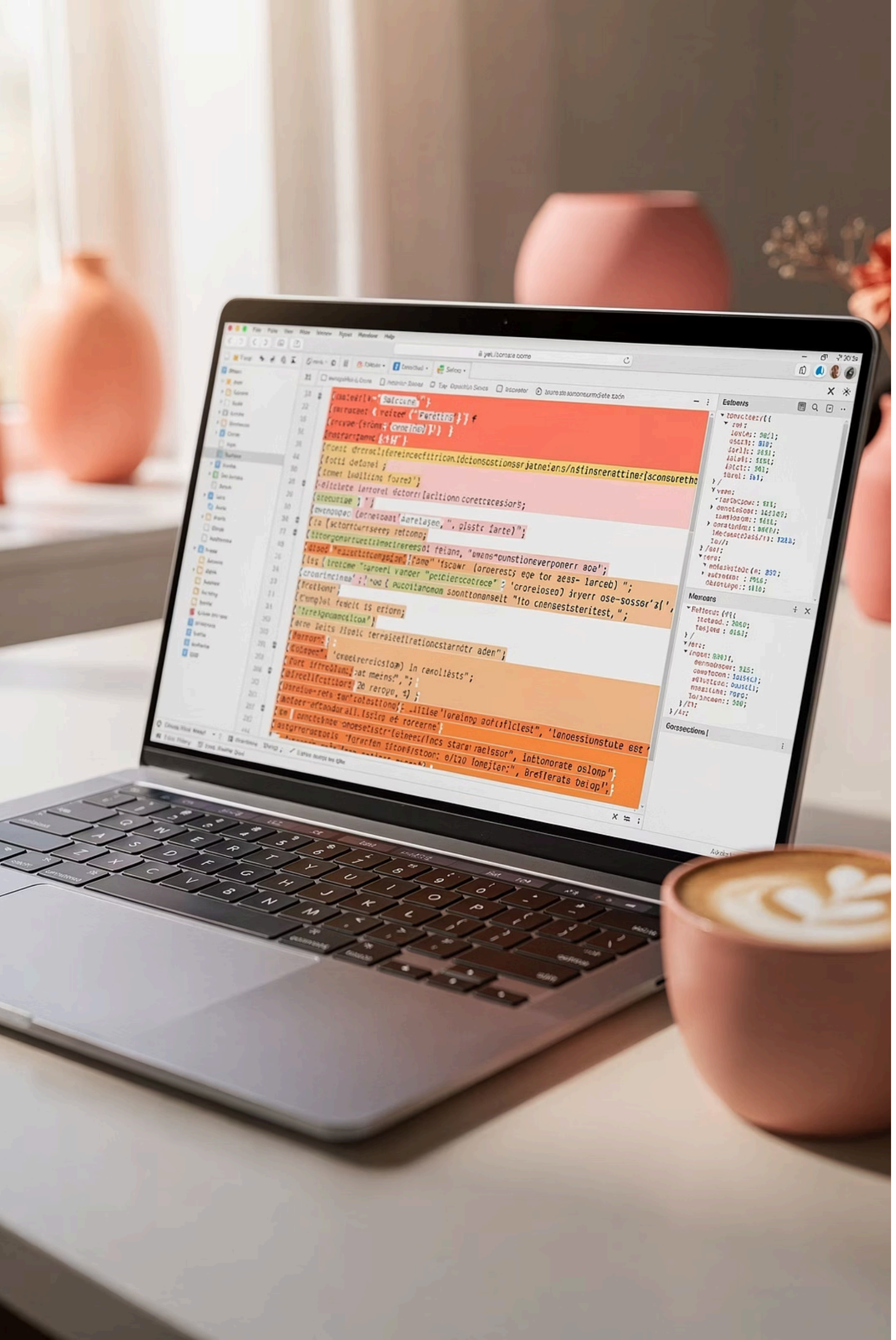


JavaScript

JavaScript brings web pages to life by enabling dynamic behaviors like click actions, search filtering, form validation, and real-time updates.

The browser console is your testing playground—a powerful tool where you can prototype and debug JavaScript instantly without touching your source files.

Click Actions

Respond to user interactions and trigger events

Search & Filter

Dynamically show or hide content based on input

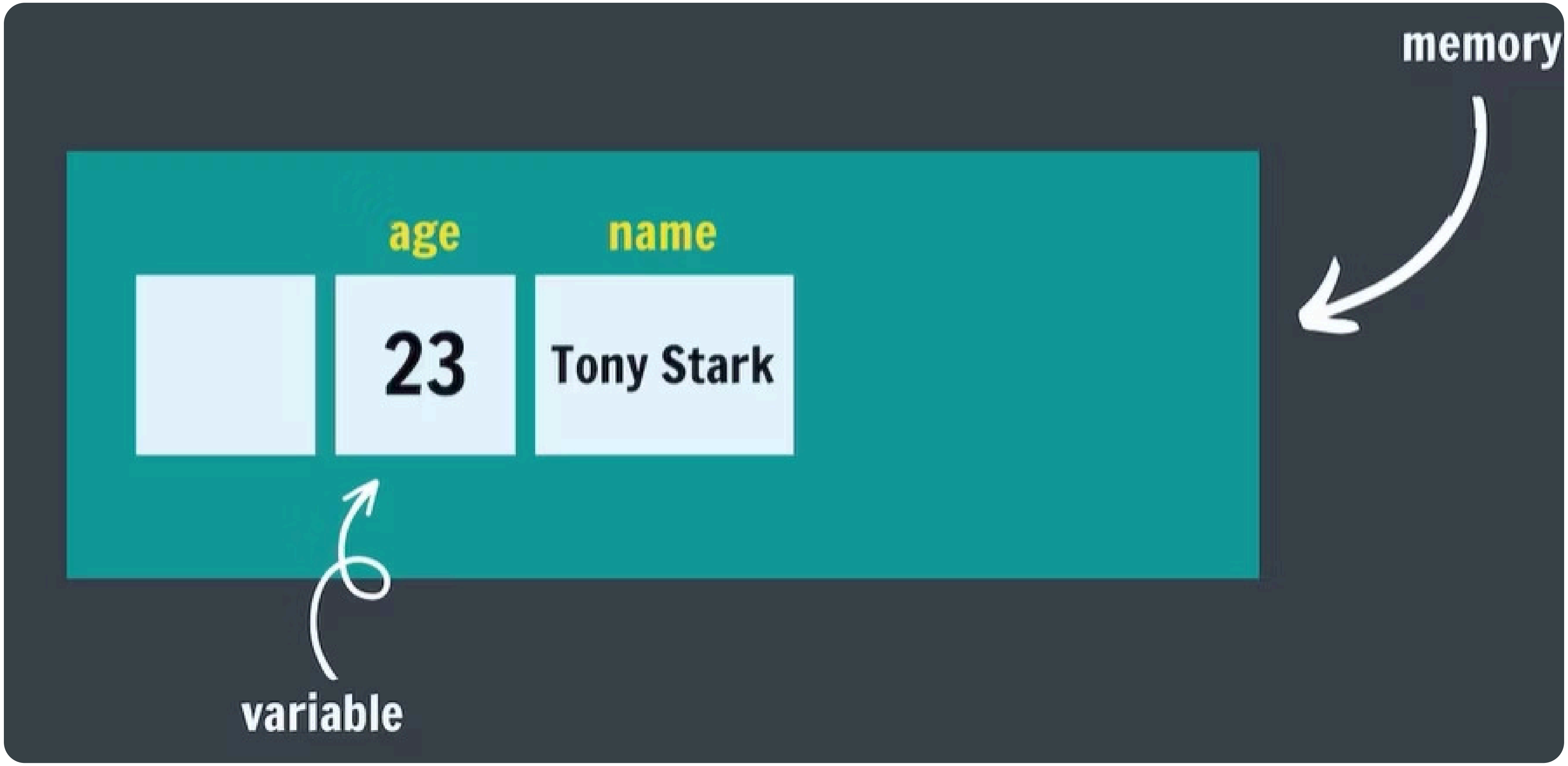
Live Updates

Modify page content without reloading

What is a Variable?

A variable is simply the name of a storage location.

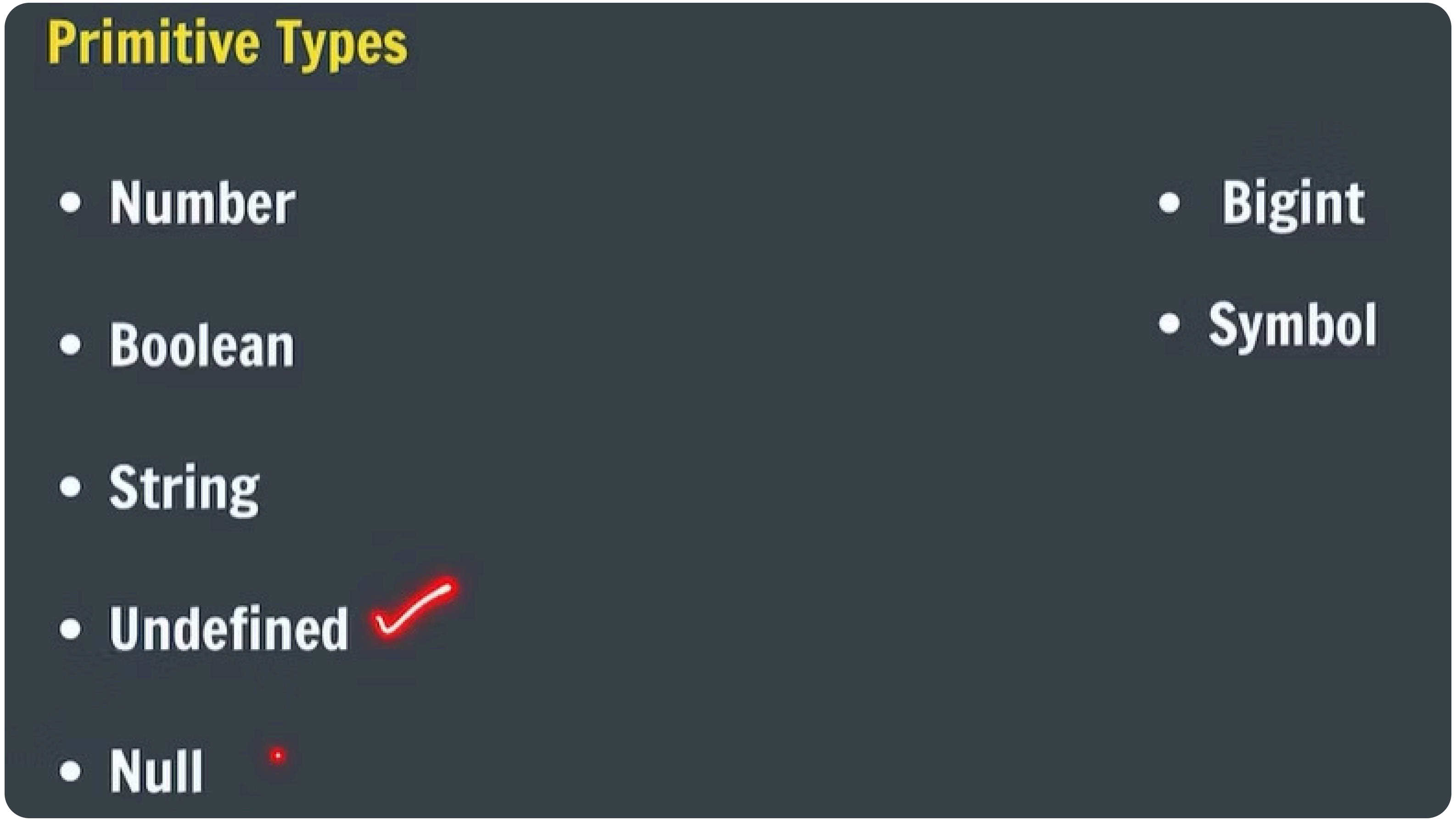
In JavaScript, variables store data values that can be used and changed throughout your code. They act like labeled containers that hold information, allowing you to reference and manipulate data dynamically.



Data Types in JS

JavaScript variables can store many different types of data values. Understanding these data types is fundamental to writing effective and error-free code, as they dictate how values behave and what operations can be performed on them.

JavaScript automatically computes the data type depending on the value given to the variable. This is called dynamic typing—you don't need to explicitly declare what type of data a variable will hold.

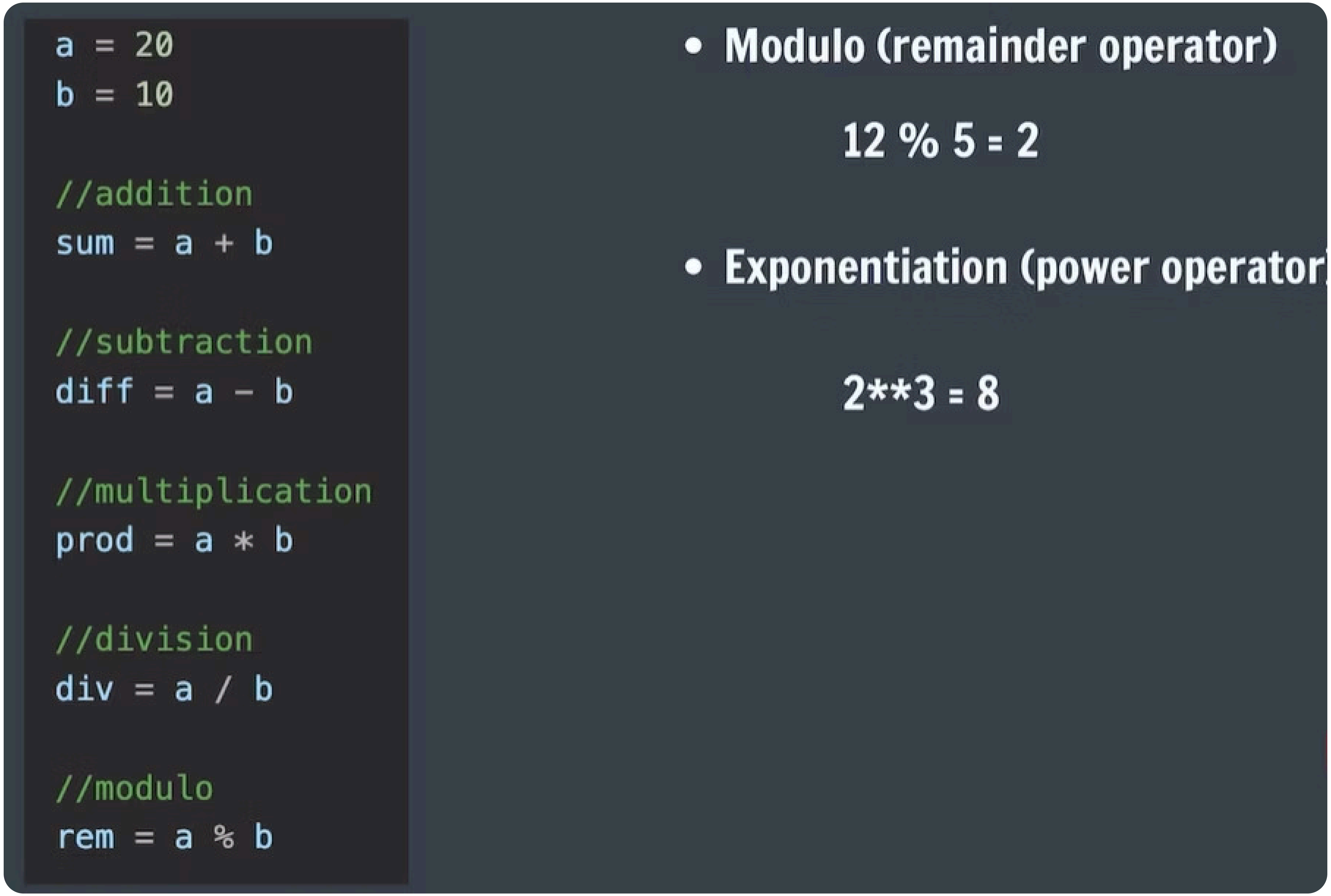


Primitive Types

- Number
- Boolean
- String
- Undefined
- Null
- Bigint
- Symbol

Operations in JS

JavaScript supports a wide array of mathematical, logical, and assignment operations that allow you to manipulate data, perform calculations, and control program flow. Understanding these operations is crucial for writing dynamic and interactive code.



```
a = 20
b = 10

//addition
sum = a + b

//subtraction
diff = a - b

//multiplication
prod = a * b

//division
div = a / b

//modulo
rem = a % b
```

- **Modulo (remainder operator)**

  12 % 5 = 2

- **Exponentiation (power operator**

  23 = 8**

NaN in JS

In JavaScript, `NaN` stands for "Not-A-Number". It is a special numeric value that represents an invalid or undefined numerical result from an operation, such as trying to parse non-numeric strings into numbers, or mathematical operations that don't yield a real number.

Examples that result in NaN:

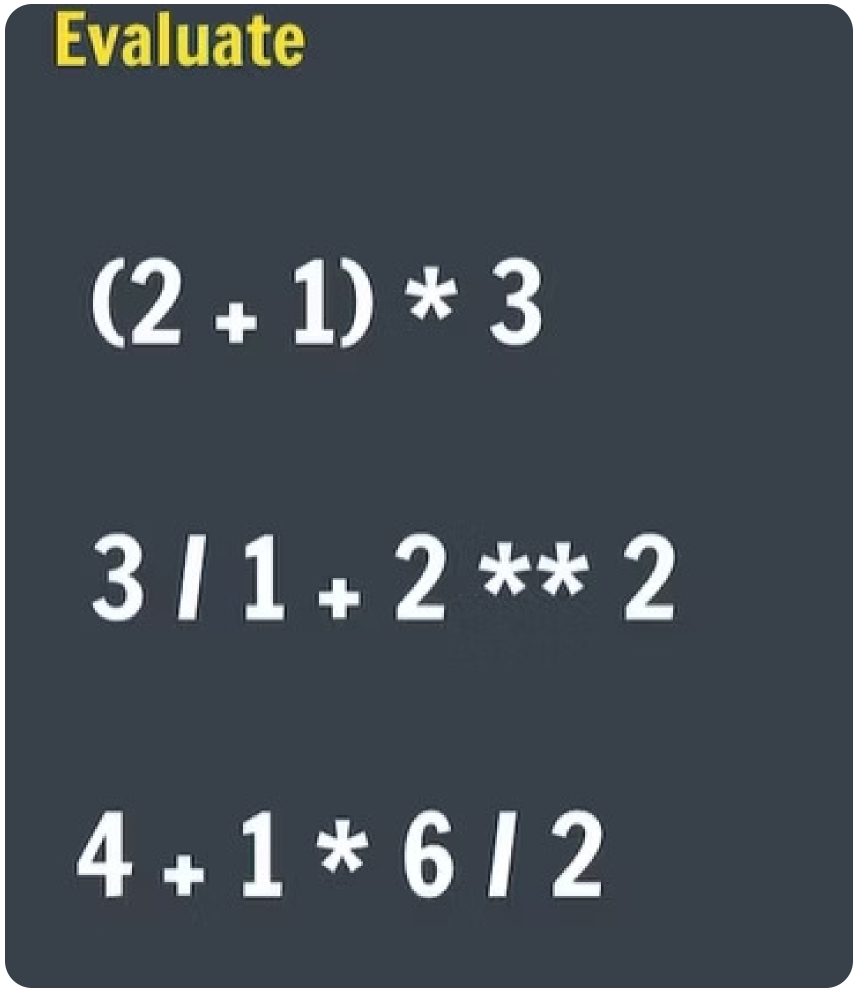
- `0 / 0`
- `NaN - 1`
- `NaN * 1`
- `NaN + NaN`

Operator Precedence

Operator precedence determines the order in which operations are performed in an expression. It's similar to the order of operations in mathematics (often remembered by acronyms like PEMDAS or BODMAS), ensuring that complex calculations yield consistent and predictable results.

Order from highest to lowest precedence:

1. `( )` - Parentheses
2. `**` - Exponentiation
3. `*`, `/`, `%` - Multiplication, Division, Modulo
4. `+`, `-` - Addition, Subtraction



Evaluate

```
(2 + 1) * 3

3 / 1 + 2 ** 2

4 + 1 * 6 / 2
```

The let Keyword

The let keyword is a fundamental way to declare variables in JavaScript. Variables declared with let are block-scoped, meaning they are only accessible within the block of code where they are defined. A key feature of let is that these variables can be reassigned new values after their initial declaration, making them suitable for values that need to change over time.

Here are some examples demonstrating the usage of the let keyword:

```
let age = 23;
age = age + 1; // 'age' is reassigned to 24
```

```
let cgpa; // Variable declared without initial value
cgpa = 8.9; // 'cgpa' is later assigned a value of 8.9
```

```
let num1 = 1;
let num2 = 2;
let sum = num1 + num2; // 'sum' is declared and initialized with the result of num1 + num2 (which is 3)
```

The const Keyword

The `const` keyword is used to declare constants in JavaScript. Unlike variables declared with `let`, variables declared with `const` cannot be reassigned after their initial declaration. Additionally, a key requirement for `const` is that these variables must be initialized with a value at the time of their declaration.

Here are some examples demonstrating the usage of the `const` keyword:

```
const year = 2025;
year = 2026;    // Error: Assignment to constant variable.
year = year + 1; // Error: Assignment to constant variable.
```

```
const pi = 3.14;
const g = 9.8;
```

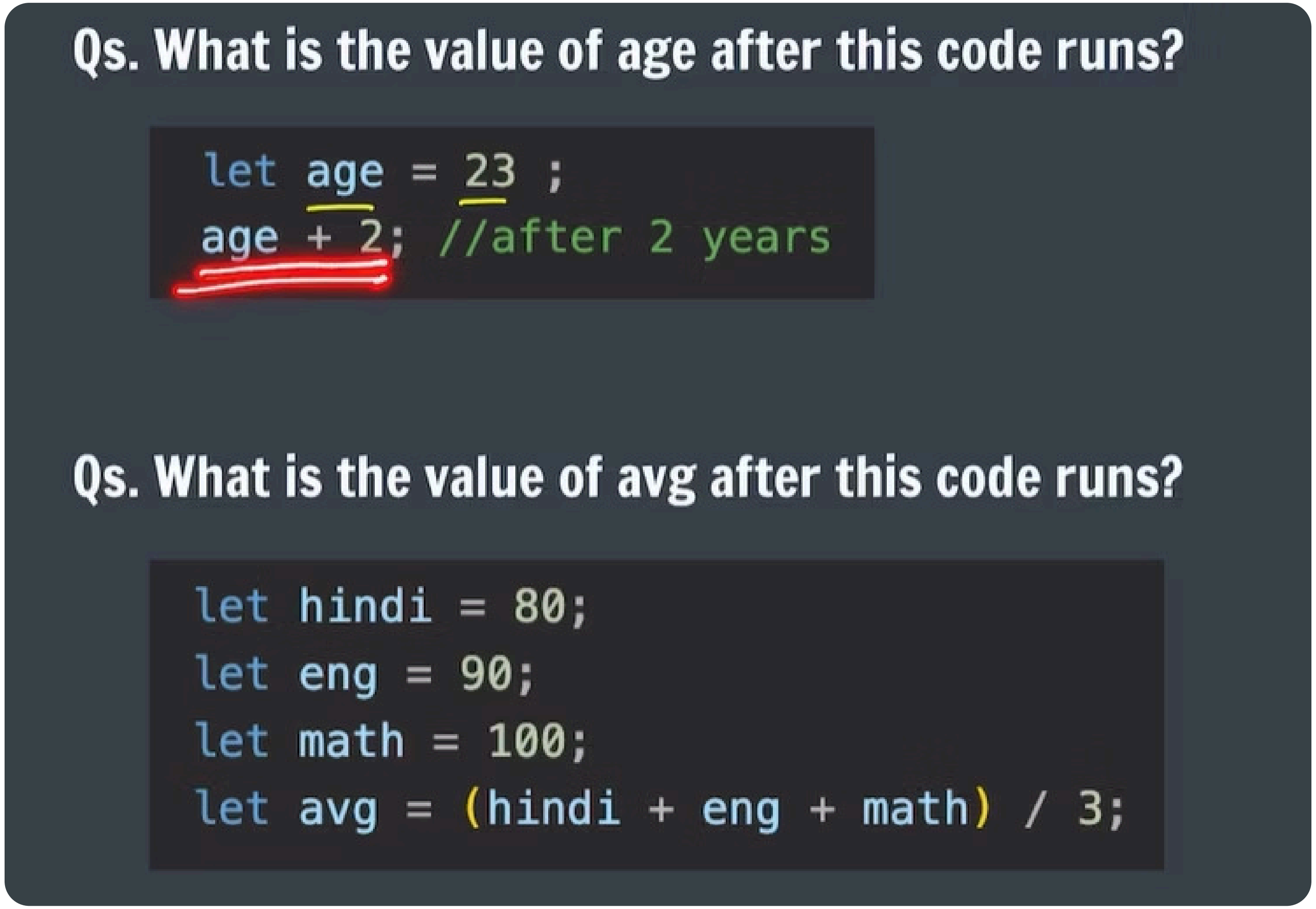
The var Keyword

The `var` keyword is the old syntax for declaring variables in JavaScript. Variables declared with `var` are function-scoped (or globally scoped if declared outside a function) and can be reassigned and re-declared. While `var` is still supported, modern JavaScript development prefers `let` and `const` for better scoping behavior and to avoid common pitfalls associated with `var`'s hoisting and scoping rules.

Here are some examples demonstrating the usage of the `var` keyword:

```
var age = 23;
var cgpa = 8.9;
var num1 = 1;
var num2 = 2;
var sum = num1 + num2;
```

Practice Qs



Qs. What is the value of age after this code runs?

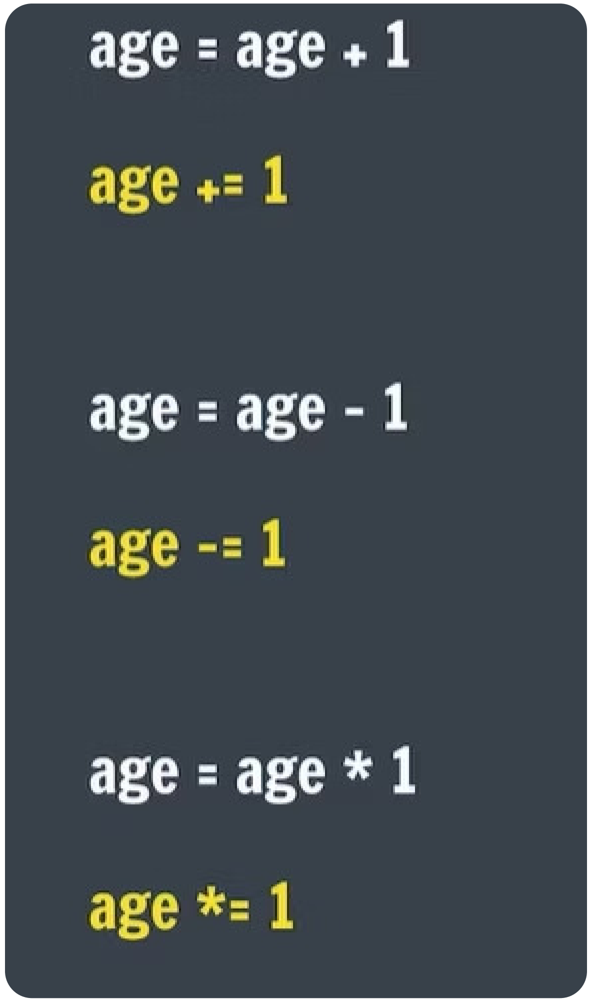
```
let age = 23 ;
age + 2; //after 2 years
```

Qs. What is the value of avg after this code runs?

```
let hindi = 80;
let eng = 90;
let math = 100;
let avg = (hindi + eng + math) / 3;
```

Assignment Operators

Assignment operators are used to assign values to variables in JavaScript. They provide shorthand ways to perform operations and assign the result back to the variable in a single step.

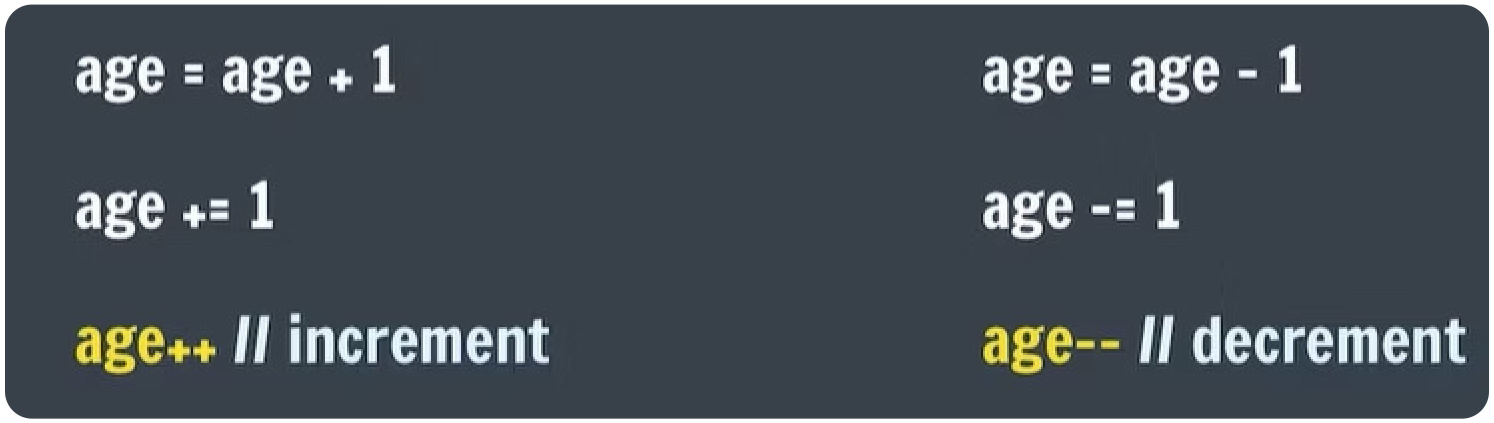


```
age = age + 1

age += 1


age = age - 1

age -= 1


age = age * 1

age *= 1
```

These demonstrate how compound assignment operators combine arithmetic operations with assignment for cleaner, more concise code.

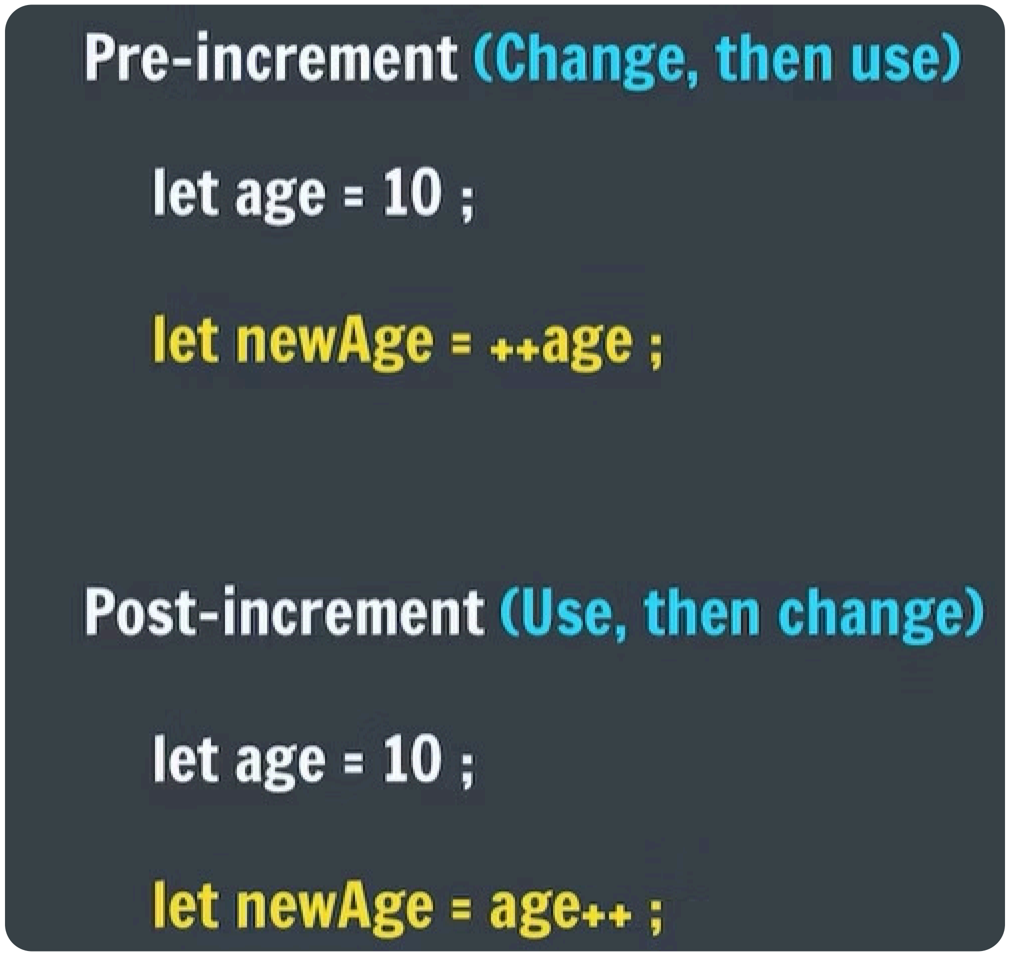
Unary Operators

Unary operators in JavaScript work with a single operand to perform an operation. Among the most common are the increment (++) and decrement (--) operators, which increase or decrease a variable's value by one, respectively.



```
age = age + 1                 age = age - 1

age += 1                      age -= 1

age++ // increment            age-- // decrement
```

The image above illustrates how the increment (`++`) and decrement (`--`) operators provide a concise way to add or subtract one from a variable's value, offering a shorthand for more verbose assignment operations.



```
Pre-increment (Change, then use)

    let age = 10 ;

    let newAge = ++age ;


Post-increment (Use, then change)

    let age = 10 ;

    let newAge = age++ ;
```

It's crucial to understand the difference between pre-increment/decrement (e.g., `++age`) and post-increment/decrement (e.g., `age++`). Pre-operators modify the variable's value first and then use the new value, while post-operators use the original value of the variable in the expression first and then modify the variable.

Identifier Rules

In JavaScript, identifiers are names given to variables, functions, and other entities. To ensure valid and readable code, specific rules must be followed when creating these identifiers.

Must start with a letter, underscore (_), or dollar sign ($)

Valid: `name`, `_count`, `$price`

Invalid: `1name`, `@value`

Can contain letters, digits, underscores, and dollar signs

Valid: `user1`, `total_price`, `$amount2`

Invalid: `user-name`, `total price`

Cannot use reserved keywords

Invalid: `let`, `const`, `var`, `if`, `for`, `while`, etc.

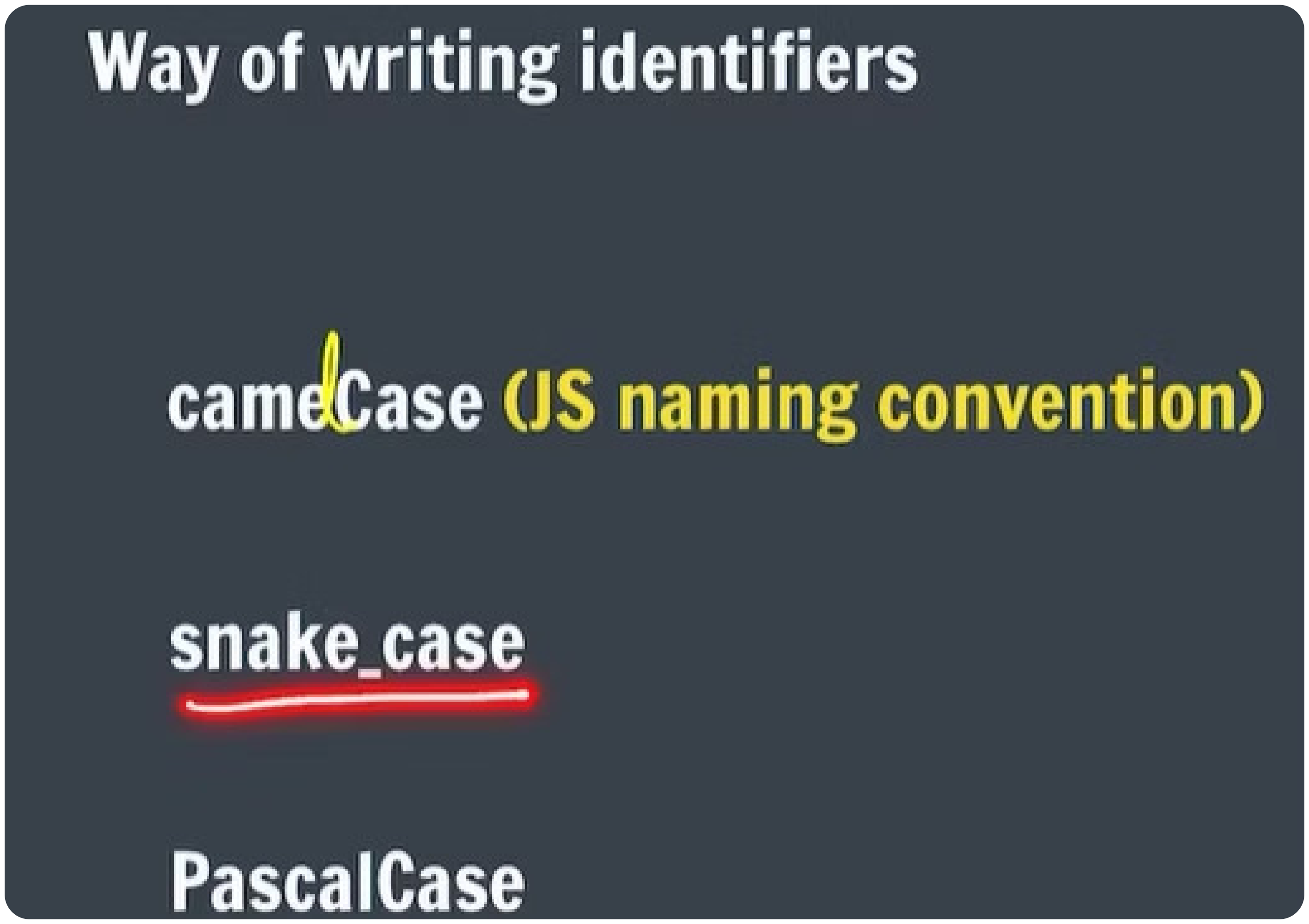
Case-sensitive

`userName` and `username` are different variables

Should be descriptive and meaningful

Good: `studentAge`, `totalPrice`

Avoid: `x`, `a`, `temp`



Way of writing identifiers

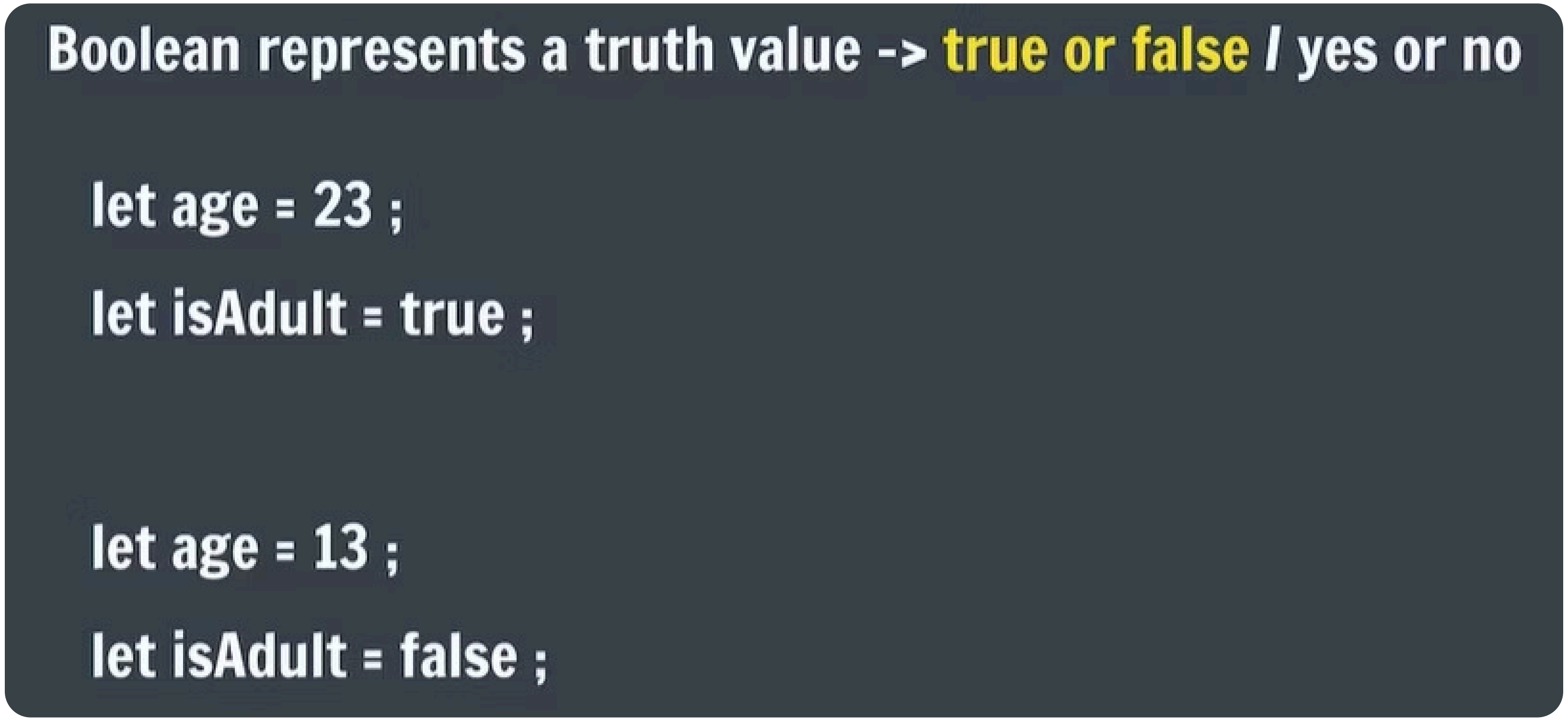
camelCase (JS naming convention)

snake_case

PascalCase

Boolean

A Boolean is a data type that represents a truth value. It can only have two possible values: true or false. Booleans are commonly used in conditional statements and logical operations to control program flow.



Boolean represents a truth value -> true or false / yes or no

```
let age = 23 ;
let isAdult = true ;


let age = 13 ;
let isAdult = false ;
```

Examples

Condition: Age 23

```
let age = 23 ;
let isAdult = true ;
```

Condition: Age 13

```
let age = 13 ;
let isAdult = false ;
```

What is TypeScript?

TypeScript is a superset of JavaScript developed by Microsoft that adds static typing to the language. It compiles down to plain JavaScript, meaning any valid JavaScript code is also valid TypeScript code.

Key features of TypeScript

1 Static Type Checking

Catch errors at compile time rather than runtime by explicitly defining variable types.

2 Enhanced IDE Support

Better autocomplete, refactoring, and navigation tools.

3 Modern JavaScript Features

Use the latest ECMAScript features with backward compatibility.

4 Object-Oriented Programming

Full support for classes, interfaces, and modules.

5 Gradual Adoption

Can be introduced incrementally into existing JavaScript projects.

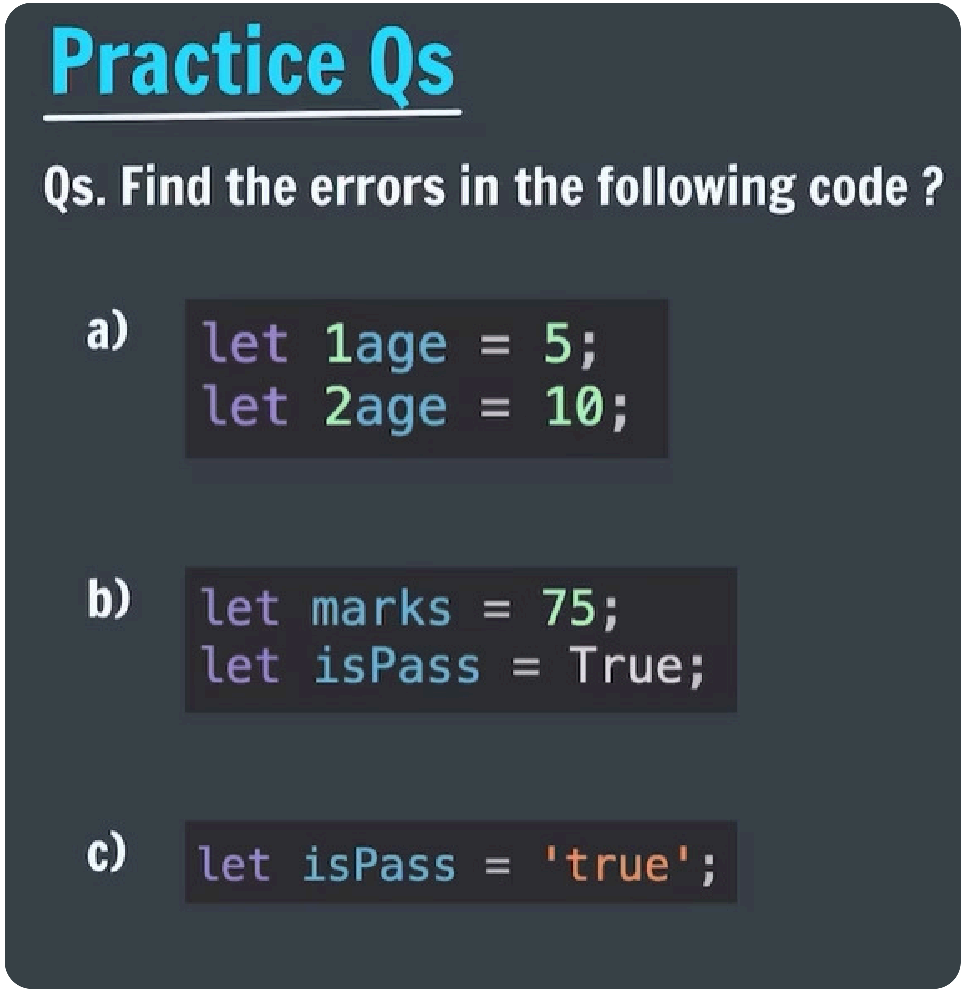
Example comparison

JavaScript:

```
let age = 23;
age = "twenty-three"; // No error
```

TypeScript:

```
let age: number = 23;
age = "twenty-three"; // Error: Type 'string' is not assignable to type 'number'
```



Practice Qs

Qs. Find the errors in the following code ?

a)

```
let 1age = 5;
let 2age = 10;
```

b)

```
let marks = 75;
let isPass = True;
```

c)

```
let isPass = 'true';
```

String in JS

Strings are a fundamental data type in JavaScript used to represent text or sequences of characters. Strings can be enclosed in double quotes (""), single quotes (''), or backticks (` `).

Examples

```
let name = "Tony Stark" ;
```

```
let role = 'ironman' ;
```

```
let char = 'a' ;
```

```
let num = '23' ;
```

```
let empty = " " ;
```