PART 1

MASYNCTECH
SCHOOL

FULLSTACK WEB DEV

JAVASCRIPT JS

MASYNCTECH MASYNCTECH www.masynctech.com

MASYNCTECH
SCHOOL

JAVASCRIPT OVERVIEW

JS

GETTING STARTED

WHAT IS JAVASCRIPT?

MASYNCTECH SCHOOL

High-Level Explanation

- **JavaScript**: Enables interactivity, dynamism on websites
- Create interactive forms, animations, games in browsers

Analogue

- Imagery: JavaScript as magic sauce
- Transforms static pages into interactive experiences
- Click buttons, see real-time changes, no page reloads

Summary

- Versatile JavaScript: Interactive websites to data science
- Web, server, mobile, data roles
- Easy start, mastery needs best practices, principles

Deep Dive

- **JavaScript**: Evolved from web interactivity to versatile use
- **Server-side, mobile apps, etc.**
- Runs in browsers, no extra software
- **Manipulates HTML, CSS**, dynamic content
- Handles events, fetches real-time data

When to use?

- JavaScript in various domains:
  - **Web Development**: Interactivity
  - **Server-Side Development**: Backend with Node.js
  - **Mobile App Development**: React Native, Ionic
  - **Data Analysis:** TensorFlow.js and data science

MASYNCTECH
SCHOOL

SETTING DEVELOPMENT ENVIRONMENT

JS

GETTING STARTED

MASYNCTECH
SCHOOL

YOUR FIRST JS CODE

JS

GETTING STARTED

MASYNCTECH
SCHOOL

WAYS OF INCLUDING JS IN HTML

JS

GETTING STARTED

WAYS OF INCLUDING JS IN HTML

MASYNCTECH
SCHOOL

High-Level Explanation

- Include JavaScript in HTML:
  - `<**script**>` tag
  - **External** `.js` file link
  - **Inline JavaScript** in HTML elements
- `**async**` and `**defer**` attributes for external file control

WAYS OF INCLUDING JS IN HTML

MASYNCTECH
SCHOOL

High-Level Explanation

- Include JavaScript in HTML:
  - `<**script**>` tag
  - **External** `.js` file link
  - **Inline JavaScript** in HTML elements
- `**async**` and `**defer**` attributes for external file control

Deep Dive

- **Embedded JavaScript:** Code within `<script>` tags, runs where located.
- **External JavaScript with `async`**: `<script async src="script.js">`
  - Downloads and executes immediately.
- **External JavaScript with `defer`**: `<script defer src="script.js">`
  - Downloads, executes after HTML parsing.
- **Inline JavaScript**: Within HTML element attributes, e.g., `onclick`, `onload`.

WAYS OF INCLUDING JS IN HTML

MASYNCTECH
SCHOOL

High-Level Explanation

- Include JavaScript in HTML:
  - `<**script**>` tag
  - **External** `.js` file link
  - **Inline JavaScript** in HTML elements
- `**async**` and `**defer**` attributes for external file control

Analogue

- Party analogy for JavaScript inclusion:
  - Embedded JavaScript: Live band playing as guests arrive.
  - External JavaScript with `async`: Early friend interrupting, mingling.
  - External JavaScript with `defer`: Special guest waiting, grand entrance.
  - Inline JavaScript: Magic trick for individual guests.

Deep Dive

- **Embedded JavaScript:** Code within `<script>` tags, runs where located.
- **External JavaScript with `async`**: `<script async src="script.js">`
  - Downloads and executes immediately.
- **External JavaScript with `defer`**: `<script defer src="script.js">`
  - Downloads, executes after HTML parsing.
- **Inline JavaScript**: Within HTML element attributes, e.g., `onclick`, `onload`.

WAYS OF INCLUDING JS IN HTML

MASYNCTECH SCHOOL

High-Level Explanation

- Include JavaScript in HTML:
  - `<**script**>` tag
  - **External** `.js` file link
  - **Inline JavaScript** in HTML elements
- `**async**` and `**defer**` attributes for external file control

Analogue

- Party analogy for JavaScript inclusion:
  - Embedded JavaScript: Live band playing as guests arrive.
  - External JavaScript with `async`: Early friend interrupting, mingling.
  - External JavaScript with `defer`: Special guest waiting, grand entrance.
  - Inline JavaScript: Magic trick for individual guests.

Deep Dive

- **Embedded JavaScript:** Code within `<script>` tags, runs where located.
- **External JavaScript with `async`**: `<script async src="script.js">`
  - Downloads and executes immediately.
- **External JavaScript with `defer`**: `<script defer src="script.js">`
  - Downloads, executes after HTML parsing.
- **Inline JavaScript**: Within HTML element attributes, e.g., `onclick`, `onload`.

When to use?

- Embedded JavaScript: Quick tests, prototypes.
- External JavaScript with `async`: Independent scripts.
- External JavaScript with `defer`: DOM-dependent scripts.
- Inline JavaScript: Simple, one-off actions.

WAYS OF INCLUDING JS IN HTML

MASYNCTECH SCHOOL

High-Level Explanation

- Include JavaScript in HTML:
  - `<**script**>` tag
  - **External** `.js` file link
  - **Inline JavaScript** in HTML elements
- `**async**` and `**defer**` attributes for external file control

Analogue

- Party analogy for JavaScript inclusion:
  - Embedded JavaScript: Live band playing as guests arrive.
  - External JavaScript with `async`: Early friend interrupting, mingling.
  - External JavaScript with `defer`: Special guest waiting, grand entrance.
  - Inline JavaScript: Magic trick for individual guests.

Important Rules

- Embedded JavaScript: Minimal, demos, testing.
- External JavaScript with `async`: DOM-independent scripts.
- External JavaScript with `defer`: For fully-loaded DOM.
- Inline JavaScript: Avoid for cleaner alternatives.

Deep Dive

- **Embedded JavaScript:** Code within `<script>` tags, runs where located.
- **External JavaScript with `async`**: `<script async src="script.js">`
  - Downloads and executes immediately.
- **External JavaScript with `defer`**: `<script defer src="script.js">`
  - Downloads, executes after HTML parsing.
- **Inline JavaScript**: Within HTML element attributes, e.g., `onclick`, `onload`.

When to use?

- Embedded JavaScript: Quick tests, prototypes.
- External JavaScript with `async`: Independent scripts.
- External JavaScript with `defer`: DOM-dependent scripts.
- Inline JavaScript: Simple, one-off actions.

WAYS OF INCLUDING JS IN HTML

MASYNCTECH SCHOOL

High-Level Explanation

- Include JavaScript in HTML:
  - `<**script**>` tag
  - **External** `.js` file link
  - **Inline JavaScript** in HTML elements
- `**async**` and `**defer**` attributes for external file control

Analogue

- Party analogy for JavaScript inclusion:
  - Embedded JavaScript: Live band playing as guests arrive.
  - External JavaScript with `async`: Early friend interrupting, mingling.
  - External JavaScript with `defer`: Special guest waiting, grand entrance.
  - Inline JavaScript: Magic trick for individual guests.

Important Rules

- Embedded JavaScript: Minimal, demos, testing.
- External JavaScript with `async`: DOM-independent scripts.
- External JavaScript with `defer`: For fully-loaded DOM.
- Inline JavaScript: Avoid for cleaner alternatives.

Deep Dive

- **Embedded JavaScript:** Code within `<script>` tags, runs where located.
- **External JavaScript with `async`**: `<script async src="script.js">`
  - Downloads and executes immediately.
- **External JavaScript with `defer`**: `<script defer src="script.js">`
  - Downloads, executes after HTML parsing.
- **Inline JavaScript**: Within HTML element attributes, e.g., `onclick`, `onload`.

When to use?

- Embedded JavaScript: Quick tests, prototypes.
- External JavaScript with `async`: Independent scripts.
- External JavaScript with `defer`: DOM-dependent scripts.
- Inline JavaScript: Simple, one-off actions.

Summary

- JavaScript inclusion methods vary with use-cases.
- `async` and `defer` usage controls script execution, enhances performance.

MASYNCTECH
SCHOOL

JS VARIABLES OVERVIEW

JS

VARIABLES & OPERATORS

JS VARIABLES OVERVIEW

MASYNCTECH
SCHOOL

Syntax

How to declare a variable?

var

let/const

keyword

=

Value/Items

Variable Name/Container

JS VARIABLES OVERVIEW

Syntax

```
1   const variableName = 'some value';
```

Reserved Keyword
eg: var,let,const

Any Identifier

Value

JS VARIABLES OVERVIEW

MASYNCTECH SCHOOL

High-Level Explanation

- JavaScript variables: Data containers for code manipulation.

Analogue

- Imagery: JavaScript variables as backpack pockets
- Pockets (variables) hold stuff (data)
- Dynamic typing: Pockets hold various stuff types
- `let`: Zippers to replace stuff
- `const`: Sealed, stuff can't be changed

Important Rules

- Default to `const`, unless reassignment needed.
- Avoid `var` in new code; use `let` or `const`.
- Use descriptive, camelCase variable names for readability.

Deep Dive

- JavaScript variables: Hold data values or references.
- Keywords: `var`, `let`, `const`.
- `**var**`: Function-scoped, redeclarable, updatable.
- `**let**`: Block-scoped, updatable, not redeclarable.
- `**const**`: Block-scoped, not updatable, not redeclarable.

When to use?

- **var**: Older code, or when function-scoping is needed.
- **let**: Variable value changes.
- **const**: Variable value remains constant.

Summary

- JavaScript variables: Data storage and management.
- Use `var`, `let`, or `const` based on needs.
- Select based on best practices and specific use-cases.

MASYNCTECH
SCHOOL

DIFFERENCES BETWEEN VAR, LET & CONST

JS

VARIABLES & OPERATORS

DIFFERENCES BETWEEN VAR,LET & CONST

MASYNCTECH SCHOOL

High-Level Explanation

- Imagery: Variable types as boxes
- `var`: Old, changeable box
- `let`: Modern, changeable with rules
- `const`: Locked, unchangeable box

Deep Dive

- `var`: Function-scoped, hoisted, less predictable.
- `let`: Block-scoped, not hoisted, more predictable.
- `const`: Block-scoped, immutable (not reassignable).

Analogue

- Imagery: Lunchboxes representing variables
- `var`: Opens, spills unpredictably.
- `let`: Smart, stays shut, changeable.
- `const`: Locked, unchangeable, choose wisely.

When to use?

- `var`: Less common, function-scoped.
- `let`: Value changes, block-specific.
- `const`: Value constant, consistent.

Important Rules

- `var`: Avoid, prefer `let` and `const`.
- `let`: Changeable variables.
- `const`: Immutable variables for clarity.

Summary

- `var`, `let`, and `const`: Variable declaration.
- `var`: Function-scoped, older, avoid.
- `let`: Changeable, block-scoped.
- `const`: Immutable, block-scoped.

MASYNCTECH
SCHOOL

DATA TYPES

JS

VARIABLES & OPERATORS

DATA TYPES

MASYNCTECH
SCHOOL

Syntax

How to declare a variable?

var

let/const

keyword

=

Value/Items

Variable Name/Container

WHAT ARE DATA TYPES

Data Types	Explanation
Number	Numeric data, e.g., 42, 3.14
String	Textual data in quotes, e.g., "Hello", 'World'
Boolean	True or False value, e.g., true, false
Undefined	Variable declared but not initialized
Null	Variable explicitly set to no value
BigInt	Large integers beyond Number limit
Symbol	Unique value, can't be duplicated
Object	Collection of key-value pairs
Array	Ordered collection of elements

WHAT ARE DATA TYPES

MASYNCTECH SCHOOL

High-Level Explanation

- JavaScript data types: Containers for storing items
- Hold numbers, words, lists, etc.

Analogue

- Primitive Types: Solo cups for individual drinks
- Non-Primitive Types: Punch bowls for mixing and customization

Important Rules

- Use `const` for immutability
- Select appropriate data type (e.g., String vs. Number)
- Understand mutability of Objects and Arrays

Deep Dive

- JavaScript: Dynamically typed
- Two data type categories:
  - **Primitive Types**: Immutable, single value
    - Number, String, Boolean, Undefined, Null, BigInt, Symbol
  - **Non-Primitive Types**: Mutable, multiple values
    - Objects, Arrays

When to use?

- Primitive Types: Single, unchangeable values (e.g., user info, settings)
- Non-Primitive Types: Complex data (e.g., book list, user account configs)

Summary

- JavaScript data types: Primitive (Number, String, Boolean) for single, immutable values, and non-primitive (Objects, Arrays) for complex, mutable data.
- Crucial to choose the right type for effective coding.

MASYNCTECH
SCHOOL

JS OPERATORS OVERVIEW

JS

VARIABLES & OPERATORS

OPERATORS IN JAVASCRIPT OVERVIEW

MASYNCTECH
SCHOOL

High-Level Explanation

- JavaScript operators: Symbols for operations
- Tools in math-like equations
- Compare, calculate, assign, and more

Deep Dive

JavaScript operators are categorized into:

- **Arithmetic Operators:** Math calculations
- **Comparison Operators**: Value comparisons
- **Logical Operators**: Boolean logic
- **Assignment Operators:** Variable value setting
- **Unary Operators:** Single operand operations
- **Ternary Operator:** Short conditional operation
- **Bitwise Operators**: Binary representation level operations

OPERATORS IN JAVASCRIPT OVERVIEW

Category	Examples
Arithmetic	+, -, *, /, %, ++, --
Comparison	==, !=, ===, !==, >, <, >=, <=
Logical	&&, ||, !
Assignment	=, +=, -=, *=, /=, %=
Unary	typeof, instanceof, delete, void
Ternary	? :
Bitwise	&, |, ^, ~, <<, >>, >>>

MASYNCTECH
SCHOOL

ARITHMETIC OPERATORS

JS

VARIABLES & OPERATORS

ARITHMETIC OPERATORS

MASYNCTECH SCHOOL

High-Level Explanation

- JavaScript arithmetic operators: Perform math operations
- Examples: Addition, subtraction, multiplication, division

Analogue

- Arithmetic operators: Fundamental math tools
- Perform tasks like addition, division remainder, and decrement

Important Rules

- Operate on numbers or numeric strings.
- Use increment/decrement wisely; readability matters in complex expressions.

Deep Dive

- **Addition (`+`)** : Add two numbers.
- **Subtraction (`-`)** : Subtract right from left operand.
- **Multiplication (`*`)** : Multiply two numbers.
- **Division (`/`)** : Divide left by right operand.
- **Modulus (`%`)** : Get remainder after division.
- **Increment (`++`)** : Increase value by one.
- **Decrement (`--`)** : Decrease value by one.

When to use?

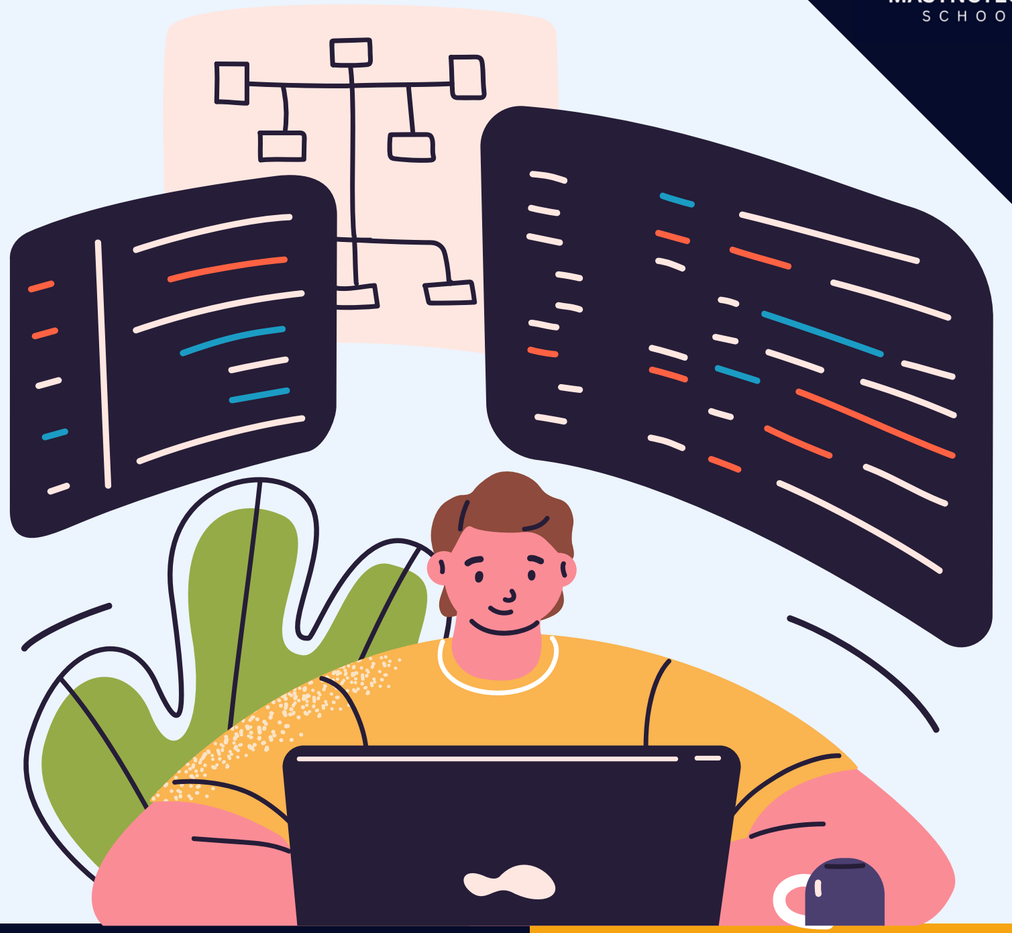
- Apply addition, subtraction, multiplication, division for math.
- Employ modulus for remainders, useful in logic.
- Use increment/decrement for quick value adjustments, like in loops.

Summary

- JavaScript arithmetic operators: Crucial for math in coding
- Fundamental for algorithms, logic in apps

MASYNCTECH
SCHOOL

COMPARISON OPERATORS



JS

VARIABLES & OPERATORS

COMPARISON OPERATORS

MASYNCTECH SCHOOL

High-Level Explanation

- JavaScript comparison operators: Compare values, yield true/false
- Determine truth of comparisons

Analogue

- Comparison operators: Like game referees
- Compare, make true/false calls

Important Rules

- Favor `===` and `!==` to prevent type conversion surprises.
- Ensure appropriate data types for comparisons.

Summary

- JavaScript comparison operators: Enable value comparisons
- Aid decision-making by returning boolean results

Deep Dive

- **Equal to (`==`):** Check equal values after type conversion.
- **Not equal to (`!=`):** Check unequal values after type conversion.
- **Strictly equal (`===`):** Verify equal values and types.
- **Strictly not equal (`!==`)**: Confirm unequal values or types.
- **Greater than (`>`)**: Examine if left is greater than right.
- **Less than (`<`)**: Examine if left is less than right.
- **Greater than or equal to (`>=`)**: Check if left is greater than or equal to right.
- **Less than or equal to (`<=`)**: Check if left is less than or equal to right.

When to use?

- **Equal to (`==`)** and **Not equal to (`!=`)** for type-agnostic comparisons.
- **Strict comparisons (`===`, `!==`)** for type-aware comparisons, preventing type conversion surprises.

MASYNCTECH
SCHOOL

LOGICAL OPERATORS

JS

VARIABLES & OPERATORS

LOGICAL OPERATORS

MASYNCTECH SCHOOL

High-Level Explanation

- JavaScript logical operators: Perform logic on values
- Determine relationships between conditions

Analogue

- Analogy for logical operators:
  - AND (`&&`): You and your friend agree
  - OR (`||`): Either you or your friend agree
  - N[illegible]d

Important Rules

- Ensure logical conditions align with your goal's context.
- Logical operators yield booleans for conditionals.

Summary

- Logical operators: Powerful for multi-condition decisions
- Return booleans based on applied logic

Deep Dive

There are mainly three logical operators:

- **AND (`&&`):** Returns true if both operands are true.
- **OR (`||`):** Returns true if at least one operand is true.
- **NOT (`!`)**: Returns true if the operand is false and false if it's true.

When to use?

- AND (`&&`): All conditions must be true.
- OR (`||`): At least one condition must be true.
- NOT (`!`): Reverse truth value of operand.

MASYNCTECH
SCHOOL

ASSIGNMENT OPERATORS

JS

VARIABLES & OPERATORS

ASSIGNMENT OPERATORS

MASYNCTECH
SCHOOL

High-Level Explanation

- JavaScript assignment operators: Assign values to variables
- Used for setting variable values

Deep Dive

- `=`: Assigns a value to a variable.
- `+=`: Adds right value to the left variable.
- `-=`: Subtracts right value from the left variable.
- `*=`: Multiplies left variable by right value.
- `/=`: Divides left variable by right value.
- `%=`: Finds modulus of left variable and right value.

Analogue

- Assignment operators: Combine calculation and assignment.
- Shortcut math for updating variable values in one step.

When to use?

- Utilize assignment operators for concise, readable code.
- Streamline variable calculations and assignments.

Important Rules

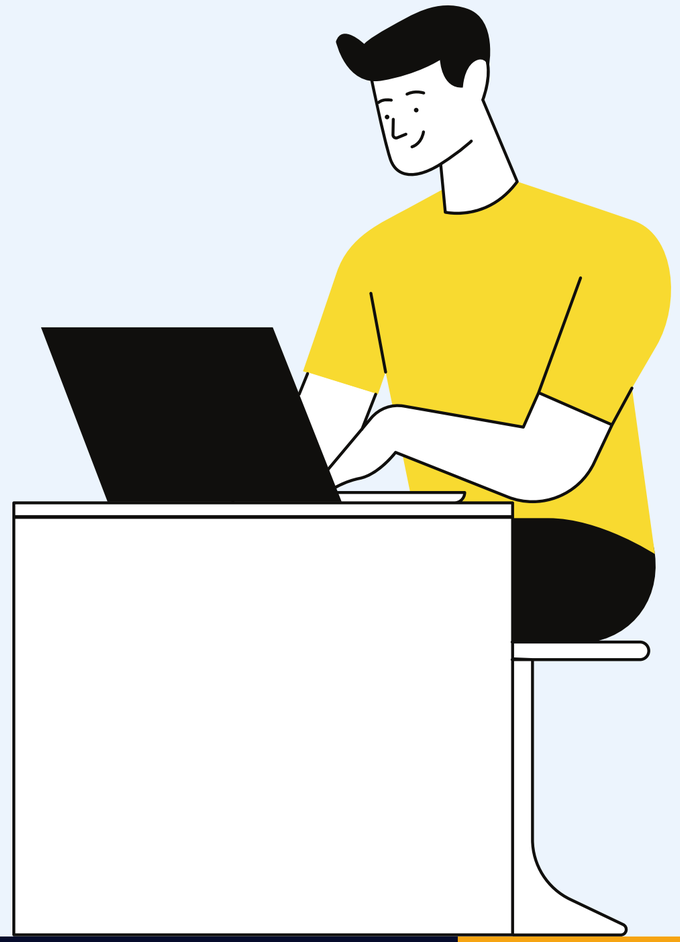
- Declare the variable before assignment.
- Consider order of operations in complex calculations.

Summary

- Assignment operators in JavaScript: Efficient for calculations and variable assignments.
- Streamline code for better readability and efficiency.

MASYNCTECH
SCHOOL

TERNARY OPERATORS



JS

VARIABLES & OPERATORS

TERNARY OPERATOR

MASYNCTECH
SCHOOL

High-Level Explanation

- Ternary operator: Shortened if-else statement
- Checks a condition
- Returns one value if true, another if false

Deep Dive

- Ternary operator parts: Condition, Value if true, Value if false.
- Format: `**`condition ? valueIfTrue : valueIfFalse;`**`

Analogue

- Analogy: Ternary operator as a decision maker
- Condition: "Is it raining?"
- Value if true: "Take an umbrella"
- Value if false: "Take sunglasses"

When to use?

- Ternary operator ideal for simple, one-line decisions.
- Use for straightforward conditions and value choices.
- Best for concise, readable code.

Important Rules

- Ternary operator for simple, concise conditionals.
- Avoid nested ternary operators for code readability.

Summary

- Ternary operator: Concise for conditional statements.
- Suitable for simple decisions in code.
- Efficient and readable.