Module 01

Purwadhika
Digital Technology School

Full Stack Web Development

Intro to programming, variables and data types

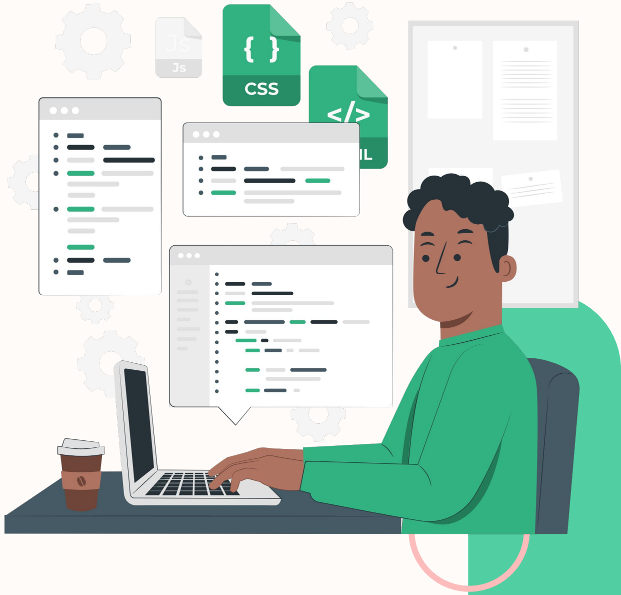
Job Connector Program

Outline

- Introduction to Programming
- Introduction to Algorithm
- Introduction to JavaScript
- Variable
- Data Types
- Introduction to TypeScript
- Type Conversion
- Operator

Introduction to Programming

- **What** is programming ?
- **Why** learn programming ?
- **What** is programming language ?



What is Programming ?

Programming is **the process of creating a set of instructions that tell computer to perform a task.**

What is Programming Language ?

A programming language is a **vocabulary** and set of **grammatical** rules for instructing a computer or computing device to perform specific tasks.

Example :

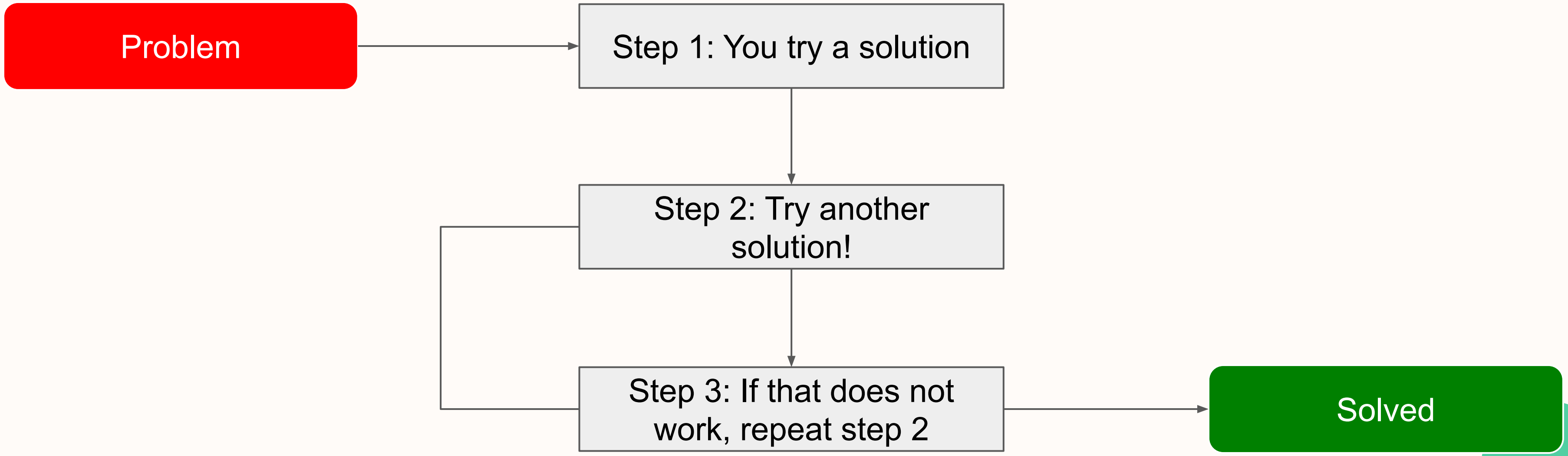
Javascript, Java, Golang, PHP, C, C++, C#, etc.

Think Like a Programmer

Before we jump in to the Algorithm, let’s talk about problem solving first.

Problem-solving skills are the ability to identify problems, brainstorm and analyze answers, and implement the best solutions. Programmers with good problem-solving skills is both a self-starter and a collaborative teammate; they are proactive in understanding the root of a problem and work with others to consider a wide range of solutions before deciding how to move forward.

Think Like a Programmer


Problem
Step 1: You try a solution
Step 2: Try another solution!
Step 3: If that does not work, repeat step 2
Solved

Why Learn Programming ?

- Improve problem solving & logical thinking
- Grow your creativity
- Level-up your career
- Great earning potential
- Technology are ruling the world

“Everybody in this country should learn how to program a computer … because it teaches you how to think”

–Steve Jobs–

Introduction to Algorithm

- What Is an Algorithm?
  - An algorithm is a set of step-by-step procedures, or a set of rules to follow, for completing a specific task or solving a particular problem. Algorithms are all around us.
- Why are Algorithms Important to Understand?
  - Algorithmic thinking, or the **ability to define clear steps to solve a problem**, is crucial in many different fields. Even if we're not conscious of it, we use algorithms and algorithmic thinking all the time. Algorithmic thinking allows you to break down problems and conceptualize solutions in terms of discrete steps. Being able to understand and implement an algorithm requires you to **practice structured thinking and reasoning abilities.**

Introduction to Algorithm

- Where are Algorithms Used as a Web Developer?

Algorithms are used in every part of IT industries. They form the field's backbone. Algorithm gives the computer a specific set of instructions, which allows the computer to do everything. **Algorithms written in programming languages that the computer can understand.**

Computer algorithms play a big role in how social media works: which posts show up, which ads are seen, and so on. These decisions are all made by algorithms.

Google’s programmers use algorithms to optimize searches, predict what users are going to type, and more. **In problem-solving, a big part of computer programming is knowing how to formulate an algorithm**.

Introduction to JavaScript

- **What** is JavaScript ?
- **Why** use JavaScript ?
- Setting up development environment
- Let’s write our first code !
- JavaScript **code structure**

What is JavaScript ?

JavaScript is a programming language. It is lightweight and **most commonly used as a part of web pages.** It is an **interpreted programming language** with **object-oriented** capabilities.

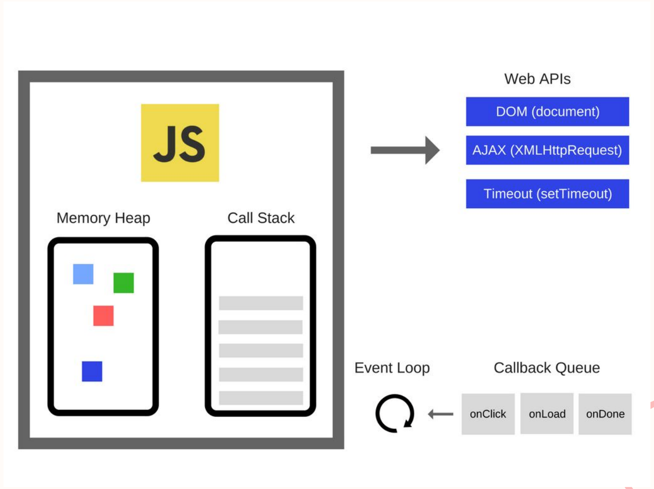
JavaScript can execute **not only in the browser, but also on the server, or actually on any device** that has a special program called the **JavaScript engine**.

What is JavaScript ?

Javascript is **single-threaded**, **non-blocking**, **asynchronous**, **concurrent** language.

- **Single-threaded** means that it runs only one thing at a time.
- **Non-blocking & Asynchronous** means that it doesn't wait for the response of an API call, I/O events, etc., and can continue the code execution.
- **Concurrent** means executing multiple tasks at the same time but not simultaneously. E.g. two tasks works in overlapping time periods.

For more information, watch this **video**



Why Use JavaScript ?

- Easy to Learn
- Popularity
- Large Community
- Speed
- Versatility
- Interoperability

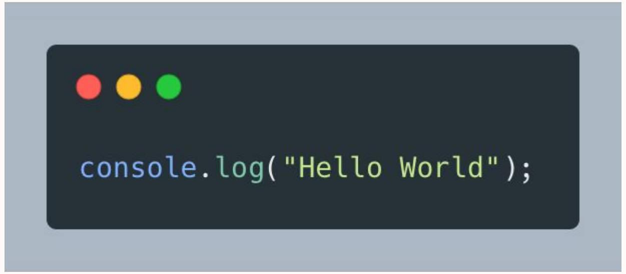
Setting Up the Development Environment

Install the following tools & extensions :

- Visual Studio Code
  - IDE / Code editor
- QuokkaJS
  - VS Code Extension, to run your code with instant result / feedback
- Git
  - Source code management

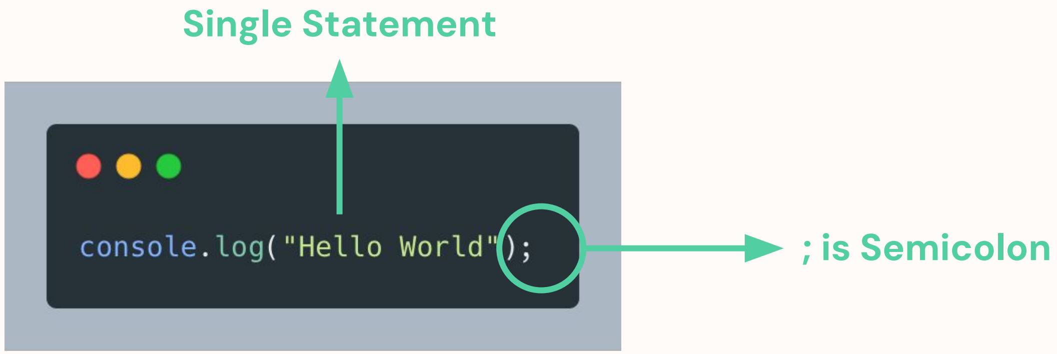
Hello World !

Let’s write our first code !

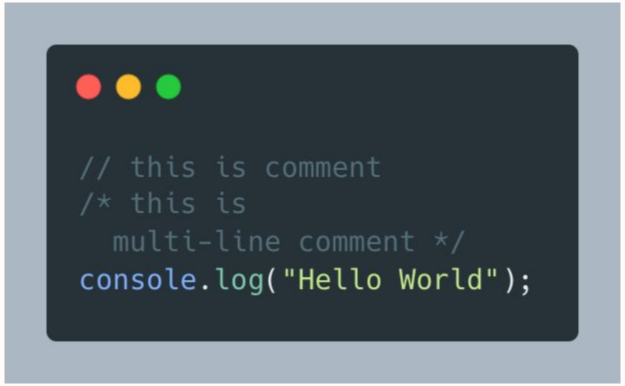


```
console.log("Hello World");
```

Javascript Code Structure



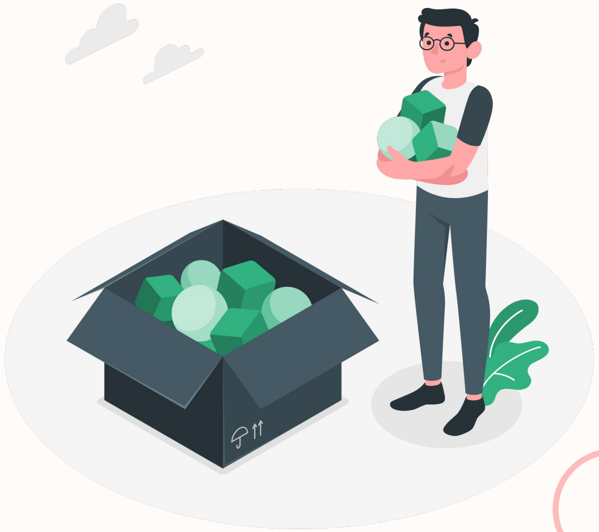
```
console.log("Hello World");
```



```
// this is comment
/* this is
  multi-line comment */
console.log("Hello World");
```

Variable

Variable is a **“named storage”** for data. We can use variable **to store any data** you need.

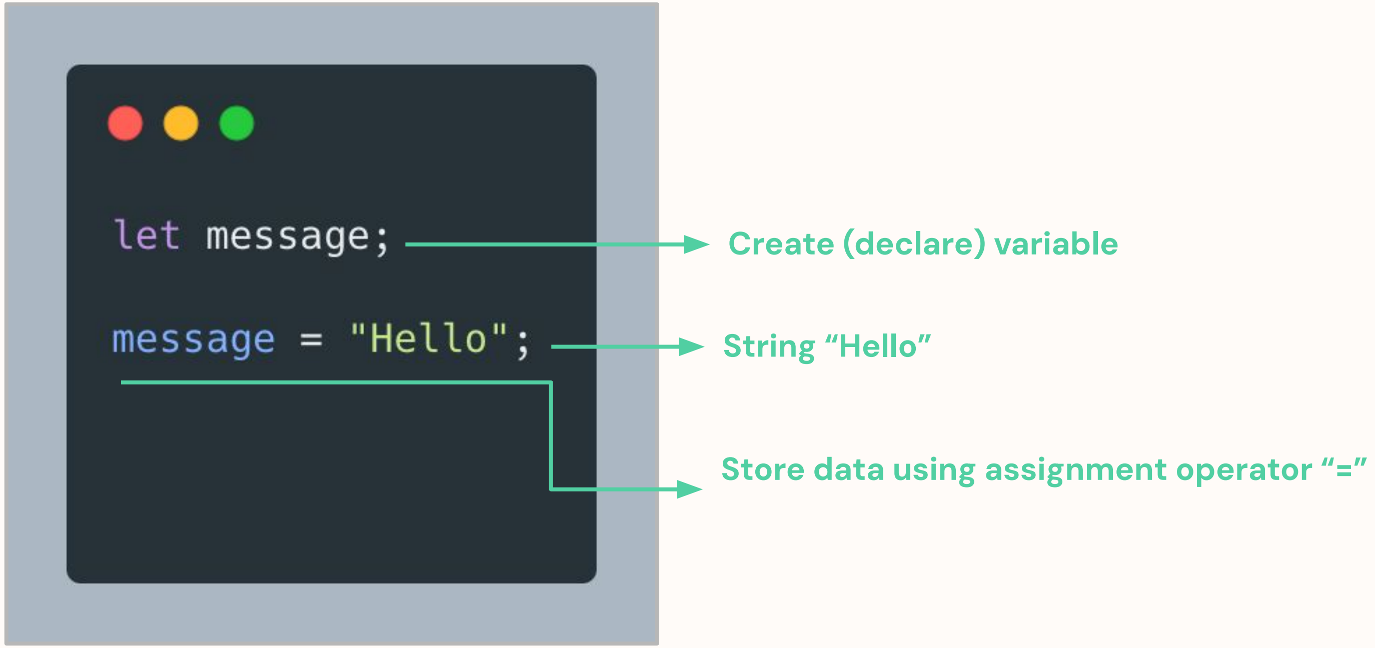


Example

In package delivery apps, there’s information about package details, address, sender’s name, etc.

Variable are used to store all the information.

Code Example



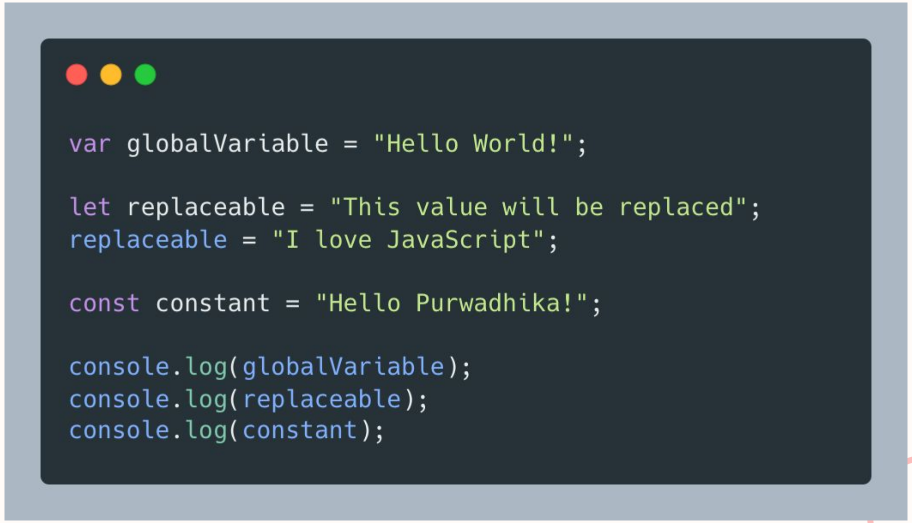
```
let message;

message = "Hello";
```

Create (declare) variable
String “Hello”
Store data using assignment operator “=”

Variable Declaration

Different ways to declare variable :

- **var** : To create global variables
- **let** : To create scoped, replaceable variables
- **const** : Can't be updated or redeclared within the scope



```
var globalVariable = "Hello World!";

let replaceable = "This value will be replaced";
replaceable = "I love JavaScript";

const constant = "Hello Purwadhika!";

console.log(globalVariable);
console.log(replaceable);
console.log(constant);
```

Variable Naming

- Must contain only letters, digits, or the symbols “$” and “_”
- The first character must not digit
- Case-sensitive
- Can’t use reserved words

Data Types

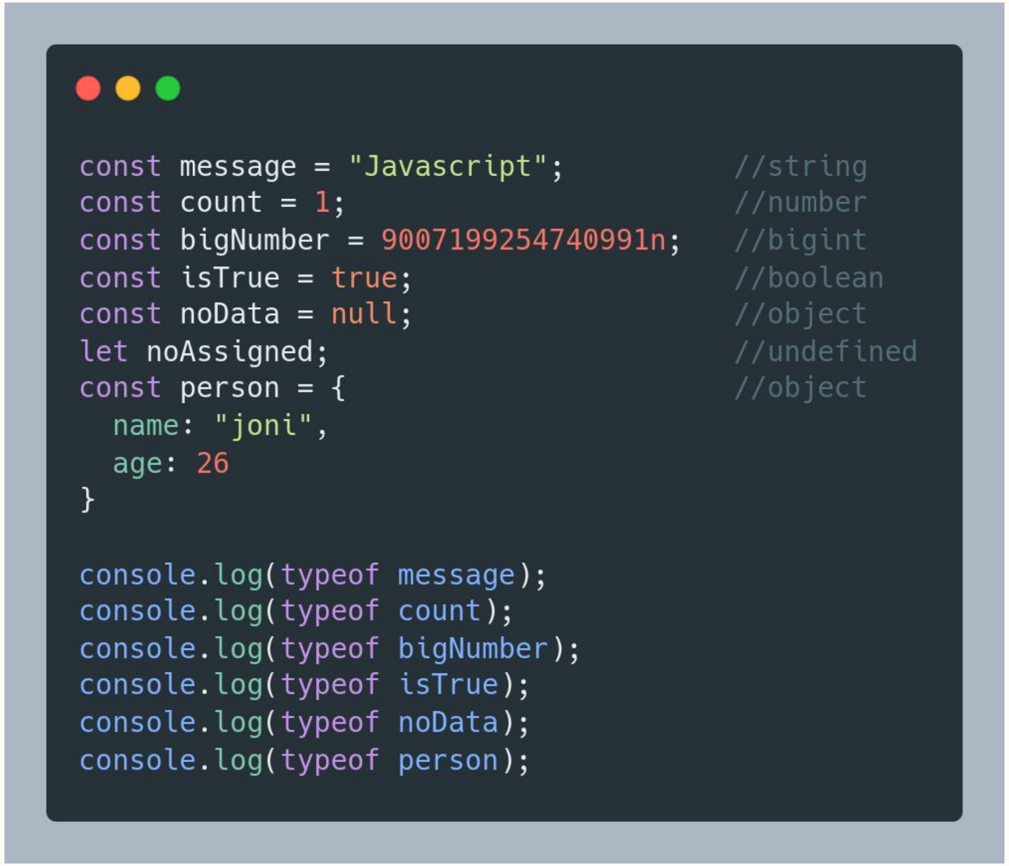
A value in JavaScript is always of a certain **type**.

Primitive data types : The predefined data types provided by JavaScript.
Non-primitive data types : The data types that are derived from primitive data types.

Primitive	
String	Used to represent textual data
Number & BigInt	Used to hold decimal values as well as values without decimals
Boolean	Represents a logical entity and can have two values: **true** and **false**
Null	Has exactly one value: **null**. Represents the intentional absence of any object value
Undefined	A variable that has not been assigned a value has the value **undefined**

Non Primitive	
Object	Is an entity having properties and methods (keyed collection) → Will be explained in the next session
Array	Used to store more than one element under a single variable → Will be explained in the next session

Data Types



```
const message = "Javascript";          //string
const count = 1;                       //number
const bigNumber = 9007199254740991n;   //bigint
const isTrue = true;                   //boolean
const noData = null;                   //object
let noAssigned;                        //undefined
const person = {                       //object
  name: "joni",
  age: 26
}

console.log(typeof message);
console.log(typeof count);
console.log(typeof bigNumber);
console.log(typeof isTrue);
console.log(typeof noData);
console.log(typeof person);
```

What is TypeScript?

TypeScript is a superset of **JavaScript**, which means that it adds additional features to JavaScript, but does not break any existing JavaScript code.

The main feature that TypeScript adds is static typing, which allows developers to specify the types of data that variables and functions can hold. This can help to catch errors early in the development process and make code more maintainable.

Why use TypeScript?

There are several benefits to using TypeScript, including:

- **Improved type safety:** TypeScript static type checking can help to catch errors early in the development process, which can save time and frustration.
- **Better code readability:** TypeScript type annotations can make code more readable and easier to understand.
- **Increased developer productivity:** TypeScript can help developers to be more productive by catching errors early and making code more maintainable.

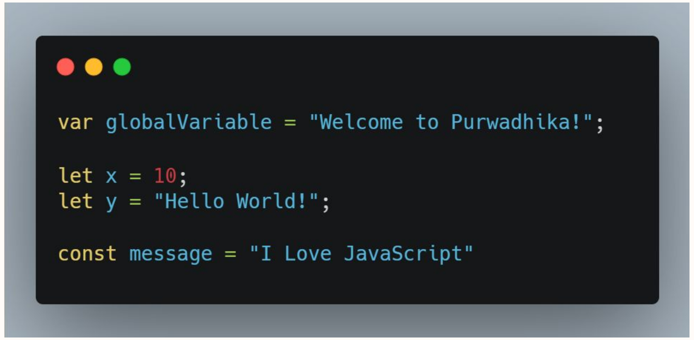
Getting started with TypeScript

To get started with TypeScript, you can follow these steps:

- Install TypeScript: You can install TypeScript using npm or yarn.
- Create a TypeScript file: Create a file with a .ts extension.
- Write TypeScript code: You can write TypeScript code using the same syntax as JavaScript.
- Compile TypeScript code: You can compile TypeScript code into JavaScript code using the TypeScript compiler.
- Reference : https://www.typescriptlang.org/docs/handbook/typescript-tooling-in-5-minutes.html

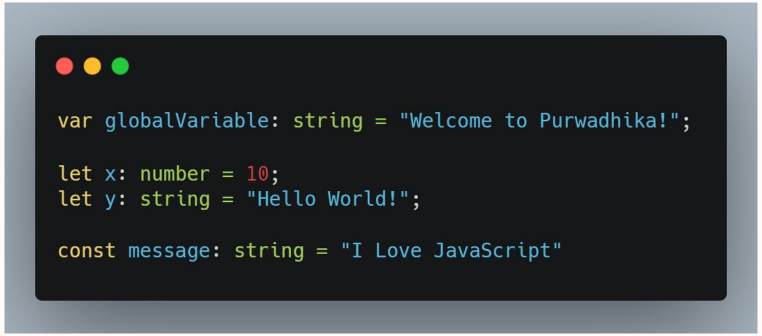
TypeScript VS JavaScript

JavaScript Example



```
var globalVariable = "Welcome to Purwadhika!";

let x = 10;
let y = "Hello World!";

const message = "I Love JavaScript"
```

TypeScript Example



```
var globalVariable: string = "Welcome to Purwadhika!";

let x: number = 10;
let y: string = "Hello World!";

const message: string = "I Love JavaScript"
```

String Built-in Method

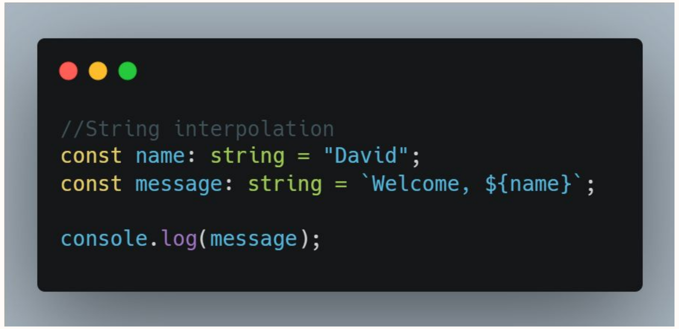
- slice
- substring
- substr
- replace
- toUpperCase
- toLowerCase
- concat
- trim

- padStart
- padEnd
- chartAt
- charCodeAt
- split
- indexOf
- lastIndexOf
- search

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Template Literals

- **Template literals** (template strings) allow you to use strings or embedded expressions in the form of a string.
- Template literals are enclosed by backtick (`) characters instead of double or single quotes.
- With template literals, you can get :
  - A **multiline string** ➜ a string that can span multiple lines.
  - **String formatting** ➜ the ability to substitute part of the string for the values of variables or expressions. This feature is also called **string interpolation**.
  - **HTML escaping** ➜ the ability to transform a string so that it's safe to include in HTML.



```
//String interpolation
const name: string = "David";
const message: string = `Welcome, ${name}`;

console.log(message);
```

Number Built-in Method

Number built-in method

- toString
- toExponential
- toFixed
- toPrecision
- valueOf

Global built-in method & property

- Number
- parseInt
- parseFloat

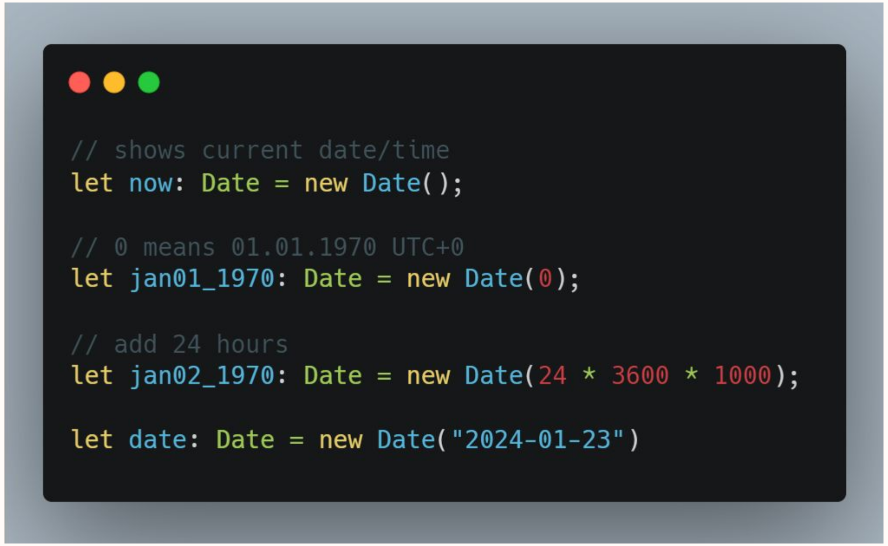
- MAX_VALUE
- MIN_VALUE
- POSITIVE_INFINITY
- NEGATIVE_INFINITY
- NaN

Type Conversion

- String Conversion
  - `String(123)                  // return a string from a number literal 123`
- Numeric Conversion
  - `const num = "3" * "3"// return 9 in number`
  - `Number("3.14")               // return 3.14 in number`
- Boolean Conversion
  - `Boolean(1)                   // return true`
  - `Boolean(0)                   // return false`
  - `Boolean("Hello")             // return true`
  - `Boolean("")                  // return false`

Date Data Type

It stores the date, time and provides methods for date/time management.



```
// shows current date/time
let now: Date = new Date();

// 0 means 01.01.1970 UTC+0
let jan01_1970: Date = new Date(0);

// add 24 hours
let jan02_1970: Date = new Date(24 * 3600 * 1000);

let date: Date = new Date("2024-01-23")
```

Date Built-in Method

Get Methods

- getFullYear
- getMonth
- getDate
- getHours
- getMinutes
- getSeconds
- getMilliseconds
- getTime
- getDay
- Date.now
- Date.parse

Set Methods

- setDate
- setFullYear
- setHours
- setMilliseconds
- setMinutes
- setMonth
- setSeconds
- setTime

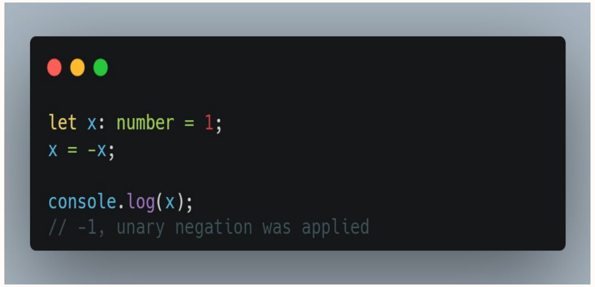
Basic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder (modulo)
**	Exponentiation

Unary, Binary and Operand

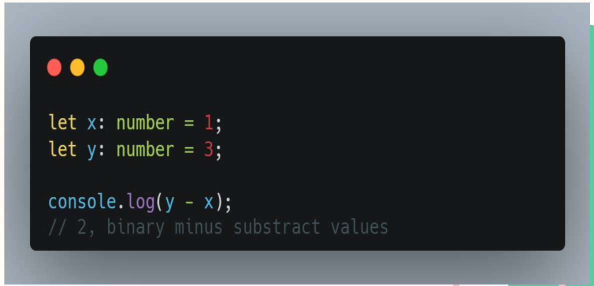
- An **operand** is what operators are applied to. For instance, in the multiplication of 5 * 2 there are two operands: the left operand is 5 and the right operand is 2. Sometimes, people call these “arguments” instead of “operands”.

- An **operator** is **unary** if it has a single operand. For example, the unary negation - reverses the sign of a number.



```
let x: number = 1;
x = -x;

console.log(x);
// -1, unary negation was applied
```

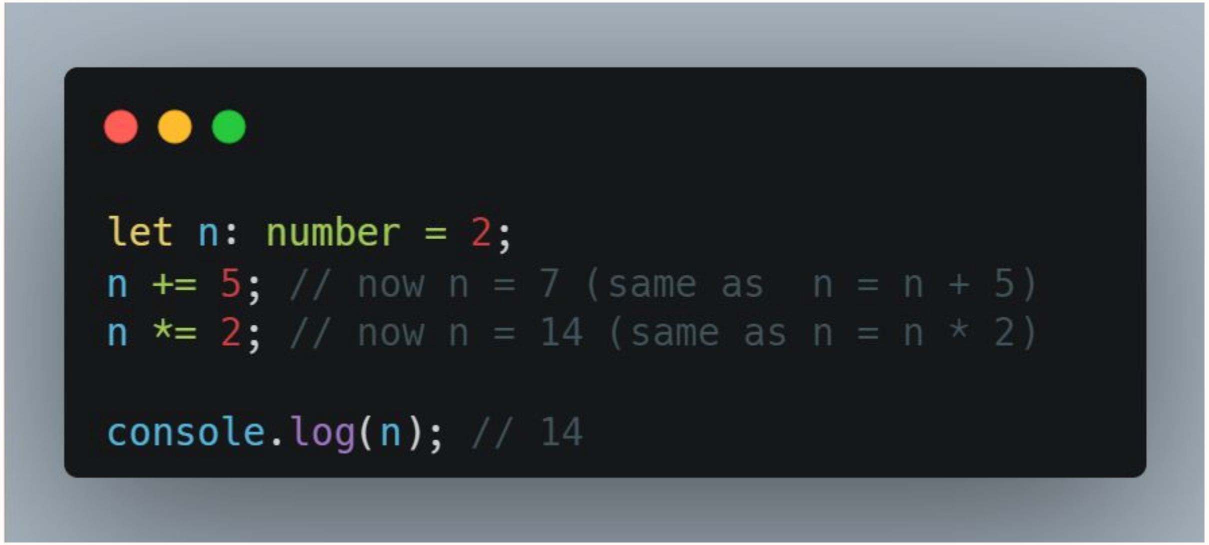
- An **operator** is **binary** if it has two operands. The same minus exists in binary form as well.



```
let x: number = 1;
let y: number = 3;

console.log(y - x);
// 2, binary minus substract values
```

Modify in Place

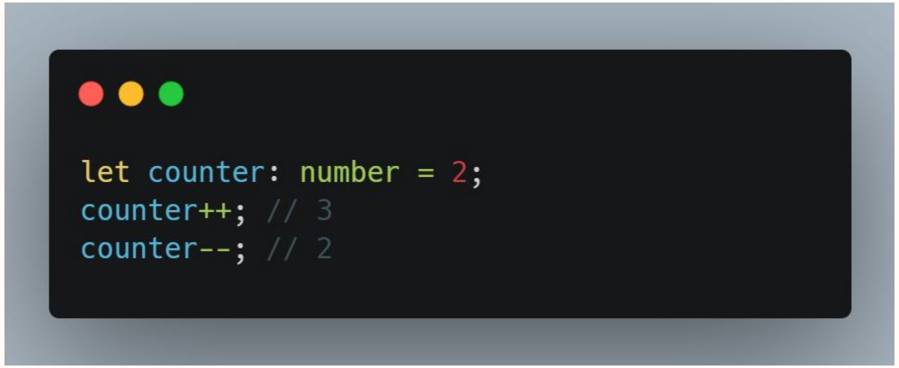
We often need to apply an operator to a variable and store the new result in that same variable.



```
let n: number = 2;
n += 5; // now n = 7 (same as  n = n + 5)
n *= 2; // now n = 14 (same as n = n * 2)

console.log(n); // 14
```

Increment & Decrement

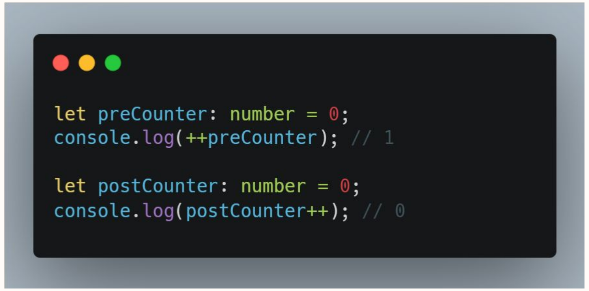
- Increasing or decreasing a number by one is among the most common numerical operations.
- Increment ++ increases a variable by 1.
- Decrement -- decreases a variable by 1



```
let counter: number = 2;
counter++; // 3
counter--; // 2
```

Postfix & Prefix Form

- The operators **++** and **--** can be placed either before or after a variable.
- When the operator goes after the variable, it is in "**postfix form**": counter++.
- The "**prefix form**" is when the operator goes before the variable: ++counter.
- If we'd like to increase a value and immediately use the result of the operator, we need the prefix form
- If we'd like to increment a value but use its previous value, we need the postfix form

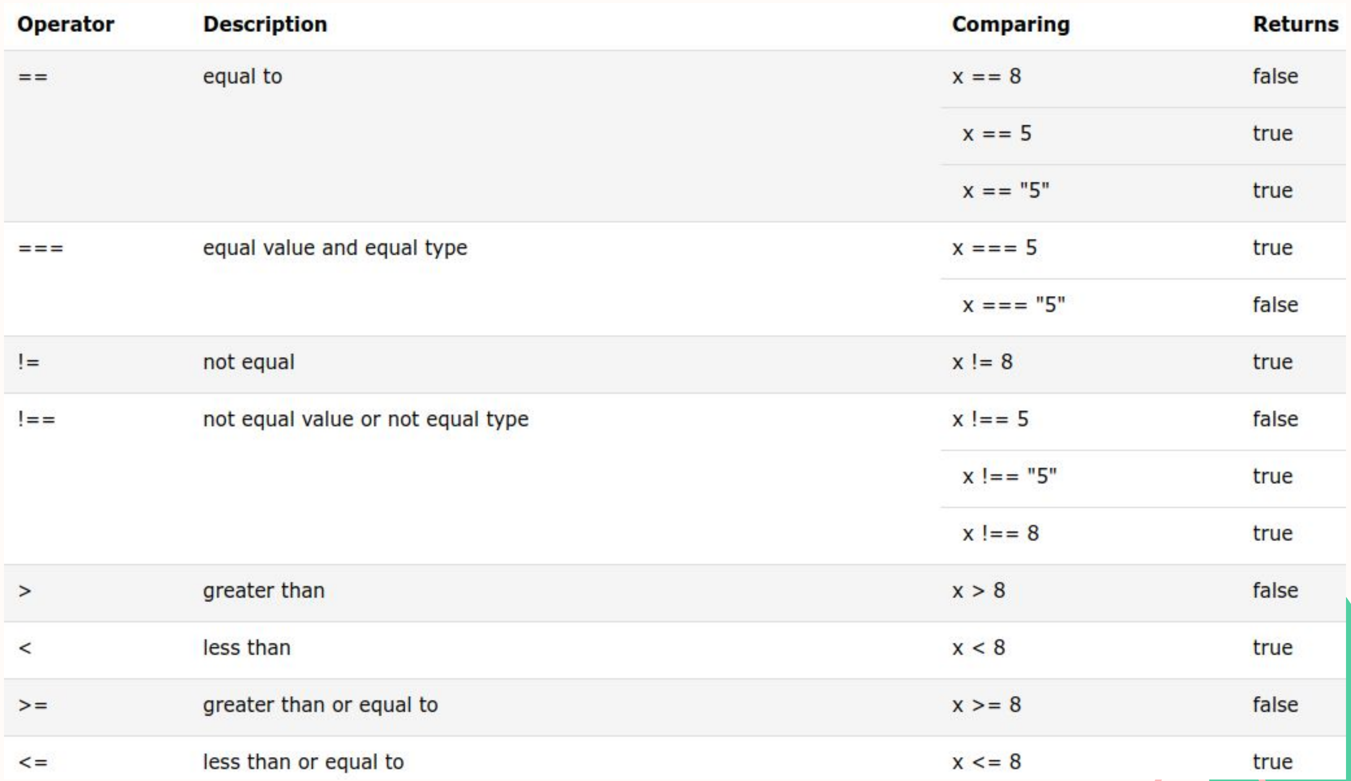


```
let preCounter: number = 0;
console.log(++preCounter); // 1

let postCounter: number = 0;
console.log(postCounter++); // 0
```

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x = 5, this table would explains the comparison operators



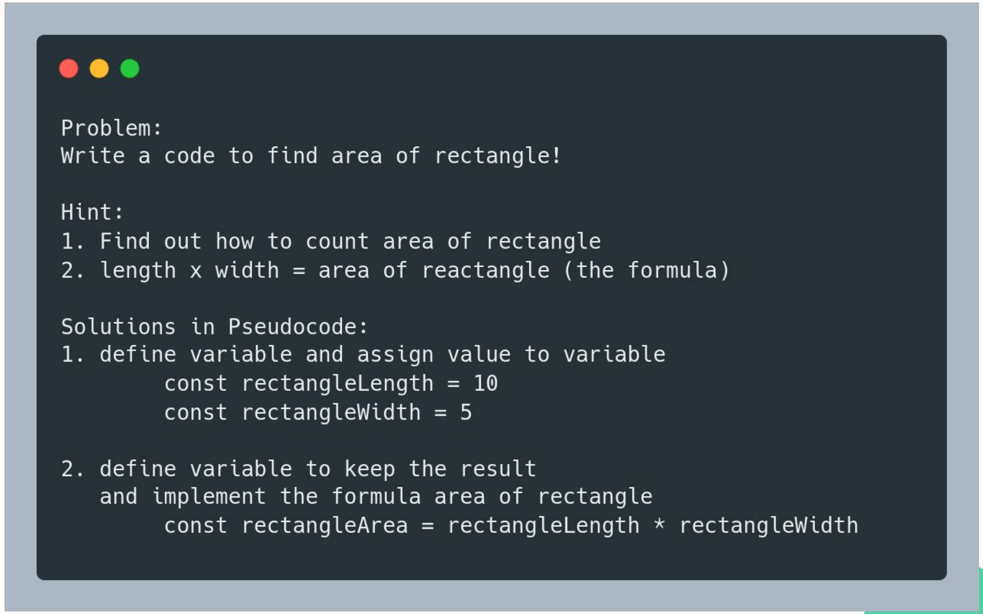
Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Introduction to Pseudocode

Before we start our exercise on the next slides, let’s talk about Pseudocode first.

Pseudocode is basically an “easier-to-understand” version of programming languages, usually in the form of simple natural language.

Through pseudocode, you can express detailed step-by-step process in a much simpler language.



```
Problem:
Write a code to find area of rectangle!

Hint:
1. Find out how to count area of rectangle
2. length x width = area of reactangle (the formula)

Solutions in Pseudocode:
1. define variable and assign value to variable
        const rectangleLength = 10
        const rectangleWidth = 5

2. define variable to keep the result
   and implement the formula area of rectangle
        const rectangleArea = rectangleLength * rectangleWidth
```

Exercise

Since you already know about pseudocode, let's solve this exercise through pseudocode first, and then convert it into a programming code!

- Write a code to find area of rectangle.
    - Example : length = 5, width = 3
    - Output : 15
- Write a code to find perimeter of rectangle.
    - Example : length = 5, width = 3
    - Output : 16
- Write a code to find diameter, circumference and area of a circle.
    - Example : radius = 5
    - Output : diameter = 10, circumference = 31.4159, area = 78.539
- Write a code to find angles of triangle if two angles are given.
    - Example : a = 80, b = 65
    - Output : 35
- Write a code to convert days to years, months and days (Notes: 1 year : 365 days, 1 month : 30 days).
    - Example : 400 days → 1 year, 1 month, 5 days
    - Example: 366 days → 1 year, 0 month, 1 day
- Write a code to get difference between dates in days.
    - Example : date1 = 2022-01-20, date2 = 2022-01-22
    - Output : 2

Thank You!