

**Full Stack AI Software Development**

Intro to Software  
Development and  
Programming Fundamental

[Job Connector Program](#)

# Outline

## **Foundations of Software Development**

The basics of how software is built and how programming works to instruct computers.

## **Programming with JavaScript & TypeScript**

Get to know JavaScript & Typescript, the main language for creating interactive websites.

## **Problem-Solving & Logic**

Discover how to design step-by-step solutions for problems using logical thinking.

## **Core Programming Concepts**

Essential building blocks of coding, including variables, data types, type conversion, and operators.

# What is Software Development?

The process of designing, creating, testing, and maintaining software applications.

Examples in daily life:

- WhatsApp, Instagram → Mobile apps
- Google Chrome → Web browser
- Microsoft Word → Desktop application

Analogy: Software is like a recipe, developers write “instructions” (code) so computers know what to do.



# Why Learn Software Development?



## Career Growth

High demand in almost every industry.

## Problem Solving

Automate tasks, analyze data, and improve processes.

## Creativity

Build apps, websites, or even games.

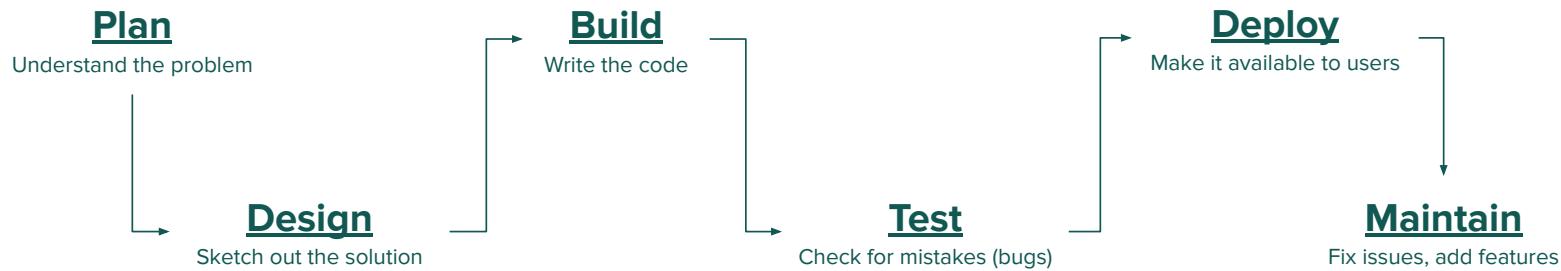
## Digital Literacy

Understand how modern technology works.

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

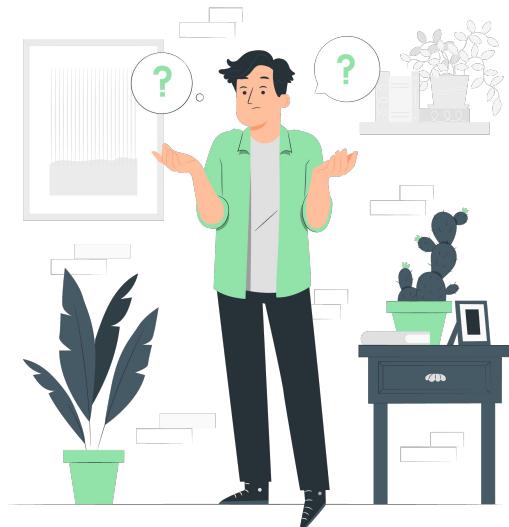
**Steve Jobs**

# Software Development Process (Simplified)



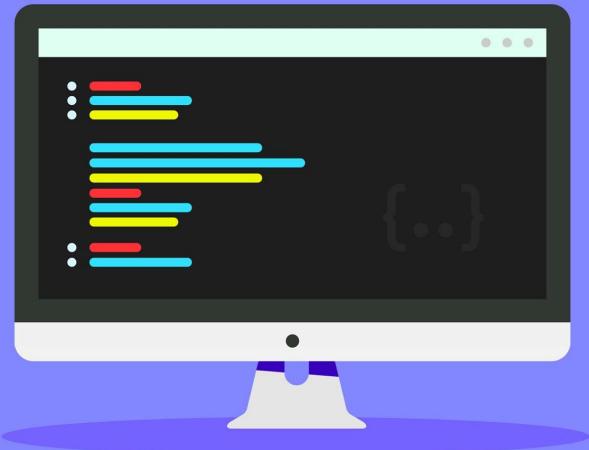
# Common Myths About Software Development

- ✗ “You need to be a math genius” → Only basic logic is required.
- ✗ “It’s only for young people” → People switch careers at any age.
- ✗ “You’ll work alone” → Development is team-based.



# WEB {..} DEVELOPMENT

---



## Our Focus: Web Development

### Why Web Development?

- Websites are the foundation of the digital world
- Every business, service, and community needs a web presence
- Skills are practical, in-demand, and beginner-friendly

### What You'll Learn?

- Programming Fundamentals → Core logic & problem-solving
- Frontend → How websites look (HTML, CSS, JavaScript)
- Backend → How websites work (servers, logic, databases)

# What is the Web & Web Development?

## Web

- The World Wide Web is a collection of websites connected through the Internet.
- You access it using a browser (Chrome, Safari, Edge).
- Example: Shopping online, reading news, social media.
- Analogy:
  - The Web = A big city
  - Websites = Houses/buildings
  - Browser = Your car to visit them

## Web Development

- The process of building websites & web apps.
- Involves three main parts:
  - Frontend (look & feel) → What users see
  - Backend (logic & server) → How it works
  - Database (storage) → Where information lives

To build websites, we first need to understand the language of computers. That's where **Programming** comes in.

# What is Programming ?

**Telling a computer what to do using a language it understands.**

- **Programming** is the process of creating a set of instructions that tell computer to perform a task.
- **Programming language** is a vocabulary and set of grammatical rules for instructing computers.
- Examples: JavaScript, Java, Golang, PHP, C, C++, C#

To do it well, we need a clear step-by-step plan → an **Algorithm**.

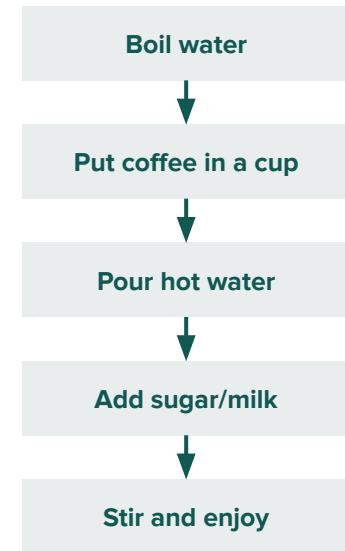


# What is an Algorithm?

An **algorithm** is simply a **step-by-step set of instructions to solve a problem or complete a task**. You already use algorithms in daily life without realizing it!

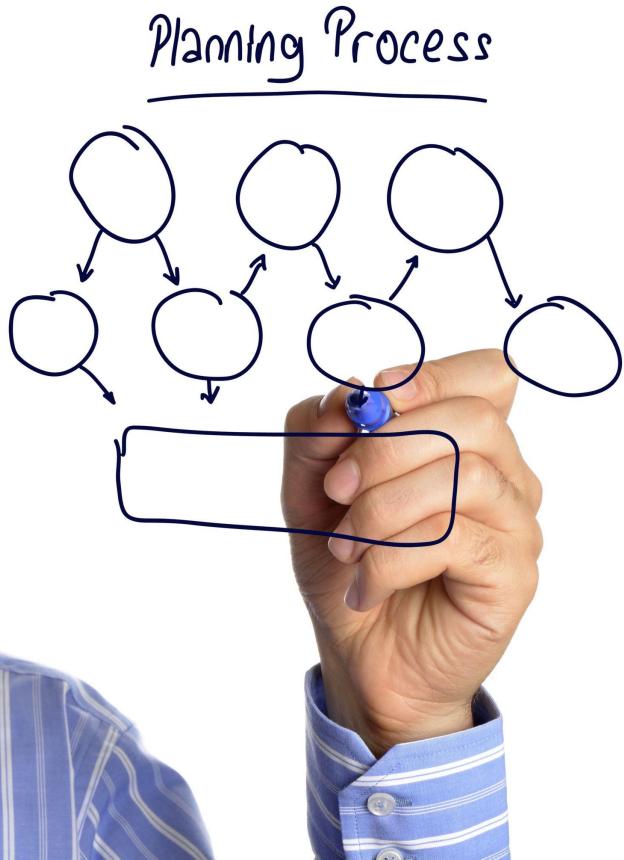


**Example (Making a cup of coffee)**



This is an **algorithm** for coffee-making

# Why Are Algorithms Important?



## Example in daily life:

- Google Maps uses algorithms to find the fastest route to your destination.
- Online shopping platforms use algorithms to recommend products.

# Key Characteristics of an Algorithm

## 01. Clear and unambiguous

Each step must be clear and easy to follow

## 02. Step-by-step process

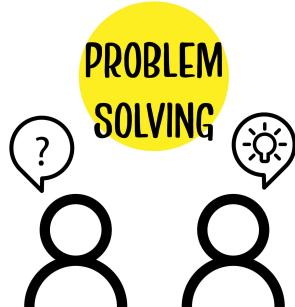
Instructions are performed in order

## 03. Definite start and end

It must eventually stop with a solution

## 04. Effective

Should solve the problem correctly



# Simple Representation of Algorithms

## Plain Language (Step List)

Example: Finding the biggest number from 3 numbers

- Compare **number1** with **number2**.
- Take the bigger one.
- Compare it with **number3**.
- The biggest one is the result.

## Flowchart

Visual diagrams with arrows showing the steps.

# About Flowchart

- A flowchart is a graphical representation of steps in sequential order.
- Used to present algorithms, workflows, or processes.
- Helps visualize complex processes and clarify structure.

## Common Flowchart Symbols



**Terminator**

Starting or ending point of the system.



**Process**

A box indicates some particular operation.



**Document**

This represents a printout, such as a document or a report.



**Decision**

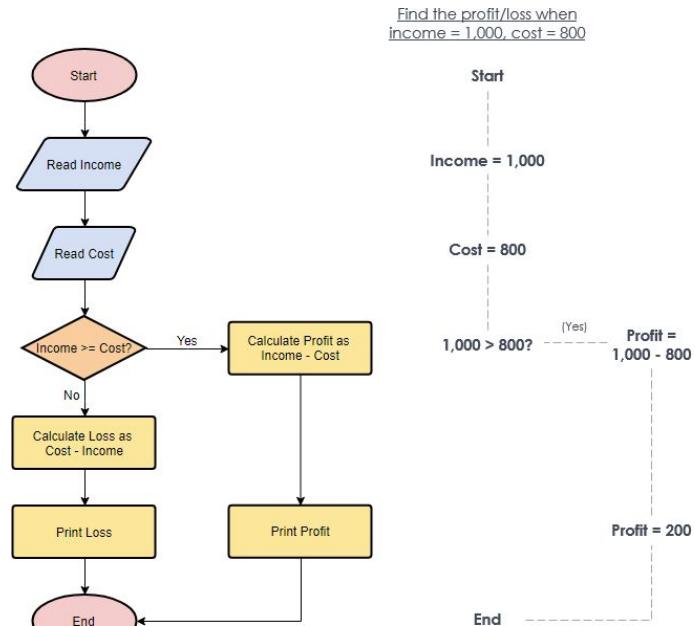
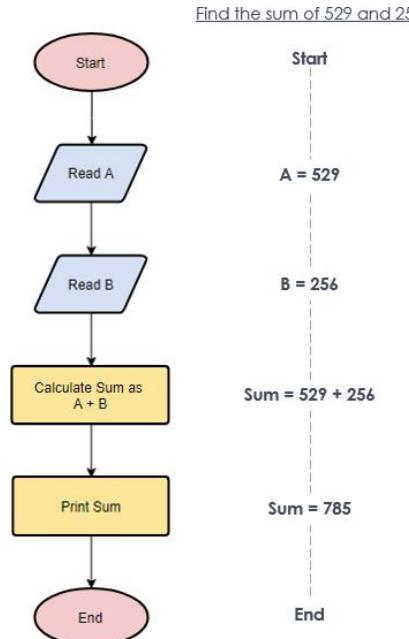
A decision or branching point; leads to different flows.



**Data**

Information entering or leaving the system (input/output).

# Examples of Flowcharts



# Exercise

## Daily Life Algorithms

Task: Write down the step-by-step algorithm for the following:

- Brushing your teeth
- Ordering food online
- Crossing the street safely

## Decision-Making Algorithm

Task: Write an algorithm for deciding what to wear based on the weather:

- If it's raining → bring an umbrella
- If it's sunny → wear sunglasses
- If it's cold → wear a jacket

# Exercise - Tower of Hanoi

- The Tower of Hanoi is a classic problem in computer science and mathematics.
- You have 3 pegs (rods): Source, Auxiliary, and Target.
- You have n disks stacked on the Source peg in decreasing size (largest at bottom, smallest at top).
- Goal: Move all disks from Source to Target.
- <https://www.mathsisfun.com/games/towerofhanoi.html>

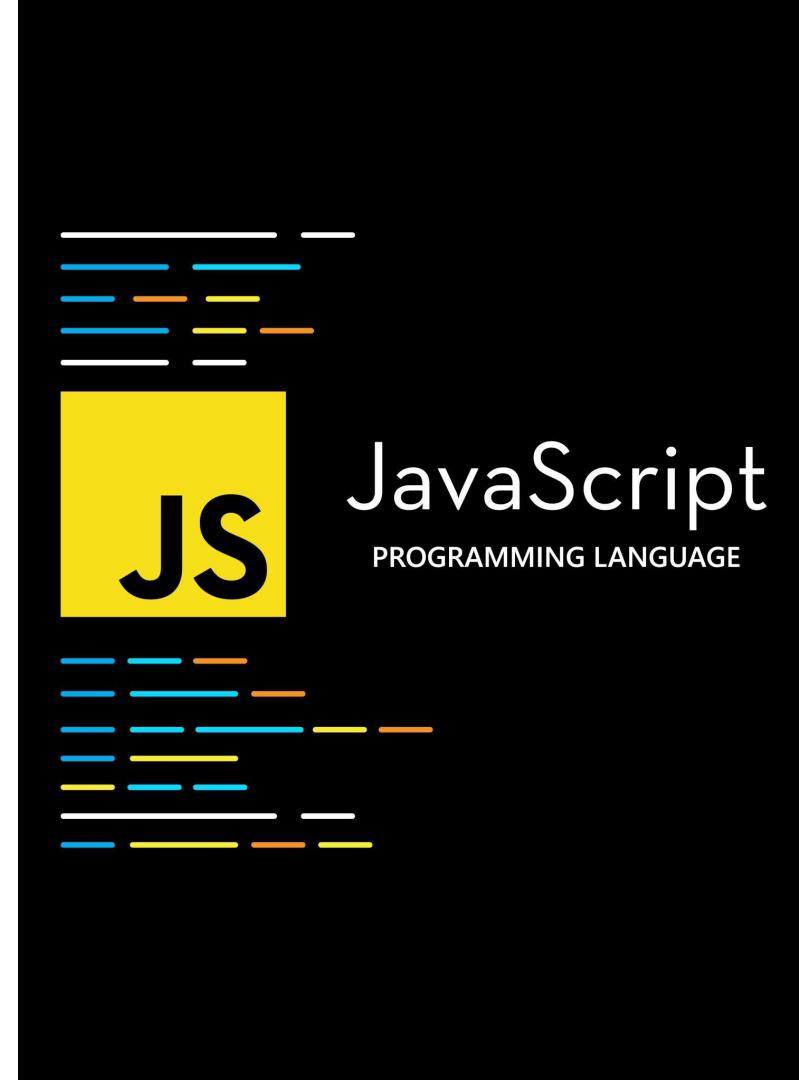
## Rules:

- Only one disk can be moved at a time.
- You can only move the top disk of a peg.
- No disk may be placed on top of a smaller disk.

# Introduction to JavaScript

## What is JavaScript?

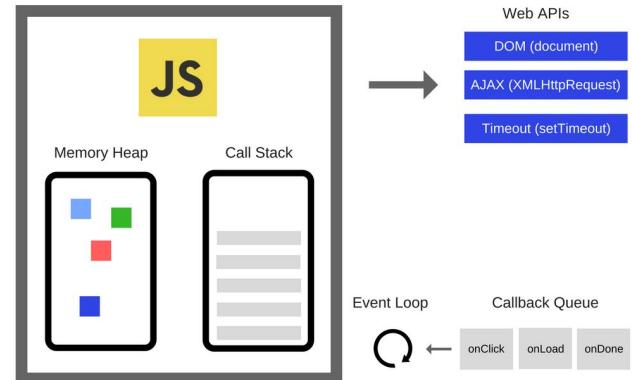
- A programming language mainly used in web development.
- Makes websites interactive (animations, buttons, forms, games).
- Runs inside the browser (Chrome, Safari, Firefox, etc.).
- Analogy:
  - HTML = structure of a house
  - CSS = design/decoration of the house
  - JavaScript = electricity that makes things work



# Deep Dive into JavaScript

- JavaScript 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.
- 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](#)

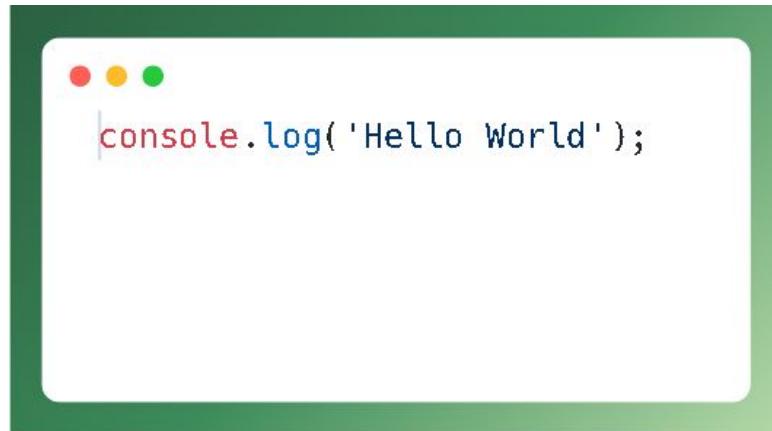


# Setting Up the Development Environment

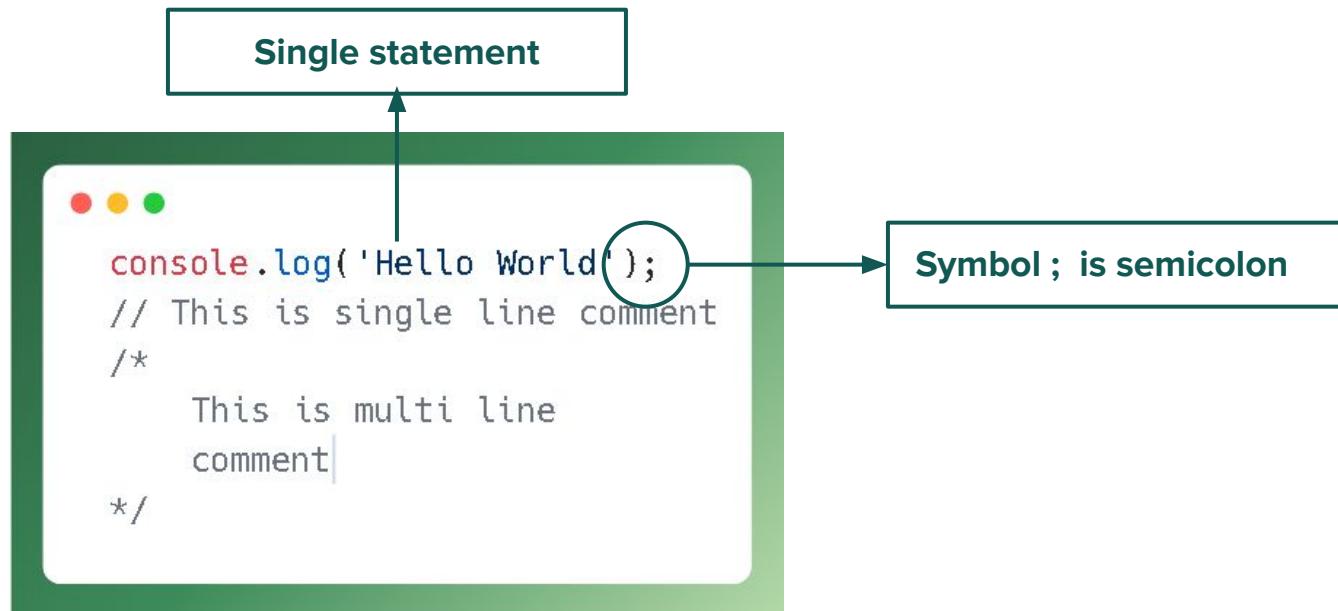
Install the following tools & extensions :

- IDE / Code editor: [Visual Studio Code](#)
- VS Code Extension: [QuokkaJS](#)
- Source code management: [Git](#)

# Let's Write Our First Code!



# Javascript Code Structure



# Variable

## What is a variable?

A container to store data (like a box with a label).

## Why use variables?

To save and reuse information in a program.

## How to create a variable?

```
let message;  
  
message = "hello";
```



# Code Explanation



# 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 replaceableVariable = 'This string will be replaced';  
replaceableVariable = 'The new string';  
  
const constantVariable = 'Purwadhika School';  
  
console.log(globalVariable);  
console.log(replaceableVariable);  
console.log(constantVariable);
```

# 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		Non Primitive	
<b>String</b>	Used to represent textual data	<b>Object</b>	Is an entity having properties and methods (keyed collection) → Will be explained in the next session
<b>Number &amp; BigInt</b>	Used to hold decimal values as well as values without decimals	<b>Array</b>	Used to store more than one element under a single variable → Will be explained in the next session
<b>Boolean</b>	Represents a logical entity and can have two values: <b>true</b> and <b>false</b>		
<b>Null</b>	Has exactly one value: <b>null</b> . Represents the intentional absence of any object value		
<b>Undefined</b>	A variable that has not been assigned a value has the value <b>undefined</b>		

# Data Types

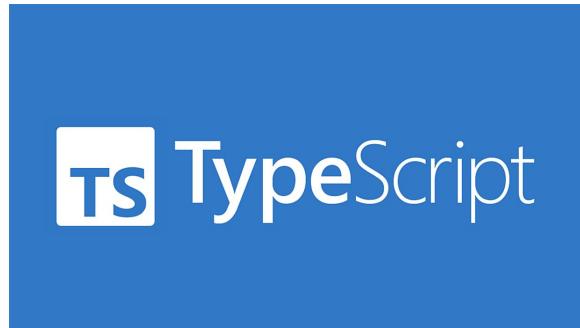
```
const programming = 'Javascript';
const text = "Welcome All";
const description = `lorem ipsum dolor amet...`;
const count = 1;
const bigNumber = 900719925474099n;
const isGraduated = true;
const isMarried = false;
const emptyData = null;
let noAssigned;

console.log(typeof programming); // String
console.log(typeof text); // String
console.log(typeof description); // String
console.log(typeof count); // Number
console.log(typeof bigNumber); // Bigint
console.log(typeof isGraduated); // Boolean
console.log(typeof isMarried); // Boolean
console.log(typeof emptyData); // Object
console.log(typeof noAssigned); // Undefined
```

# 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

```
● ● ●  
const productName = 'Indomie Goreng';  
let price = 2950;  
var isExpiry = false;
```

## TypeScript Example

```
● ● ●  
const productName: string = 'Indomie Goreng';  
let price: number = 2950;  
var isExpiry: boolean = false;
```

# String Built-in Method

- slice
- substring
- substr
- replace
- toUpperCase
- toLowerCase
- concat
- trim
- padStart
- padEnd
- charAt
- charCodeAt
- split
- indexOf
- lastIndexOf
- search

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/String](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.

```
● ● ●  
const fullName: string = 'Aboy';  
const age: number = 25;  
  
// Regular string (using single quote '' or double quote "")  
const message1: string =  
  'Hi, my name is ' + fullName + ' and I am ' + age + ' years old.';  
  
// Template literal  
const message2: string = `Hi, my name is ${fullName} and I am ${age}  
  years old.`;  
  
console.log(message1);  
console.log(message2);
```

# 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.

```
// Declare a date with current time
const now: Date = new Date();
console.log('Current Date & Time:', now);

// Declare a specific date
const birthday: Date = new Date('1998-05-12');
console.log('Birthday:', birthday);

// Create date using year, month (0-based), day
const customDate: Date = new Date(2025, 7, 5); // August 5, 2025
console.log('Custom Date:', customDate);
```

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



```
let count: number = 3;
count = -count;
console.log(count);
// -1, unary negation was applied
```



```
let x: number = 1;
let y: number = 3;
console.log(y - x);
// 2, binary minus subtract values
```

# Modify in Place

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



```
n += 5; // Now n = 7 (same as n = n + 5)  
n *= 2; // Now n = 14 (same as n = n * 2)
```

# 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 = 1;  
counter++; // Now counter = 2 (same as counter = counter + 1)  
counter--; // Now counter = 1 (same as counter = counter + 1)
```

# 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

# Variable Hoisting

Hoisting is a mechanism in JavaScript where **variable and function declarations are moved to the top** of their scope **before the code is executed**. This means you can use variables or functions before declaring them in the code. Variables declared **using var** are hoisted to the top but are not initialized. Their value will be undefined if accessed before initialization.

```
● ● ●  
console.log(discount); // Output: undefined  
var discount = 10;  
console.log(discount); // Output: 10  
  
// Actually interpreted by Javascript  
var discount;  
console.log(discount); // Output: undefined  
discount = 10;  
console.log(discount); // Output: 10|
```

# Variable Hoisting

Variables declared with **let** or **const** are also hoisted, but they cannot be accessed before declaration due to the **Temporal Dead Zone (TDZ)**.



```
console.log(b); // ReferenceError: Cannot access 'b' before initialization
let b = 100;
```



```
console.log(c); // ReferenceError: Cannot access 'c' before initialization
let c = 100;
```

# 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:**  
Calculate the area **of** a rectangle given its length and width.

**Hint:**

1. Find out how to count area **of** rectangle
2. The formula to find the area **of** a rectangle is: Area = Length × Width

**Solution:**

1. Define variables and assign **with** value

```
const width = 10;
const length = 5;
```
2. Define **new** variable to assign the result and implement the formula area **of** rectangle

```
const areaOfRectangle = width * length;
```

# Exercise

- 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!

