

LEARNING FUNDAMENTALS OF TYPESCRIPT 5.0+

<https://www.panaverse.co/>

<https://github.com/panaverse/>

<https://www.youtube.com/@panaverse>

<https://www.facebook.com/groups/panaverse>

Daniyal Nagori

About Instructor

- Chief Executive Officer - PIAIC
- Director at Panacloud
- Chief Technical Officer - TravelclubIQ



About Instructor



Developer
ASSOCIATE

AWS Certified Developer – Associate

Issued by [Amazon Web Services Training and Certification](#)

Earners of this certification have a comprehensive understanding of application life-cycle management. They demonstrated proficiency in writing applications with AWS service APIs, AWS CLI, and SDKs; using containers; and deploying with a CI/CD pipeline. Badge owners are able to develop, deploy, and debug cloud-based applications that follow AWS best practices.

Skills

Amazon Web Services

AWS

AWS Certification

AWS Cloud

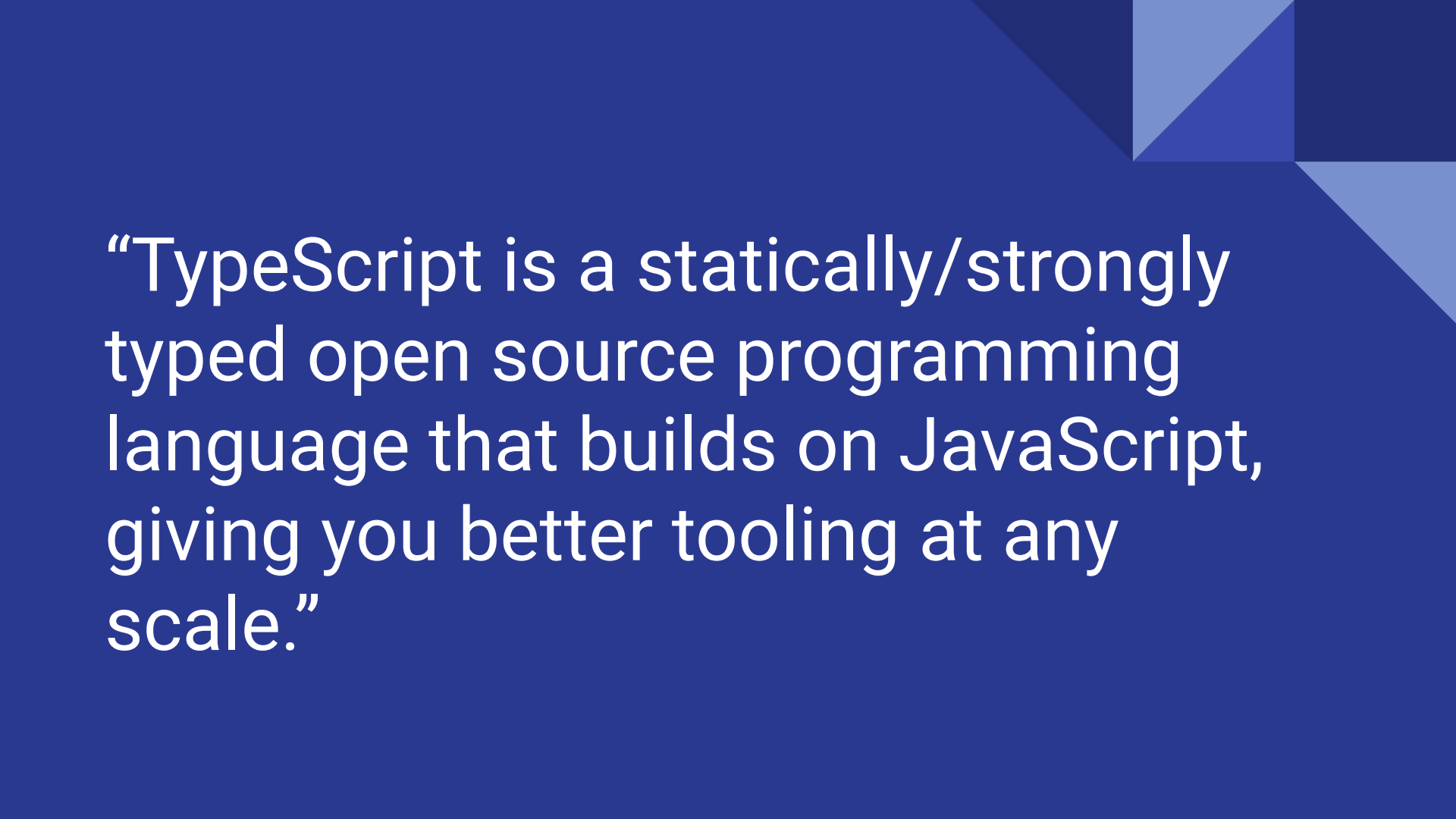
Cloud Certification

Code Deployment

Code Development

Earning Criteria

- Successfully passed the AWS Certified Developer – Associate exam.



“TypeScript is a statically/strongly typed open source programming language that builds on JavaScript, giving you better tooling at any scale.”

Why should you use TypeScript?

- **Static typing.**
 - TypeScript is a superset of JavaScript that adds optional static typing and other features such as classes and modules.
 - TypeScript checks a program for errors before execution, and does so based on the kinds of values, it's a static type checker.
 - Once TypeScript's compiler is done with checking your code, it erases the types to produce the resulting "compiled" code. This means that once your code is compiled, the resulting plain JS code has no type information.
- **Access to latest features(ES6, ES7, etc...), before they become supported by major browsers.**
 - Use cutting edge ECMA features such as `Optional Chaining (?.)` operator without having to worry about browser support.



Why should you use TypeScript?

- Code completion and Intellisense
- Object Oriented Programming
 - Its helps programmers to write code in a object oriented manner if required. Thus helps users to jump into TS
- IDE Support
 - It is super well-supported by text editors including (VS Code, Atom, Sublime, Eclips, etc.)
- Large Community/Adoption
 - TypeScript is made for creating large, complex systems that the modern Web abounds with.





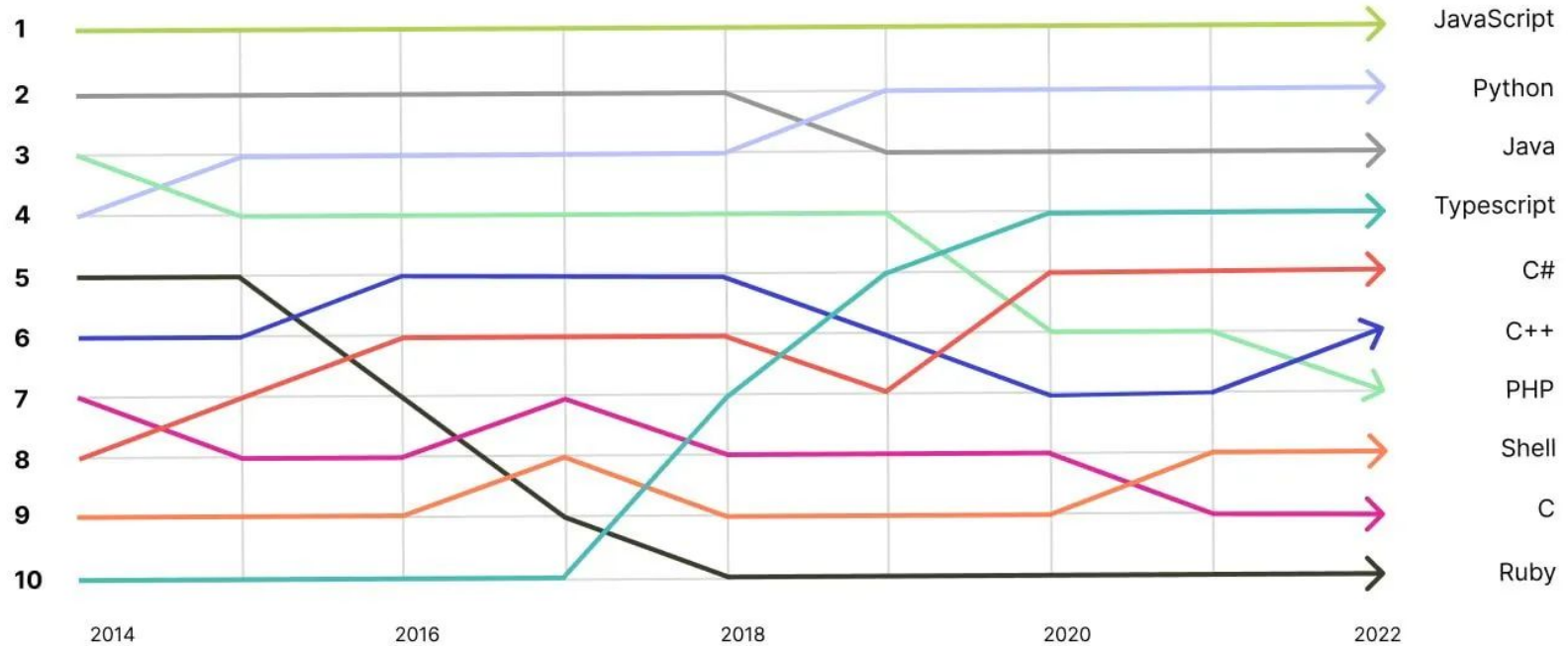
TypeScript's Popularity

TypeScript Is Fastest-Growing Programming Language

"TypeScript's share has almost tripled over the course of 6 years, increasing from 12 percent in 2017 to 34 percent in 2022," said the company's **State of Developer Ecosystem 2022**.

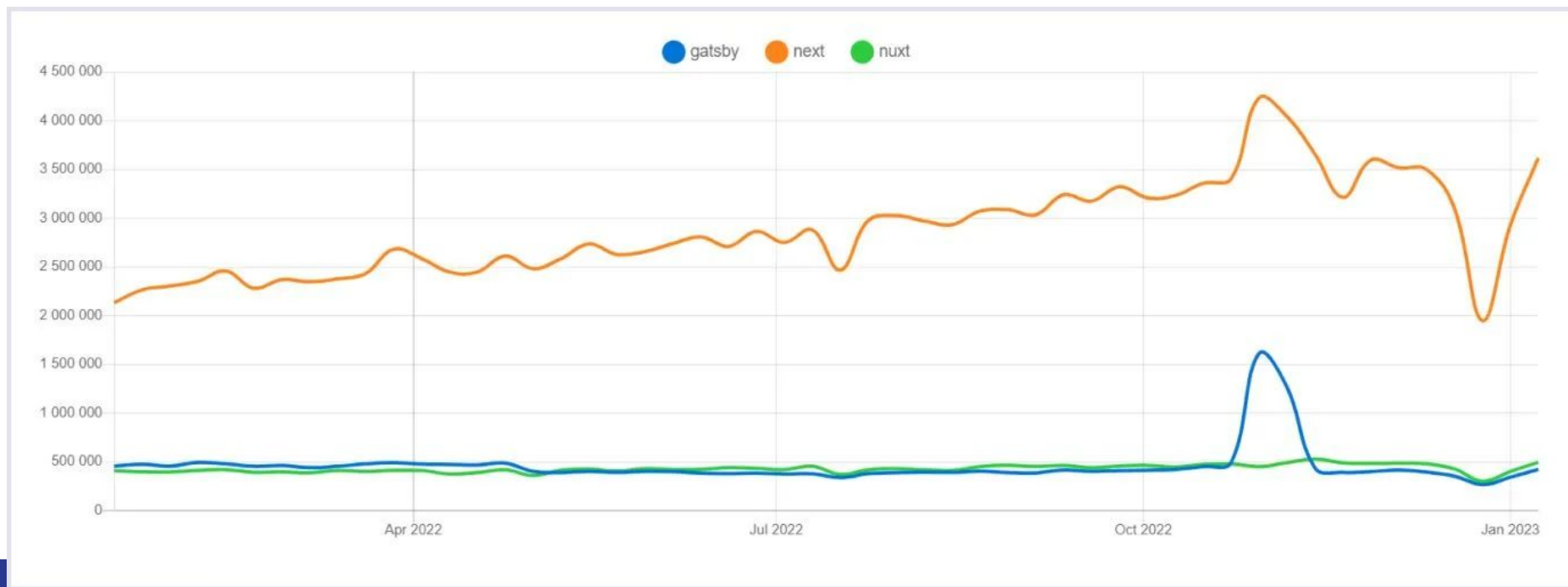
TypeScript passed four languages (C, PHP, C# and C++) over the past six years.

Top programming languages over the years



Next.js 13 Now Supports TypeScript By Default

Next.js is the Most Demanded Full Stack Framework in 2023





Getting Started

Setting up your environment

Setting up your environment

- There are many ways in which you can set up a coding environment. Such as:
 - Integrated Development Environment (IDE). Example: **VS Code**, Sublime Text, Atom, etc.
 - Web browser. Example: **Chrome**, Firefox, etc.
 - Online editor (optional). Example: StackBlitz, Replit, etc.



Install NodeJs and VS Code

<https://nodejs.org/en/download/current/>

Install Version 18.10.0+

node -v

<https://code.visualstudio.com/>



Install Typescript

<https://www.npmjs.com/package/typescript>

```
npm install -g typescript
```

```
tsc -init
```

Create a new file and name it whatever you want or better name it `index.ts` just for convention.

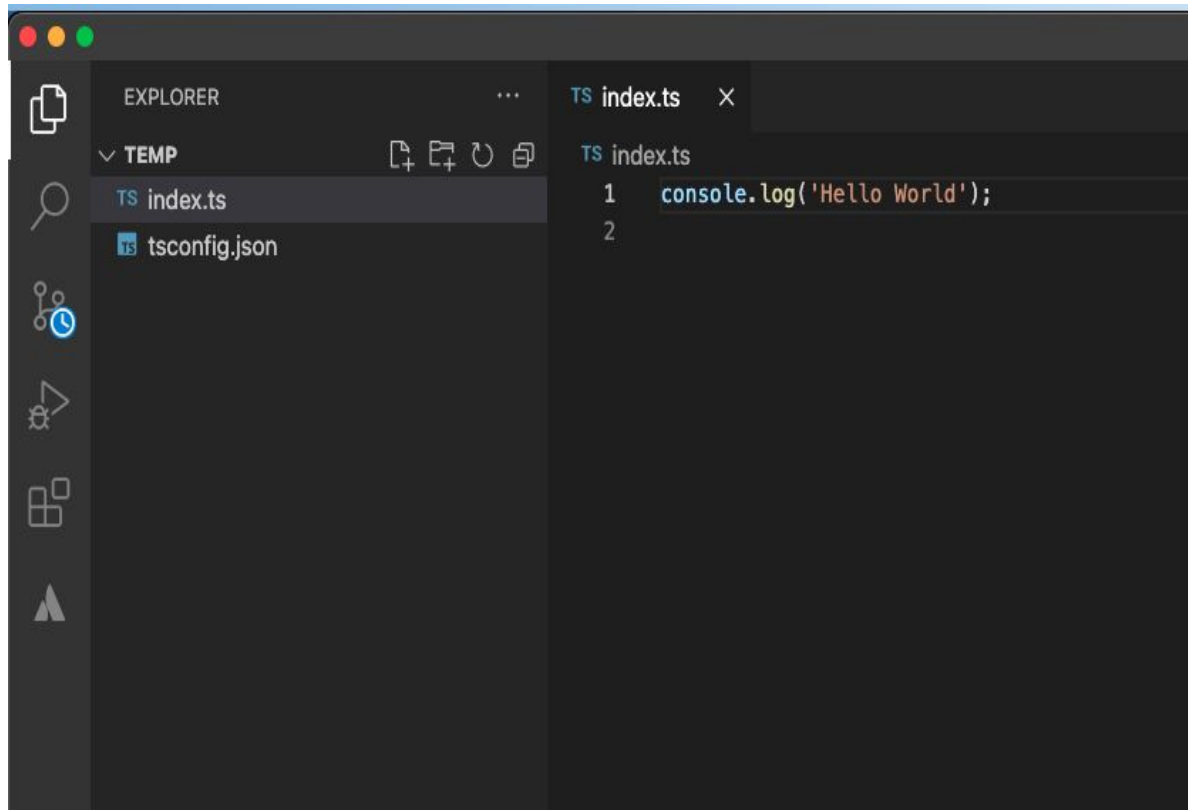
Open your file in any text editor like vscode, notepad etc



First Typescript Program

Write the following code:

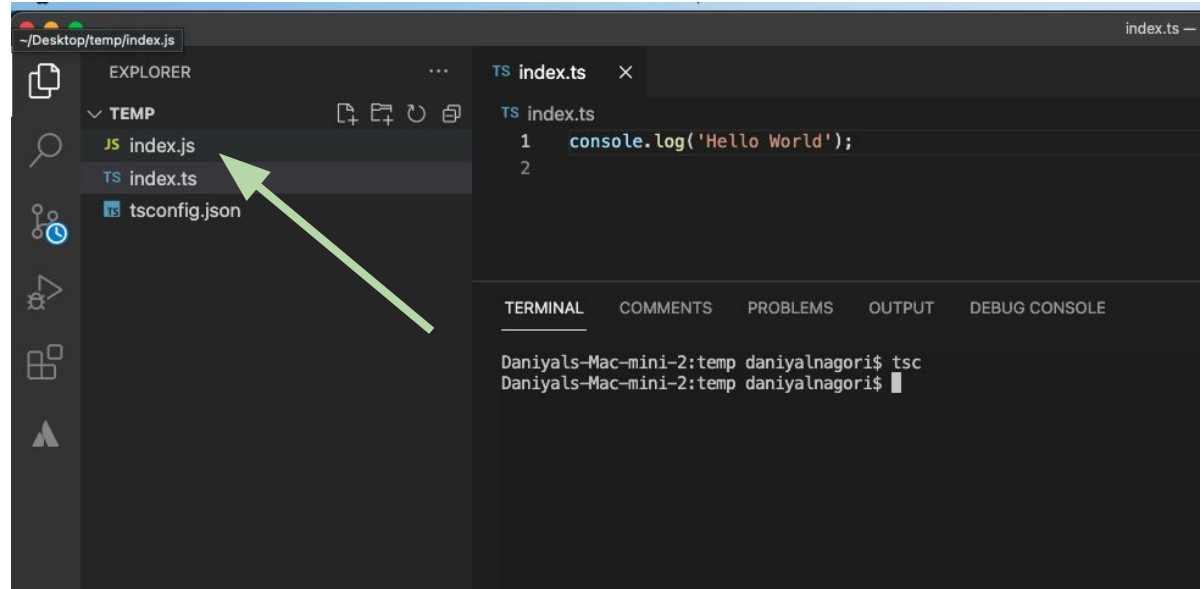
```
console.log("Hello World");
```



Run Typescript Program

tsc

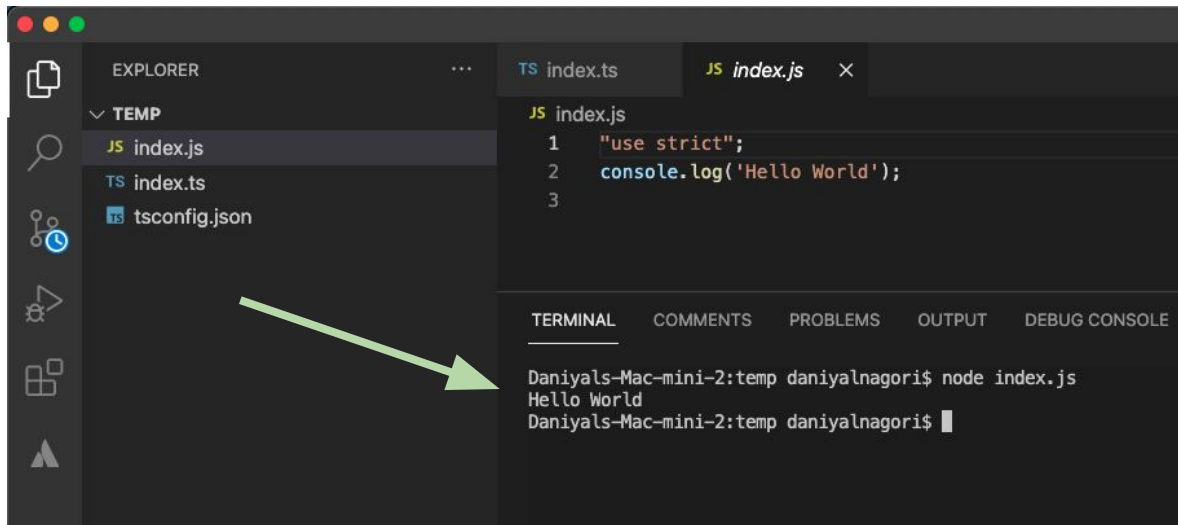
It will compile into Javascript.



Run Typescript Output

```
node index.js
```

You will see the output inside
the terminal.



Writing code

- Formatting code:
 - Code needs to be formatted well. If you have a long file with many lines of code and you didn't stick to a few basic formatting rules, it is going to be hard to understand what you've written.
 - The two most important rules for formatting the code are indentations and semicolons.

```
1 var variable1 = 1; var variable2 = 2; var variable3 = 3
2 if (variable1 === 1) {
3   console.log(variable1)
4   if (variable2 === 1) {
5     console.log(variable2)
6   } else {
7     console.log(variable3)
8   }
9 }
```

```
1 var variable1 = 1;
2 var variable2 = 2;
3 var variable3 = 3;
4
5 if (variable1 === 1) {
6   console.log(variable1);
7   if (variable2 === 1) {
8     console.log(variable2);
9   } else {
10    console.log(variable3);
11  }
12 }
```



Basics

Variables

Variables

- Variable means anything that can vary.
- A TypeScript variable is simply **a name of storage location**.
- A variable must have a unique name.



Variables

- Variables are values in your code that can represent different values each time the code runs.
- The first time you create a variable, you declare it. And you need a special word for that: `let` , `var` , Or `const` .

Example: `let firstname = "Ali";`

- The commonly used naming conventions used for **variables** are camel-case.

Example: `let firstName = "Ali";`



Variables Names

- A variable name can't contain any spaces
- A variable name can contain only letters, numbers, dollar signs, and underscores.
- The first character must be a letter, or an underscore (-), or a dollar sign (\$).
- Subsequent characters may be letters, digits, underscores, or dollar signs.
- Numbers are not allowed as the first character of variable.



Type Annotations on Variables

When you declare a variable using `const`, `var`, or `let`, you can optionally add a type annotation to explicitly specify the type of the variable:

```
let myName: string = "Alice";
```

TypeScript doesn't use "types on the left"-style declarations like `int x = 0`; Type annotations will always go after the thing being typed.

In most cases, though, this isn't needed. Wherever possible, TypeScript tries to automatically infer the types in your code.

```
// No type annotation needed -- 'myName' inferred as type 'string'
```

```
let myName = "Alice";
```

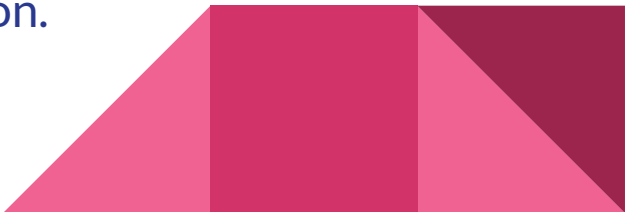


Comments

- Single line TypeScript comments **start with two forward slashes (//)**.
- All text after the two forward slashes until the end of a line makes up a comment
- Even when there are forward slashes in the commented text.
- Multi-line Comments
- Multi-line comments start with `/*` and end with `*/`.
- Any text between `/*` and `*/` will be ignored by JavaScript.



Statements

- A computer program is a list of "instructions" to be "executed" by a computer.
 - In a programming language, these programming instructions are called statements.
 - A JavaScript program is a list of programming statements.
 - TypeScript applications consist of statements with an appropriate syntax. A single statement may span multiple lines. Multiple statements may occur on a single line if each statement is separated by a semicolon.
- 

Let, Var, Const

var and let are both used for variable declaration in TypeScript but the difference between them is that var is function scoped and let is block scoped. Variable declared by let cannot be redeclared and must be declared before use whereas variables declared with var keyword are hoisted.

const is an augmentation of let in that it prevents re-assignment to a variable.

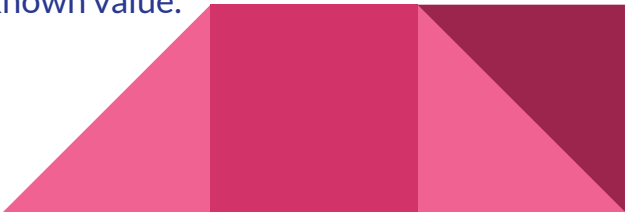
Dont use var, use let and const



Primitive data types

Primitive data types

- String
 - A string is used to store a text value.
Example: `let firstName = "Ali";`
- Number
 - A number is used to store a numeric value.
Example: `let score = 25;`
- Boolean
 - A boolean is used to store a value that is either `true` or `false`.
Example: `let isMarried = false;`
- Undefined
 - An undefined type is either when it has not been defined or it has not been assigned a value.
Example: `let unassigned;`
- Null
 - null is a special value for saying that a variable is empty or has an unknown value.
Example: `let empty = null;`



Template Literals

A new and fast way to deal with strings is **Template Literals** or **Template String**.

How we were dealing with strings before ?

```
var myName = "daniyal" ;
```

```
var hello = "Hello " + myName ;
```

```
console.log(hello); //Hello daniyal
```



Template Literals

What is Template literals ?

As we mentioned before , it's a way to deal with strings and specially dynamic strings ; so you don't need to think more about what's the next quote to use single or double.

How to use Template literals

It uses a `backticks` to write string within it.



Analyzing and modifying data types

Analyzing and modifying data types

- You can check the type of a variable by entering `typeof`.

Example:

```
let testVariable = 1;  
console.log(typeof testVariable);
```

- The variables in TypeScript cannot change types. Example:

```
let a = 2;  
a = "2"; //Error
```



Operators

Operators

- Arithmetic operators:

- Addition

Example:

- ```
let n1 = 1;
let n2 = 2;
console.log(n1 + n2); // ?
```
  - ```
let str1 = "1";  
let str2 = "2";  
console.log(str1 + str2); // ?
```



Operators

- Arithmetic operators:

- Subtraction

Example:

- ```
let n1 = 5;
let n2 = 2;
console.log(n1 - n2); // ?
```

- Multiplication

Example:

- ```
let n1 = 5;  
let n2 = 2;  
console.log(n1 * n2); // ?
```



Operators

- Arithmetic operators:

- Division

Example:

- `let n1 = 4;`
`let n2 = 2;`
`console.log(n1 / n2); // ?`

- Exponentiation

Example:

- `let n1 = 2;`
`let n2 = 2;`
`console.log(n1 ** n2); // ?`



Operators

- Arithmetic operators:

- Modulus

Example:

- `let n1 = 10;`
`let n2 = 3;`
`console.log(n1 % n2); // ?`



Operators

- Assignment operators:
 - Assignment operators are used to assign values to variables.

Example:

- ```
let n = 5;
console.log(n); // 5
n += 5;
console.log(n); // 10
n -= 5;
console.log(n); // 5
```





# Operators

- Comparison operators:

- Comparison operators are used to compare values of variables.

Example:

```
■ let n = 5;
 console.log(n == 5); // true
 console.log(n === 5); // true
 console.log(n != 5); // false
 console.log(n > 8); // false
 console.log(n < 8); // true
 console.log(n >= 8); // false
 console.log(n <= 8); // true
```



# Operators

- Logical operators:

- Logical operators are used to combine multiple conditions in one.

Example:

- ```
let n = 5;
console.log(n >= 5 && n < 10); // true
console.log(n > 5 && n < 10); // false
console.log(n >= 5 || n < 10); // true
console.log(n > 5 || n < 10); // true
console.log(!(n < 10)); // false
console.log(!(n > 10)); // true
```



Thank You

Start Working on Getting Started Exercises with
TypeScript and Node.js

[https://github.com/panaverse/typescript-node-projects/
blob/main/getting-started-exercises.md](https://github.com/panaverse/typescript-node-projects/blob/main/getting-started-exercises.md)

Advanced TypeScript:

<https://github.com/panaverse/learn-typescript>