# TypeScript Basics for Automation Testers – Day 4

## Topic: Datatypes in TypeScript (Part 1 – Dynamic vs Static Typing)

---

### What is Data?

Data is something we assign as a **value** to any variable.
This value could be a **number, string, boolean, or character** and we need to specify what type of data it is.

---

### Before knowing about datatypes in TypeScript, remember:

- **JavaScript (JS)** → Dynamically typed programming language
- **TypeScript (TS)** → Statically typed programming language

**Note:**
Since JS is *dynamic*, it is **not type-safe**.
Since TS is *static*, it **is type-safe**.

---

### What is Dynamic?

"Dynamic" means *something that keeps changing or can vary depending on the situation.*
It is not fixed — its behaviour or value can change while the program is running.

**In Programming Context:**
When we say a language is **dynamic**, it means:

- The data type of a variable is **decided automatically** when you assign a value.
- You can **change the data type later** without any restriction.
- The program determines what kind of data it's dealing with **at runtime** (while it's executing).

---

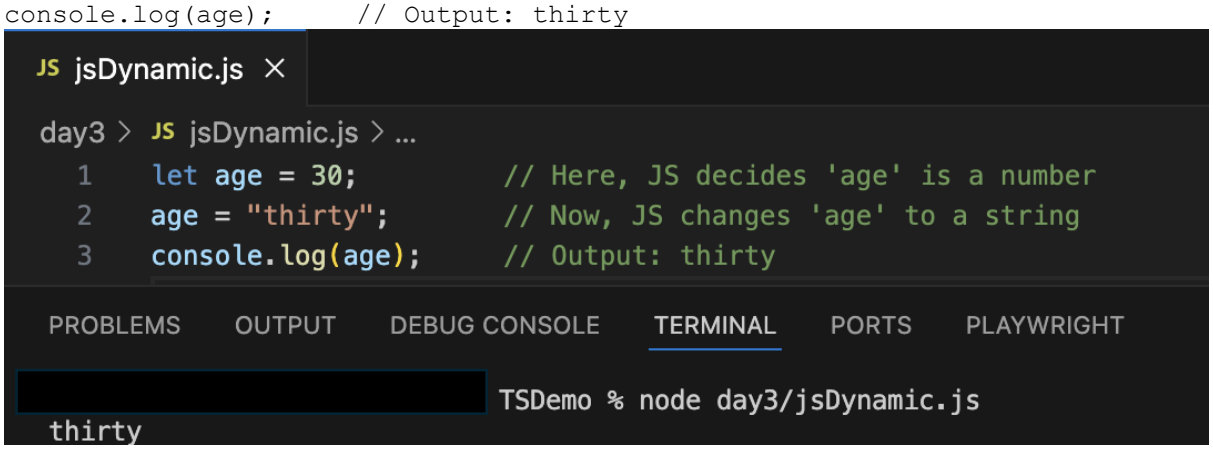
### How Dynamic is Related Here (JavaScript):

JavaScript is a **dynamically typed programming language** because:

- It doesn't require specifying the data type when declaring a variable.
- It assigns the type automatically based on the value given.
- The same variable can hold different data types at different times.

## Example:

```
let age = 30;          // Here, JS decides 'age' is a number
age = "thirty";        // Now, JS changes 'age' to a string
console.log(age);      // Output: thirty
```




Here, you can see —

- The type of `age` changed **dynamically** from number → string.
- JavaScript doesn't stop you from doing this.

Hence, JavaScript is **dynamic and not type-safe**.

---

## What is Static?

"Static" means *something that does not change or remains fixed.*
It's the opposite of dynamic — once something is defined, it stays that way.

**In Programming Context:**
When a language is **statically typed**, it means:

- The data type of a variable is **defined before execution** (at compile time).
- The type **cannot change later** during the program's execution.
- If you try to assign a different type of value, it gives a **compile-time error**.

---

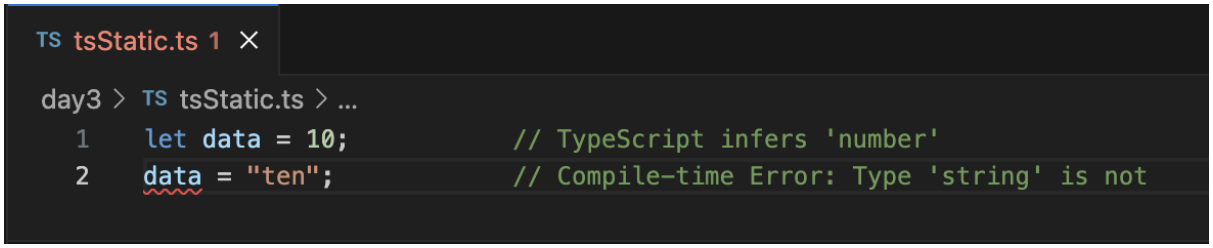
## How Static is Related Here (TypeScript):

TypeScript is a **statically typed programming language** because:

- You must declare the variable type (or it infers it only once).
- Once a variable's type is decided, it **cannot be changed**.
- The compiler checks for type mismatches **before running the code**.

**Example:**

```
let data = 10;            // TypeScript infers 'number'
data = "ten";             // Compile-time Error: Type 'string' is not
assignable to type 'number'
```




So, TypeScript helps you catch these mistakes **before running** the code.
That's why it's called **statically typed** — the type is fixed at compile time.

---

## What is Type Safety?

"Type safety" means *making sure we use the correct kind of data in the correct place.*
It prevents mixing incompatible types — like adding a number to a string by mistake.

**In Programming Context:**
A **type-safe** language ensures that:

- You can't assign values of the wrong type to a variable.
- You can't call functions or access properties with invalid types.
- Errors are caught **early at compile time**, not during execution.

---

## How Type Safety is Related Here (TypeScript):

TypeScript is **type-safe** because it enforces the type rules strictly.

**Example:**

```
let count: number = 5;
count = "five";   // Compile-time Error: Type 'string' is not assignable to
type 'number'
```


This means TypeScript prevents you from using the wrong data type.
So your code becomes **more reliable, easier to debug, and less error-prone.**

---

## How to Prove JavaScript is Dynamically Typed

```
let age = 30;
console.log(typeof(age)); // number
age = "thirty";
console.log(typeof(age)); // string
console.log(age); // thirty
```


**Explanation:**

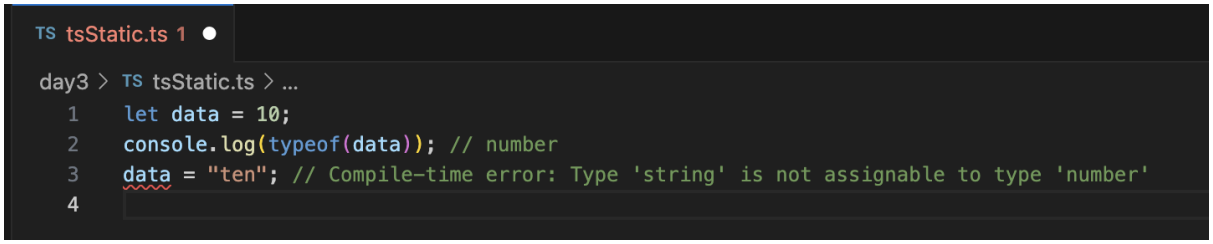
- Initially, `age` was a number.
- Later, it was changed to a string.
- JS allowed it without any error.
- The **type changes dynamically** based on the value.

That's why JS is **not type-safe** and is **dynamically typed**.

---

## How TypeScript is Statically Typed

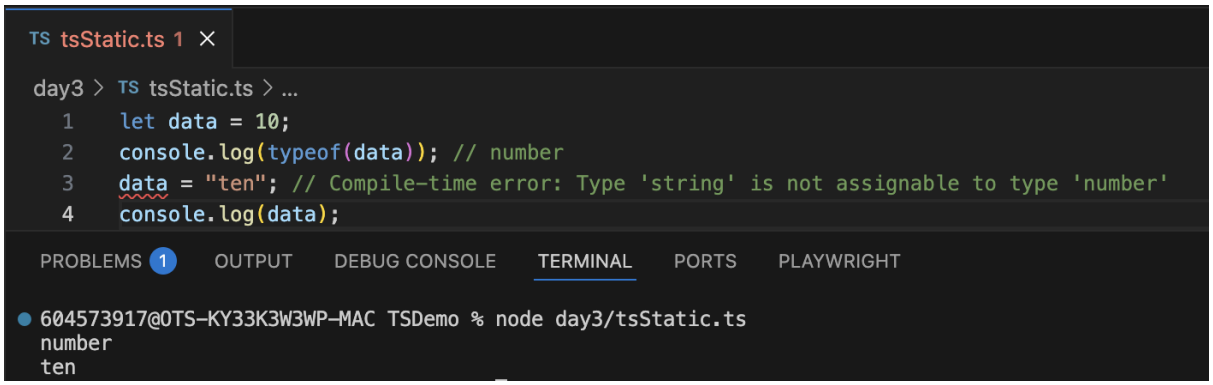
```
let data = 10;
console.log(typeof(data)); // number
data = "ten"; // Compile-time error: Type 'string' is not assignable to
type 'number'
```




**Explanation:**
Once the type is decided (`number`), you cannot assign another type (like a string).

---

### But if you try running this file directly using Node...




You may still see:

```
ten
```

**Why?**
Because Node.js executes **JavaScript**, not TypeScript.
Even though TypeScript showed a compile-time error, Node doesn't understand it.

---

## Correct Way to Run TypeScript

Always use **`ts-node`** or **`tsx`** command to run `.ts` files.

**For TypeScript files:**

```
tsx test.ts
```

**Don't use Node for TS files.**

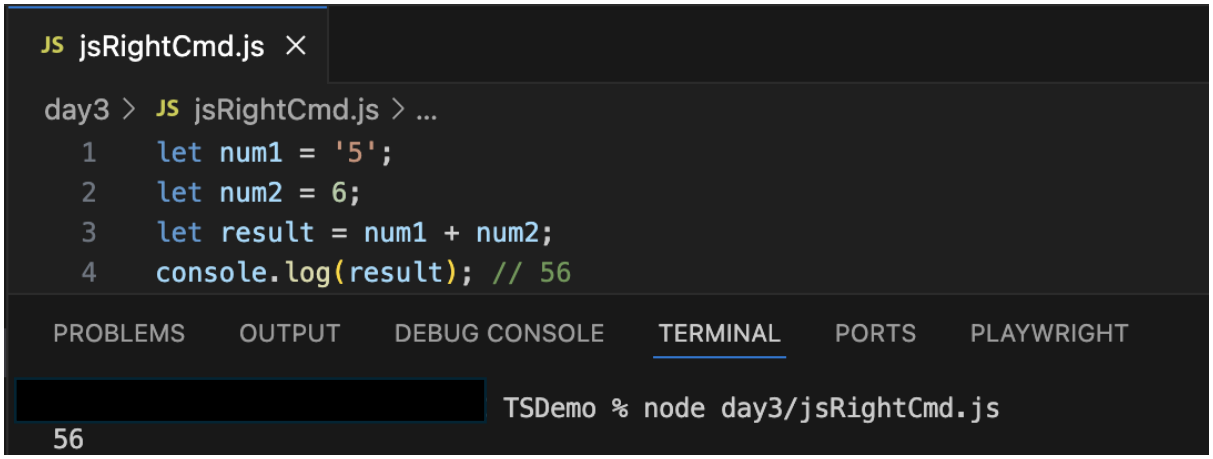
**For JavaScript files:**

```
node test.js
```

---

## Example: Number + String in JS

```
let num1 = '5';
let num2 = 6;
let result = num1 + num2;
console.log(result); // 56
```




**Why 56?**
Because `'5'` is a string — JS **concatenates** it with 6 → `'56'`.

---

## Example: Number + String in TS

```
let num1: string = "5";
let num2: number = 6;
let result = num1 + num2;
console.log(result); // 56
```


Run using:

```
tsx test.ts
```

Even in TS, the result is `'56'`, but **the type is clearly defined**, and any wrong type assignment will cause an error before running.

---

## Note:

Some code may throw errors if you use `node` to execute TypeScript files —
because **Node cannot identify TypeScript types**.

Always use:

- `tsx` → for `.ts` files
- `node` → for `.js` files

---

## Practice:

1. Try creating one JS file and one TS file.
2. Change variable types and check what happens.
3. Use both `node` and `tsx` to understand the difference.

---

## Questions

1. What makes JavaScript a dynamically typed language?
2. Why is TypeScript considered statically typed?
3. What is type safety in programming?
4. How can we prove that JavaScript is dynamically typed?
5. How can we prove that TypeScript is statically typed?
6. Why do we use `tsx` for running TypeScript files instead of `node`?
7. What happens if we run a `.ts` file using Node.js?
8. Why is using TypeScript preferred over JavaScript for automation projects?

---

## Answers

1. **What makes JavaScript a dynamically typed language?**
    - In JavaScript, the **type of a variable is decided automatically at runtime** based on the assigned value.
    - You can change the type later (for example, from number to string) without any error.
    - That's why JS is called **dynamically typed** — because the type is decided "on the go."

    **Example:**

    ```
    let age = 25;       // number
    age = "twenty";     // string (allowed in JS)
    console.log(typeof age); // string
    ```

---

2. **Why is TypeScript considered statically typed?**
    - In TypeScript, the **type of a variable is fixed at compile time**, and you can't assign a different type later.
    - The compiler checks for type mismatches before execution, which ensures safer and more predictable code.

**Example:**

```
let age: number = 25;
age = "twenty"; // Compile-time error
```

---

3. **What is type safety in programming?**
   - **Type safety** means using the correct kind of data in the correct place.
   - It prevents assigning or using the wrong data type for a variable.
   - TypeScript enforces type safety at compile time, avoiding potential runtime errors.

**Example:**

```
let name: string = "Yogi";
name = 123; // Type 'number' is not assignable to type 'string'
```

---

4. **How can we prove that JavaScript is dynamically typed?**
   - In JavaScript, a variable's type changes based on its latest value.
   - You can assign a number first and then a string later without any compile-time error.

**Example:**

```
let data = 30;
console.log(typeof data); // number
data = "thirty";
console.log(typeof data); // string
```

---

5. **How can we prove that TypeScript is statically typed?**
   - In TypeScript, once a variable's type is defined, you cannot change it.
   - The compiler immediately throws an error when you try to assign a different type.

**Example:**

```
let data: number = 10;
data = "ten"; // Compile-time error
```

---

6. **Why do we use `tsx` for running TypeScript files instead of `node`?**
   - `tsx` (or `ts-node`) understands and executes TypeScript syntax directly.
   - It compiles `.ts` code internally before running it.
   - Node.js only understands JavaScript, not TypeScript.

**Command Example:**

```
tsx test.ts
```

---

7. **What happens if we run a `.ts` file using Node.js?**
    - Node.js throws a syntax error because it can't recognize TypeScript syntax (like `:number` or `:string`).
    - Node only runs JavaScript files (`.js`).

    **Example:**

    ```
    node test.ts
    ```

    Output:

    ```
    SyntaxError: Unexpected token ':'
    ```

---

8. **Why is using TypeScript preferred over JavaScript for automation projects?**
    - TypeScript provides **type safety**, **better debugging**, and **compile-time error checking**.
    - It reduces runtime bugs and makes code **more predictable** and **maintainable** — especially in large automation frameworks.
    - This is why modern automation projects (like Playwright) use TypeScript instead of plain JavaScript.

---