Built-in Types in TypeScript

You are developing a financial analytics dashboard for a multinational company.

- You need to store and process numbers (balances, interest rates), strings (account names, currencies), booleans (active/inactive), and more.
- Some functions return nothing, some return computed values, and some data may be missing or intentionally absent.
- You must ensure that every value is validated and handled correctly, with no room for type confusion or runtime errors.

The challenge:
How do you use TypeScript's built-in types to model, validate, and process all the different kinds of data in a financial system, ensuring correctness and safety at every step?

2. Learning Objectives

- Understand and use all built-in types in TypeScript.
- Declare variables and functions with explicit types.
- Handle missing or special values (`null` , `undefined` , `void` , `never`).
- Use type-safe objects, arrays, and symbols.

3. Concept Introduction with Analogy

Analogy: The Bank Vault with Specialized Lockers

In a bank vault, each locker is designed for a specific kind of item:

- Some hold cash (numbers), others hold documents (strings), some are for yes/no agreements (booleans), some are empty by design (void), and some are reserved for unique or rare items (symbols, never).
- The vault manager (TypeScript) ensures you never put the wrong item in the wrong locker, keeping everything safe and organized.

4. Technical Deep Dive

Built-in Types Table

Data type	Keyword	Description
Number	number	Double precision 64-bit floating point values
String	string	Sequence of Unicode characters
Boolean	boolean	Logical values, true and false
Void	void	For functions that do not return a value
Null	null	Intentional absence of an object value
Undefined	undefined	Value given to uninitialized variables
Symbol	symbol	Unique and immutable primitive for unique object keys
Object	object	Instances of classes, arrays, functions, etc.
Never	never	Values that never occur (e.g., a function that always throws)

Number

```
let age: number = 30;
let marks: number = 30.5;
let hex: number = 0xff;
let binary: number = 0b1010;
```

String

```
let first_name: string = 'John';
let last_name: string = "Doe";
let full_name: string = `${first_name} ${last_name}`;
```

Boolean

```
let isReady: boolean = true;
```

Symbol

```
const UNIQUE_KEY = Symbol();
let obj = { [UNIQUE_KEY]: "SecretValue" };
```

Null & Undefined

```
let empty: null = null;
let undef: undefined;
```

Object

```
let person: object = { name: "Bob" };
```

Void

```
function log(): void {
  console.log("log");
}
```

Never

```
function fail(): never {
  throw new Error("This always fails");
}
```

5. Step-by-Step Data Modeling & Code Walkthrough

1. **Declare variables for each built-in type:**

   ```
   let accountBalance: number = 1000.50;
   let accountName: string = "Savings";
   let isActive: boolean = true;
   let transactionId: symbol = Symbol("txn");
   let account: object = { id: 1, name: "Main" };
   let missingValue: null = null;
   let notSet: undefined;
   ```

2. **Function with void return:**

   ```
   function printStatement(): void {
     console.log("Statement printed.");
   }
   ```

3. **Function with never return:**

   ```
   function criticalError(): never {
     throw new Error("Critical failure!");
   }
   ```

6. Interactive Challenge

Your Turn!

- Create a function `processTransaction` that takes an amount (number), a description (string), and a flag `isCredit` (boolean).

- If the amount is negative, the function should throw an error (never).
- If the description is missing, use `undefined` and handle it in the function.
- Print a summary of the transaction.

7. Common Pitfalls & Best Practices

- **Always specify types for variables and function parameters.**
- **Handle `null` and `undefined` explicitly to avoid bugs.**
- **Use `never` only for truly unreachable code.**
- **Prefer objects for structured data; use symbols for unique keys.**

8. Quick Recap & Key Takeaways

- TypeScript's built-in types cover all common values and behaviors.
- Strong typing prevents many runtime errors.
- Each type has a clear purpose and usage.

9. Optional: Programmer's Workflow Checklist

- Use explicit types for all variables and function parameters.
- Handle special values (`null` , `undefined` , `never`) with care.
- Use objects and symbols where appropriate.
- Test for edge cases (missing values, errors).