

# TypeScript Variables



# Basic Types

Type	Description
<b>boolean</b>	<b>true/false values</b>
<b>number</b>	<b>Supports integer and floating point numbers</b>
<b>string</b>	<b>Text data. Enclosed in single or double quotes</b>
<b>any</b>	<b>Supports "any" datatype assignment</b>
<b>Others ...</b>	<b>See details at <a href="http://www.typescriptlang.org">www.typescriptlang.org</a></b>

# Define Variables

Syntax

```
let <variableName>: <type> = <initial value>;
```

Example

```
let found: boolean = true;
```

# Examples

```
let found: boolean = true;
```

```
let grade: number = 88.6;
```

```
let firstName: string = "Anup";
```

```
let lastName: string = 'Kumar';
```

Double-quotes

Single-quotes

# Examples

true or false

```
let found: boolean = true;
```

73

```
let grade: number = 88.6;
```

64.5  
100

```
let firstName: string = "Anup";
```

```
let lastName: string = 'Kumar';
```

-quotes

Single-quotes

# TypeScript: "let" keyword

- We are using the new TypeScript **let** keyword for variable declarations
  - As opposed to using traditional JavaScript **var** keyword
- The JavaScript **var** keyword had a number of gotchas and pitfalls
  - Scoping, capturing, shadowing etc
- The new TypeScript **let** keyword helps to eliminate those issues

# TypeScript is Strongly Typed

```
let found: boolean = true;  
let grade: number = 88.6;  
let firstName: string = "Anup";  
let lastName: string = 'Kumar';
```

```
// this is okay ... we can assign to different values
```

```
found = false;  
grade = 99;  
firstName = 'Eric';  
lastName = 'Noh';
```

This is ok

# TypeScript is Strongly Typed

```
let found: boolean = true;
let grade: number = 88.6;
let firstName: string = "Anup";
let lastName: string = 'Kumar';
```

*// this will generate compilation errors ...*

```
found = 0;
grade = "A";
firstName = false;
lastName = 2099;
```



Type mismatch

# Type: any

```
let myData: any = 50.0;  
  
// we can assign different values of any type  
  
myData = false;  
myData = 'Eric';  
myData = 19;
```

This is ok  
But be careful ...  
you lose type-safety

# Displaying Output

File: sample-types.ts

```
let found: boolean = true;
let grade: number = 88.6;
let firstName: string = "Anup";
let lastName: string = 'Kumar';

console.log(found);
console.log("The grade is " + grade);
console.log("Hi " + firstName + " " + lastName);
```

true  
The grade is 88.6  
Hi Anup Kumar

# Run the App

File: sample-types.ts

```
let found: boolean = true;
let grade: number = 88.6;
let firstName: string = "Anup";
let lastName: string = 'Kumar';

console.log(found);
console.log("The grade is " + grade);

console.log("Hi " + firstName + " " + lastName);
```

Compile code using: tsc

Remember, tsc generates a .js file

C:\> tsc sample-types.ts

C:\> node sample-types.js  
true  
The grade is 88.6  
Hi Anup Kumar

Run code using: node

Run the .js file

# Template Strings

```
let firstName: string = "Anup";  
let lastName: string = 'Kumar';
```

```
console.log("Hi " + firstName + " " + lastName);
```

Concatenation could  
become clunky  
for long strings

```
console.log(`Hi ${firstName} ${lastName}`);
```

Use backticks:  
Reference variables with \${..}

Useful for long strings with a lot of concatenation