luv2code

TypeScript Variables

Basic Types

Basic Types

Type	Description

Basic Types

Type	Description
`boolean`	**true/false values**

Basic Types

Type	Description
`boolean`	**true/false values**
`number`	**Supports integer and floating point numbers**

Basic Types

Type	Description
`boolean`	**true/false values**
`number`	**Supports integer and floating point numbers**
`string`	**Text data. Enclosed in single or double quotes**

Basic Types

Type	Description
`boolean`	true/false values
`number`	Supports integer and floating point numbers
`string`	Text data. Enclosed in single or double quotes
`any`	Supports "any" datatype assignment

Basic Types

Type	Description
`boolean`	true/false values
`number`	Supports integer and floating point numbers
`string`	Text data. Enclosed in single or double quotes
`any`	Supports "any" datatype assignment
Others …	See details at www.typescriptlang.org

Define Variables

Define Variables

Syntax

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

Define Variables

Syntax

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

Example

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

Define Variables

Syntax

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

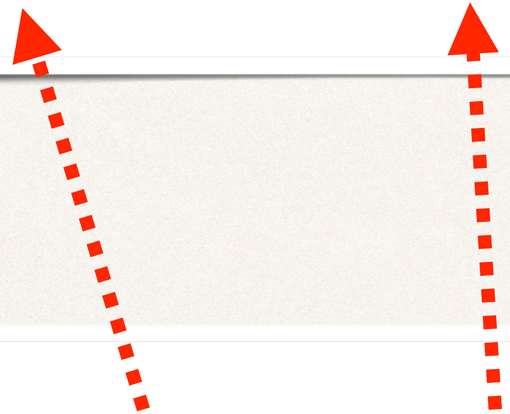
Example

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

Define Variables

Syntax

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



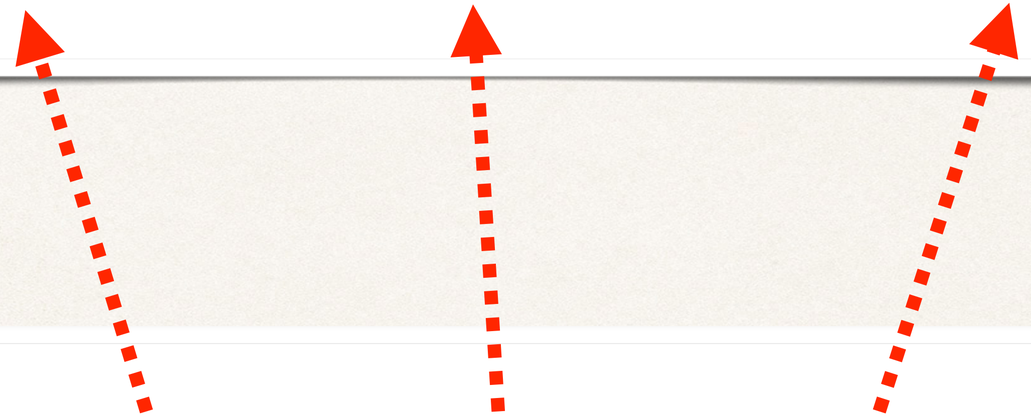
Example

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

Define Variables

Syntax

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



Example

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

Examples

Examples

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

Examples

`true` or `false`

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

Examples

```
let found: boolean = true;

let grade: number = 88.6;
```

Examples

```
let found: boolean = true;

let grade: number = 88.6;
```

```
73
64.5
100
```

Examples

```
let found: boolean = true;

let grade: number = 88.6;

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

Examples

```
let found: boolean = true;

let grade: number = 88.6;

let firstName: string = "Anup";

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

Examples

```
let found: boolean = true;

let grade: number = 88.6;

let firstName: string = "Anup";

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

Double-quotes

Examples

```
let found: boolean = true;

let grade: number = 88.6;

let firstName: string = "Anup";

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

Double-quotes

Single-quotes

TypeScript: "let" keyword

TypeScript: "let" keyword

- We are using the new TypeScript **`let`** keyword for variable declarations

TypeScript: "let" keyword

- We are using the new TypeScript **`let`** keyword for variable declarations
  - As opposed to using traditional JavaScript **`var`** keyword

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

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

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

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
```

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;
```

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;
```

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 is okay ... we can assign to different values

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

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 is okay ... we can assign to different values

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

This is ok

TypeScript is Strongly Typed

TypeScript is Strongly Typed

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

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";
```

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";
```

Type mismatch

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

Type: any

```
let myData: any = 50.0;
```

Type: any

```
let myData: any = 50.0;

// we can assign different values of any type
```

Type: any

```
let myData: any = 50.0;

// we can assign different values of any type

myData = false;
myData = 'Eric';
myData = 19;
```

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

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);
```

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);
```

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

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);
```

Run the App

Compile code using: tsc

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);
```

```
C:\> tsc sample-types.ts
```

Run the App

Compile code using: tsc

Remember, tsc generates a .js file

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);
```

```
C:\> tsc sample-types.ts
```

Run the App

Compile code using: tsc

Remember, tsc generates a .js file

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);
```

```
C:\> tsc sample-types.ts

C:\> node sample-types.js
```

Run code using: node

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
```

Run code using: node

Run the .js file

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);
```

```
C:\> tsc sample-types.ts

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

Template Strings

Template Strings

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

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

Template Strings

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

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

Concatenation could become clunky for long strings

Template Strings

Concatenation could become clunky for long strings

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

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

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

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 ${..}

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