

Taking Advantage of Built-in Types



Brice Wilson

@brice_wilson www.BriceWilson.net



Overview



Built-in TypeScript types

Declarations with *let* and *const*

Type annotations and type inference

Managing null and undefined

Control flow-based type analysis



Basic TypeScript Types

Boolean

Number

String

Array

Enum



Declarations with *let* and *const*

```
console.log(someString);  
var someString = 'Hello World';
```



Declarations with *let* and *const*

```
console.log(someString);  
var someString = 'Hello World';
```



Declarations with *let* and *const*



```
console.log(someString);  
var someString = 'Hello World';
```



Declarations with *let* and *const*



```
console.log(someString);  
var someString = 'Hello World';
```



```
console.log(someString);  
let someString = 'Hello World';
```



Declarations with *let* and *const*



```
console.log(someString);  
var someString = 'Hello World';
```



```
console.log(someString);  
let someString = 'Hello World';
```



Declarations with *let* and *const*



```
console.log(someString);  
var someString = 'Hello World';
```



```
let someString = 'Hello World';  
console.log(someString);
```



Declarations with *let* and *const*



```
console.log(someString);  
var someString = 'Hello World';
```



```
let someString = 'Hello World';  
console.log(someString);
```



Declarations with *let* and *const*



```
console.log(someString);  
var someString = 'Hello World';
```



```
let someString = 'Hello World';  
console.log(someString);
```



```
const someString = 'Hello World';  
console.log(someString);
```



Declarations with *let* and *const*



```
console.log(someString);  
var someString = 'Hello World';
```



```
let someString = 'Hello World';  
console.log(someString);
```



```
const someString = 'Hello World';  
console.log(someString);
```



Type Annotations and Type Inference

```
let x: string = 'I will forever be a string.';
```



Type Annotations and Type Inference

```
let x: string = 'I will forever be a string.';
```



Type Annotations and Type Inference

```
let x: string = 'I will forever be a string.';
```

```
x = 42;
```



Type Annotations and Type Inference

```
let x: string = 'I will forever be a string.';
```

```
x = 42;
```



```
let y = 'I will also forever be a string.';
```



Type Annotations and Type Inference

```
let x: string = 'I will forever be a string.';
```

```
x = 42; ❌
```

```
let y = 'I will also forever be a string.';
```

```
y = 42; ❌
```



Type Annotations and Type Inference

```
let x: string = 'I will forever be a string.';
```

```
x = 42;
```



```
let y = 'I will also forever be a string.';
```

```
y = 42;
```



```
let z = GetSomeValue();
```



Type Annotations and Type Inference

```
let x: string = 'I will forever be a string.';
```

```
x = 42;
```



```
let y = 'I will also forever be a string.';
```

```
y = 42;
```



```
let z = GetSomeValue();
```



```
let z: number = GetSomeValue();
```



Demo



Using *let* and *const* with type annotations



Additional Built-in Types

Void

Null

Undefined

Never

Any



Union Types


```
let someValue: number | string;
```



Union Types

```
let someValue: number | string;
```


```
someValue = 42;
```



Union Types

```
let someValue: number | string;
```

```
someValue = 42;
```




```
someValue = 'Hello World';
```



Union Types

```
let someValue: number | string;
```

```
someValue = 42;
```



```
someValue = 'Hello World';
```



```
someValue = true;
```



Using the `--strictNullChecks` Compiler Option





Using the `--strictNullChecks` Compiler Option

```
let basicString: string;  
basicString = null; ❌
```



Using the `--strictNullChecks` Compiler Option

```
let basicString: string;  
basicString = null;   
basicString = undefined; 
```





Using the `--strictNullChecks` Compiler Option


```
let basicString: string;  
basicString = null; ✖  
basicString = undefined; ✖
```

```
let nullableString: string | null;
```





Using the `--strictNullChecks` Compiler Option



```
let basicString: string;  
basicString = null;   
basicString = undefined; 
```

```
let nullableString: string | null;  
nullableString = null; 
```



Using the `--strictNullChecks` Compiler Option

```
let basicString: string;  
basicString = null;   
basicString = undefined; 
```

```
let nullableString: string | null;  
nullableString = null;   
nullableString = undefined; 
```



Using the `--strictNullChecks` Compiler Option

```
let basicString: string;  
basicString = null; ✗  
basicString = undefined; ✗
```

```
let nullableString: string | null;  
nullableString = null; ✓  
nullableString = undefined; ✗
```

```
let mysteryString: string | null | undefined;  
mysteryString = null; ✓
```



Using the `--strictNullChecks` Compiler Option



```
let basicString: string;  
basicString = null; ❌  
basicString = undefined; ❌
```



```
let nullableString: string | null;  
nullableString = null; ✅  
nullableString = undefined; ❌
```



```
let mysteryString: string | null | undefined;  
mysteryString = null; ✅  
mysteryString = undefined;
```



Using the `--strictNullChecks` Compiler Option

```
let basicString: string;  
basicString = null;   
basicString = undefined; 
```

```
let nullableString: string | null;  
nullableString = null;   
nullableString = undefined; 
```

```
let mysteryString: string | null | undefined;  
mysteryString = null;   
mysteryString = undefined; 
```



Type Assertions

```
let value: any = 5;
```



Type Assertions

```
let value: any = 5;
```

```
let fixedString: string = (<number>value).toFixed(4);
```



Type Assertions

```
let value: any = 5;
```

```
let fixedString: string = (<number>value).toFixed(4);
```

```
console.log(fixedString); // 5.0000
```



Type Assertions

```
let value: any = 5;
```

```
let fixedString: string = (<number>value).toFixed(4);  
console.log(fixedString); // 5.0000
```

```
let fixedString: string = (value as number).toFixed(4);
```



Type Assertions

```
let value: any = 5;
```

```
let fixedString: string = (<number>value).toFixed(4);  
console.log(fixedString); // 5.0000
```

```
let fixedString: string = (value as number).toFixed(4);  
console.log(fixedString); // 5.0000
```



Demo



Writing better code with the
--strictNullChecks option



Demo



Understanding control flow-based type analysis



Summary



Reduce confusion and increase clarity

Reduce unintended consequences and increase stability

Maintain flexibility



Up Next:

Writing Better Functions with TypeScript

