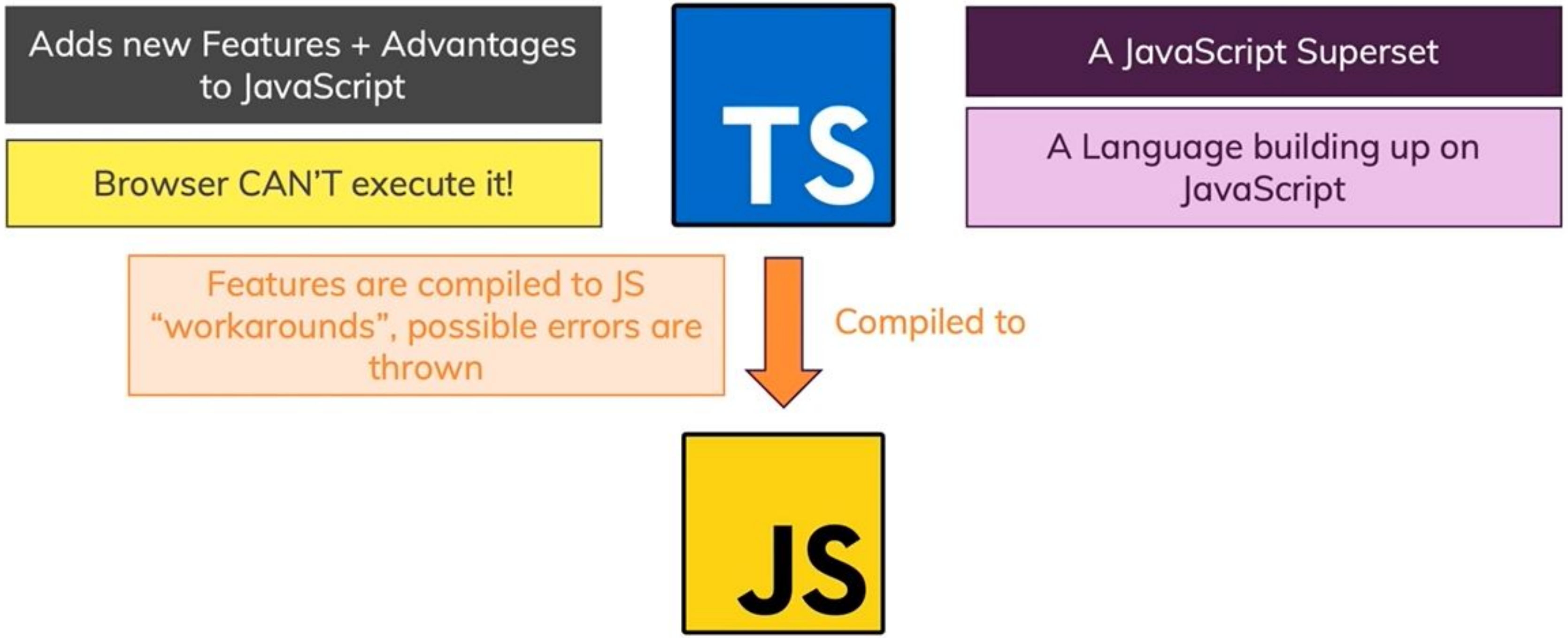
Learning TypeScript

Type-safe JavaScript

What is Typescript?



Adds new Features + Advantages to JavaScript

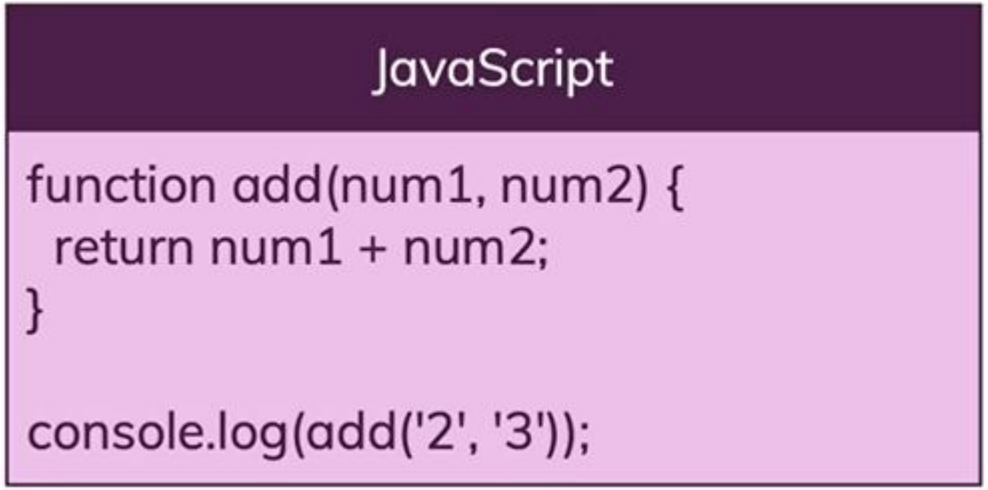
Browser CAN'T execute it!

A JavaScript Superset

A Language building up on JavaScript

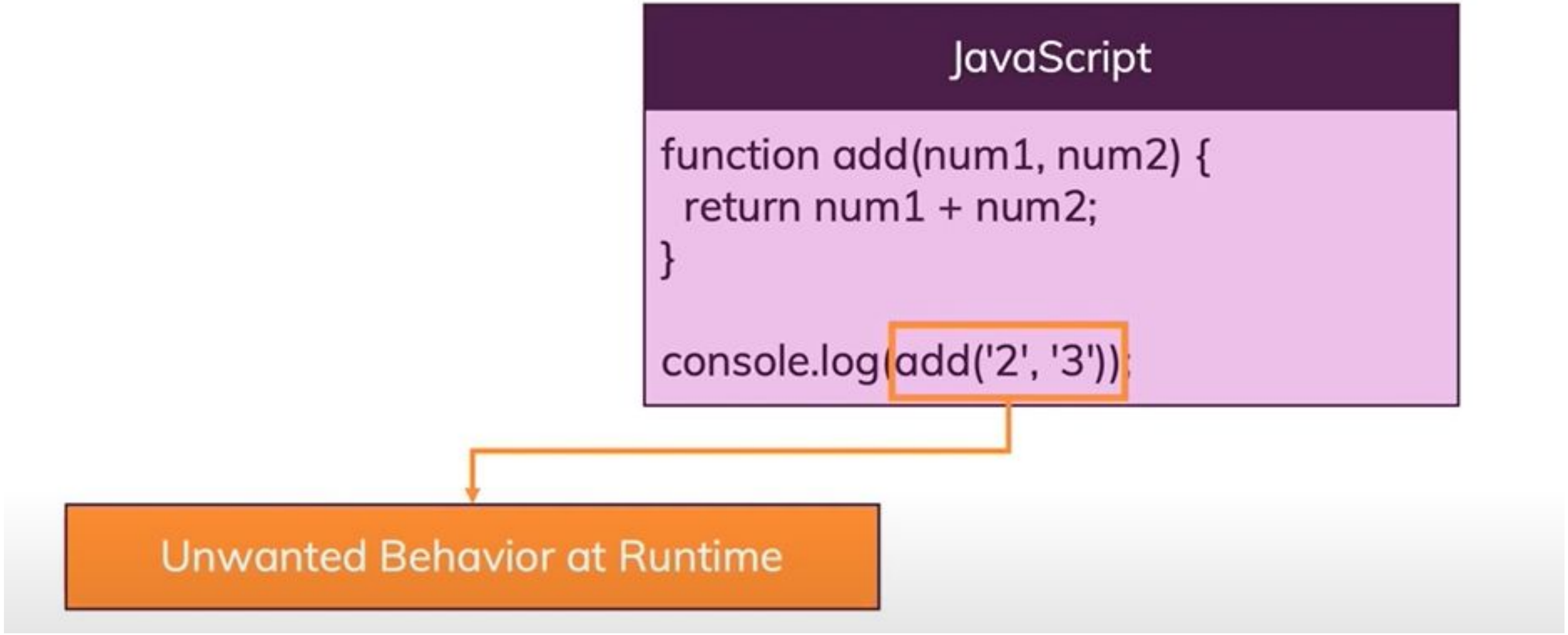
Features are compiled to JS "workarounds", possible errors are thrown

Why Typescript?


JavaScript

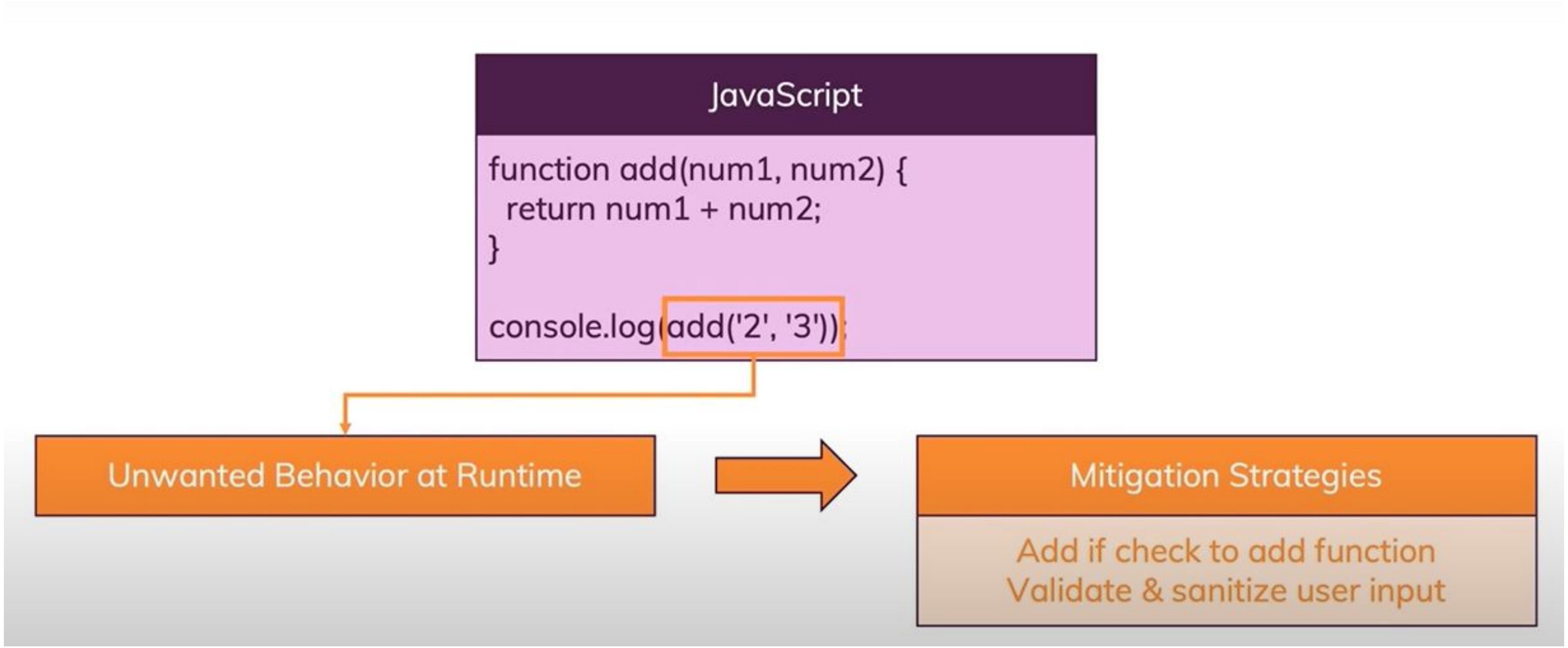
```
function add(num1, num2) {
  return num1 + num2;
}

console.log(add('2', '3'));
```

Why Typescript?



```
function add(num1, num2) {
  return num1 + num2;
}

console.log(add('2', '3'));
```

Why Typescript?



```
function add(num1, num2) {
  return num1 + num2;
}

console.log(add('2', '3'));
```

Unwanted Behavior at Runtime
Mitigation Strategies
Add if check to add function
Validate & sanitize user input

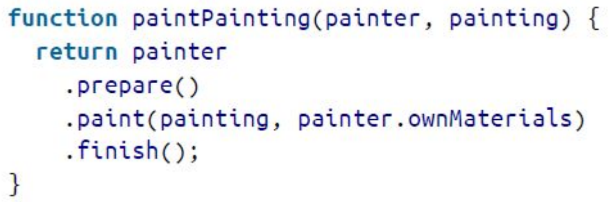
From JavaScript to TypeScript

Chapter 1

Vanilla JavaScript Pitfalls

- **Costly Freedom**

As the number of files grows in the project of JavaScript, you can only have vague ideas on how to call the functions.



```
function paintPainting(painter, painting) {
  return painter
    .prepare()
    .paint(painting, painter.ownMaterials)
    .finish();
}
```

You might even make a lucky guess that painting is a string.

Vanilla JavaScript Pitfalls

- **Loose Documentation**
  - There exists nothing in the JavaScript language specification to formalize description about code purpose.
  - Developers use JSDoc but it has key issues that often make it unpleasant to use in a large codebase
  - Maintaining JSDoc comments across a dozen files doesn’t take up too much time, but across hundreds or even thousands of constantly updating files can be a real chore.

Vanilla JavaScript Pitfalls

- **Weaker Developer Tooling**
  - Because JavaScript doesn’t provide built-in ways to identify types.
  - It can be difficult to automate large changes to or gain insights about a codebase.

TypeScript

- TypeScript was created internally at Microsoft in the early 2010s then released and open sourced in 2012.

- TypeScript is often described as a “superset of JavaScript” or “JavaScript with types.”

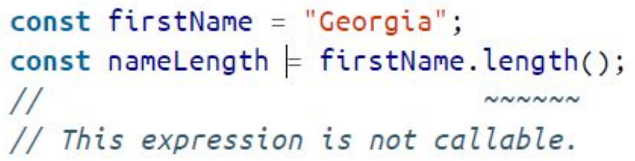
TypeScript

What is TypeScript

- Programming language - that includes all the existing JavaScript syntax, plus new TypeScript-specific syntax for defining and using types
- Type checker - It lets you know if it thinks anything is set up incorrectly
- Compiler - A program that runs the type checker, reports any issues, then outputs the equivalent JavaScript code
- Language service - A program that uses the type checker to tell editors such as VS Code how to provide helpful utilities to developers

Getting Started in the TypeScript Playground

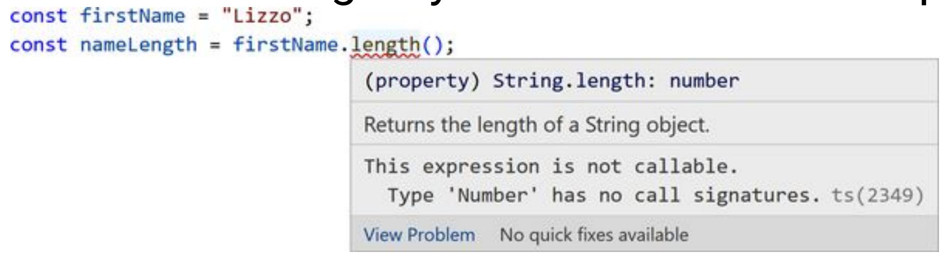
The code is written in normal JavaScript syntax. If you tried to run that code in JavaScript, it would crash!



```
const firstName = "Georgia";
const nameLength = firstName.length();
//                           ~~~~~~
// This expression is not callable.
```

If you were to run the TypeScript type checker on this code, it would use its knowledge that the length property of a string is a number—not a function

Hovering over the code would give you the text of the complaint



```
const firstName = "Lizzo";
const nameLength = firstName.length();
```

```
(property) String.length: number
```

Returns the length of a String object.

```
This expression is not callable.
  Type 'Number' has no call signatures. ts(2349)
```

View Problem   No quick fixes available

Getting Started in the TypeScript Playground

Freedom Through Restriction

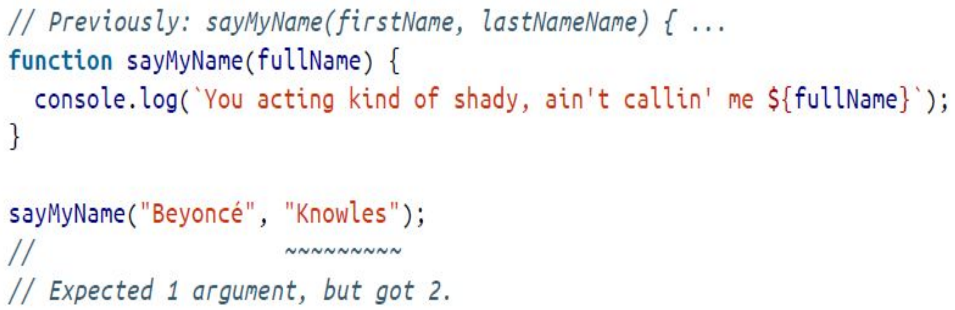
- TypeScript allows us to specify what types of values may be provided for parameters and variables.

- If you change the number of required parameters for a function, TypeScript will let you know if you forget to update a place that calls the function.

Getting Started in the TypeScript Playground

Freedom Through Restriction

- **sayMyName** was changed from taking in two parameters to taking one parameter, but the call to it with two strings wasn't updated and so is triggering a TypeScript complaint:
- That code would run without crashing in JavaScript, but its output would be different from expected (it wouldn't include "Knowles"):

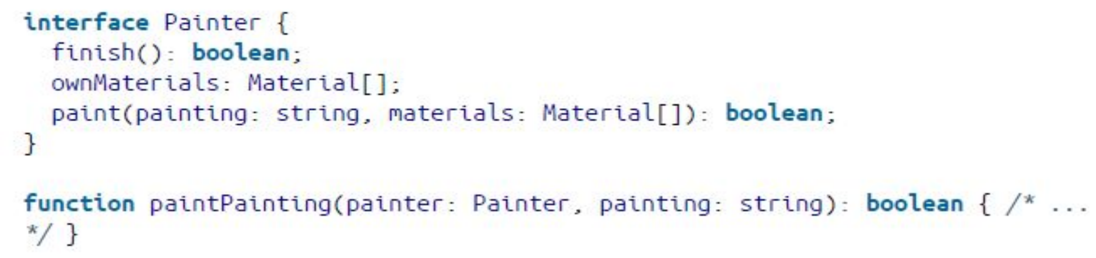


```
// Previously: sayMyName(firstName, lastNameName) { ...
function sayMyName(fullName) {
  console.log(`You acting kind of shady, ain't callin' me ${fullName}`);
}

sayMyName("Beyoncé", "Knowles");
//                   ~~~~~~~~~
// Expected 1 argument, but got 2.
```

Getting Started in the TypeScript Playground

Precise Documentation

a TypeScript version of the paintPainting function from earlier.



```
interface Painter {
  finish(): boolean;
  ownMaterials: Material[];
  paint(painting: string, materials: Material[]): boolean;
}

function paintPainting(painter: Painter, painting: string): boolean { /* ...
*/ }
```

A TypeScript developer reading this code for the first time could understand that painter has at least three properties.

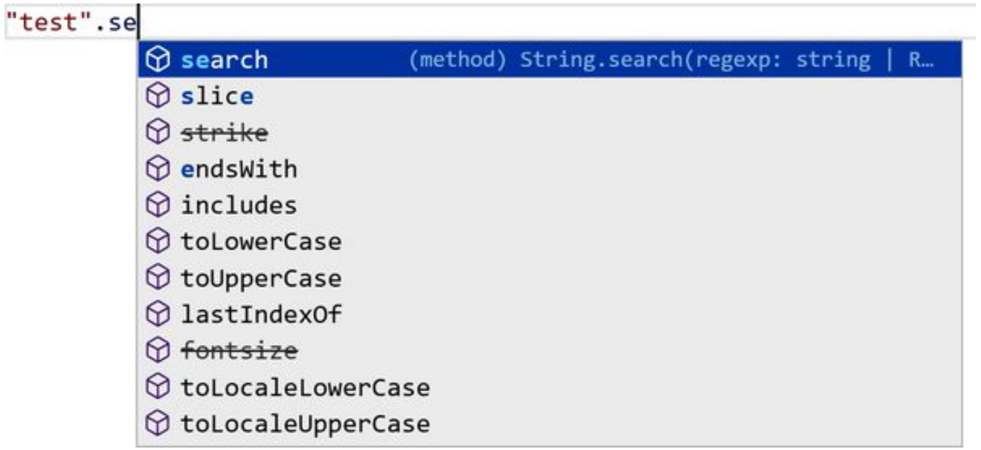
TypeScript provides an excellent, enforced system for describing how objects look.

Getting Started in the TypeScript Playground

Stronger Developer Tooling

TypeScript allow editors such as VS Code to gain much deeper insights into your code.

TypeScript can suggest all the members of the strings

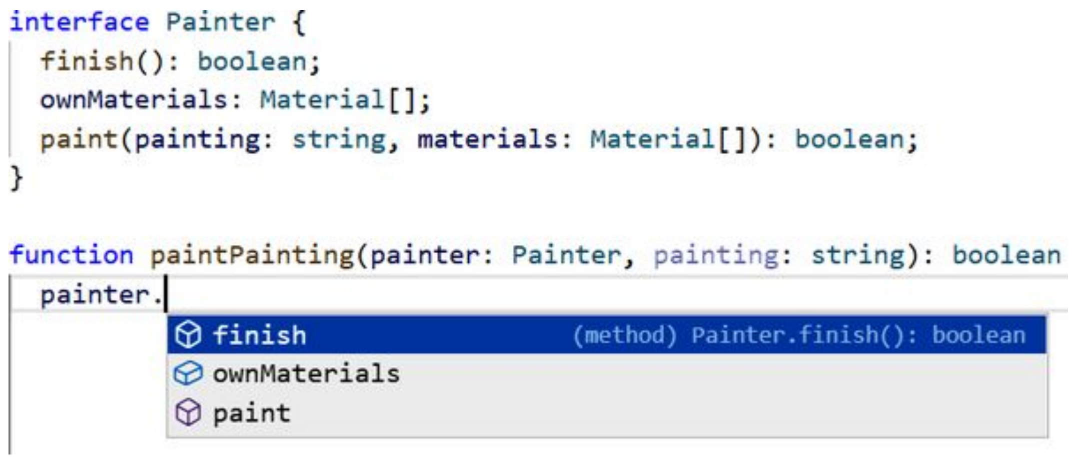


```
"test".se
search            (method) String.search(regexp: string | R…
slice
~~strike~~
endsWith
includes
toLowerCase
toUpperCase
lastIndexOf
~~fontsize~~
toLocaleLowerCase
toLocaleUpperCase
```

Getting Started in the TypeScript Playground

Stronger Developer Tooling

When you add TypeScript’s type checker for understanding code, it can give you these useful suggestions even for code you’ve written.



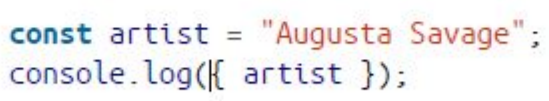
```
interface Painter {
  finish(): boolean;
  ownMaterials: Material[];
  paint(painting: string, materials: Material[]): boolean;
}

function paintPainting(painter: Painter, painting: string): boolean
  painter.
          finish                    (method) Painter.finish(): boolean
          ownMaterials
          paint
```

Getting Started in the TypeScript Playground

Compiling Syntax

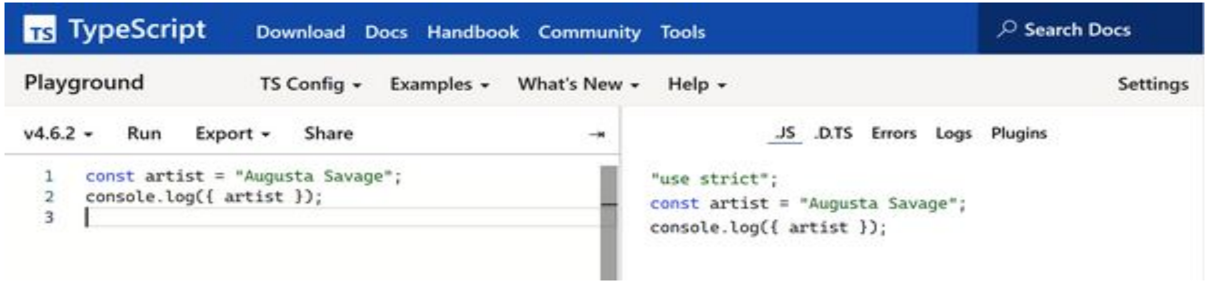
TypeScript's compiler allows us to input TypeScript syntax, have it type checked, and get the equivalent JavaScript emitted.

TypeScript Code



```
const artist = "Augusta Savage";
console.log({ artist });
```

TypeScript compiling TypeScript code into equivalent JavaScript



```
const artist = "Augusta Savage";
console.log({ artist });
```

```
"use strict";
const artist = "Augusta Savage";
console.log({ artist });
```

Getting Started Locally

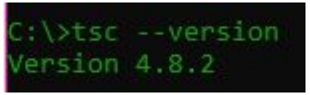
install the latest version of TypeScript globally

```
npm i -g typescript
```

run TypeScript on the command line with the tsc (TypeScript Compiler) command. Try it with the --version flag to make sure it's set up properly:

```
tsc --version
```

Output



```
C:\>tsc --version
Version 4.8.2
```

Getting Started Locally

Running Locally

- Create a folder somewhere on your computer and run this command to create a new tsconfig.json configuration file:

  ```
  tsc --init
  ```

- A tsconfig.json file declares the settings that TypeScript uses when analyzing your code.
- Create a file named index.ts with the following contents:

  ```
  console.log("Hello World");
  ```

- run tsc and provide it the name of that index.ts file:

  ```
  tsc index.ts
  ```

What TypeScript Is Not

Let’s discuss the limitations of TypeScript !

A Remedy for Bad Code

- TypeScript helps you structure your JavaScript, but other than enforcing type safety, it doesn’t enforce any opinions on what that structure should look like.

What TypeScript Is Not

Extensions to JavaScript (Mostly)

- TypeScript does not try to change how JavaScript works at all.

- TypeScript's design goals explicitly state that it should:

  - Align with current and future ECMAScript proposals
  - Preserve runtime behavior of all JavaScript code

What TypeScript Is Not

Slower Than JavaScript

- TypeScript is slow than JavaScript, That claim is generally inaccurate and misleading.

- The only changes TypeScript makes to code are if you ask it to compile your code down to earlier versions of JavaScript to support older runtime environments such as Internet Explorer 11.

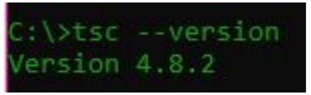
- Browsers and Node.js, will run it.

What TypeScript Is Not

Finished Evolving

- The TypeScript language is constantly receiving bug fixes and feature additions to match the ever-shifting needs of the web community.

- The current version of the TypeScript is



```
C:\>tsc --version
Version 4.8.2
```

The Type System

Chapter 2

What’s in a Type?

- A “type” is a description of what a JavaScript value shape might be.
- “shape” means which properties and methods exist on a value.

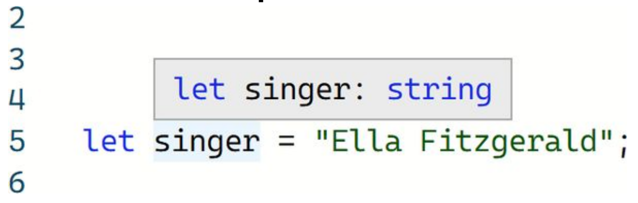
- TypeScript understands the type of the value to be

  one of the seven basic primitives:

1. null; // null
2. undefined; // undefined
3. true; // boolean
4. "Louise"; // string
5. 1337; // number
6. 1337n; // bigint
7. Symbol("Franklin"); // symbol

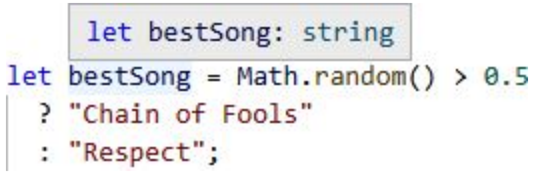
What’s in a Type?

- If you hover your mouse over the variable’s name. The resultant popover will include the name of the primitive,



```
let singer: string
let singer = "Ella Fitzgerald";
```

- TypeScript knows that the ternary expression always results in a string, so the bestSong variable is a string:

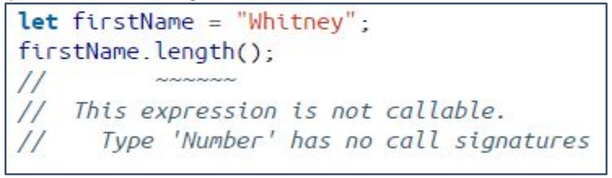


```
let bestSong: string
let bestSong = Math.random() > 0.5
  ? "Chain of Fools"
  : "Respect";
```

What’s in a Type?

Type Systems

A type system is the set of rules for how a programming language understands what types the constructs in a program may have.



```
let firstName = "Whitney";
firstName.length();
//        ~~~~~~
//  This expression is not callable.
//    Type 'Number' has no call signatures
```

TypeScript came to that complaint by, in order:

1. Reading in the code and understanding there to be a variable named firstName

2. Concluding that firstName is of type string because its initial value is a string, "Whitney"

3. Seeing that the code is trying to access a .length member of firstName and call it like a function

4. Complaining that the .length member of a string is a number, not a function (it can’t be called like a function)

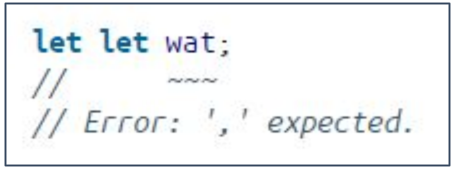
What’s in a Type?

Kinds of Errors

While writing TypeScript, the two kinds of “errors” you’ll come across most frequently are:

Syntax

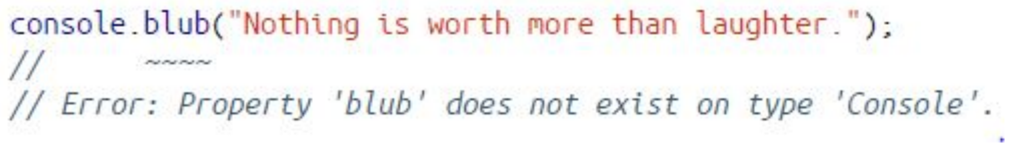
Blocking TypeScript from being converted to JavaScript



```
let let wat;
//      ~~~
// Error: ',' expected.
```

Type

Type errors occur when your syntax is valid but the TypeScript type checker has detected an error with the program’s types.

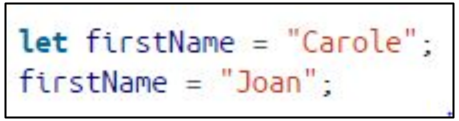


```
console.blub("Nothing is worth more than laughter.");
//      ~~~~
// Error: Property 'blub' does not exist on type 'Console'.
```

Assignability

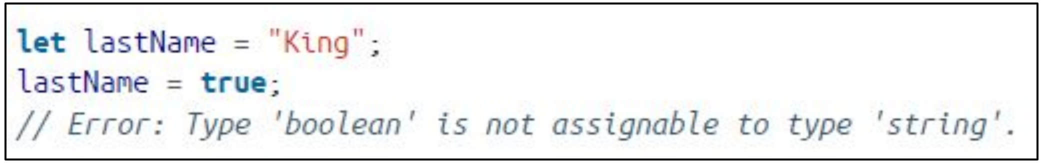
TypeScript is fine with later assigning a different value of the same type to a Variable.

If a variable is, say, initially a string value, later assigning it another string would be fine:



```
let firstName = "Carole";
firstName = "Joan";
```

If TypeScript sees an assignment of a different type, it will give us a type error.



```
let lastName = "King";
lastName = true;
// Error: Type 'boolean' is not assignable to type 'string'.
```

Assignability

Understanding Assignability Errors

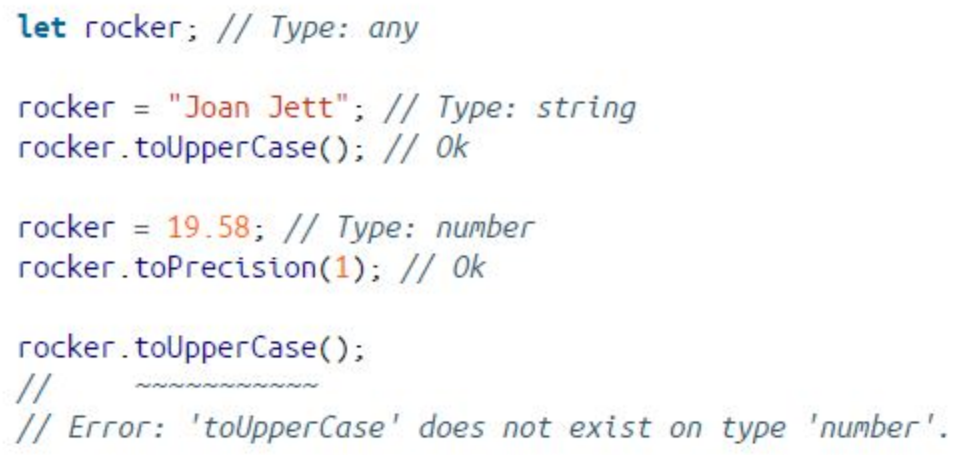
when we wrote

lastName = true in the previous snippet,

we were trying to assign the value of true—type boolean—to the recipient variable lastName—type string.

Type Annotations

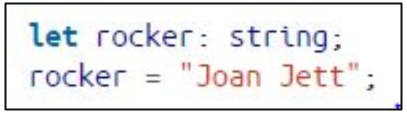
- Sometimes a variable doesn't have an initial value for TypeScript to read.
- It'll consider the variable by default to be implicitly the any type: indicating that it could be anything in the world.
-



```
let rocker; // Type: any

rocker = "Joan Jett"; // Type: string
rocker.toUpperCase(); // Ok

rocker = 19.58; // Type: number
rocker.toPrecision(1); // Ok

rocker.toUpperCase();
//     ~~~~~~~~~~~
// Error: 'toUpperCase' does not exist on type 'number'.
```

Type Annotations

- TypeScript provides a syntax for declaring the type of a variable without having to assign it an initial value, called a *type annotation*.
- A type annotation is placed after the name of a variable and includes a colon followed by the name of a type.



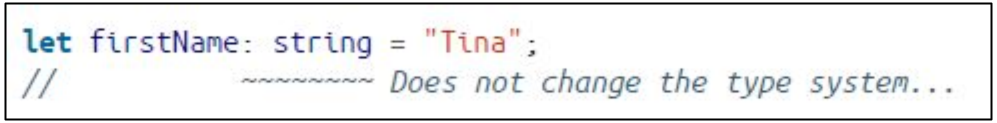
```
let rocker: string;
rocker = "Joan Jett";
```

- These type annotations exist only for TypeScript—they don't affect the runtime code and are not valid JavaScript syntax.

Type Annotations

Unnecessary Type Annotations

The following : string type annotation is redundant because TypeScript could already infer that firstName be of type string:



```
let firstName: string = "Tina";
//           ~~~~~~~~~ Does not change the type system...
```

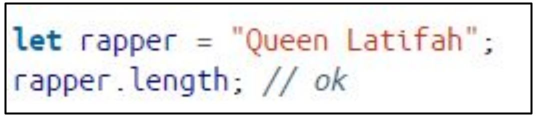
Many developers generally prefer not to add type annotations on variables where the type annotations wouldn’t change anything.

Type Annotations

Type Shapes

- TypeScript also knows what member properties should exist on objects.
-
- If you attempt to access a property of a variable, TypeScript will make sure that property is known to exist on that variable’s type.
-

Suppose we declare a rapper variable of type string. Later on, when we use that rapper variable, operations that TypeScript knows work on strings are allowed:



```
let rapper = "Queen Latifah";
rapper.length; // ok
```

Type Annotations

Modules

The JavaScript programming language did not include a specification for how files can share code between each other until relatively recently in its history.

Module

A file with a top-level export or import

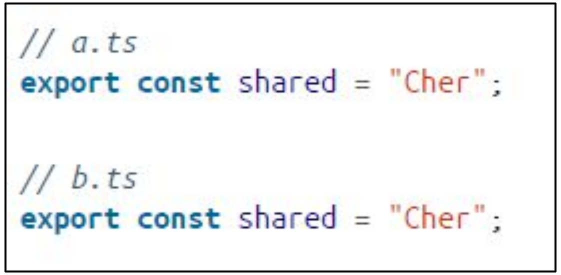
Script

Any file that is not a module

Type Annotations

Modules

- Anything declared in a module file will be available only in that file unless an explicit export statement in that file exports it.
- A variable declared in one module with the same name as a variable declared in another file won’t be considered a naming conflict (unless one file imports the other file’s variable).

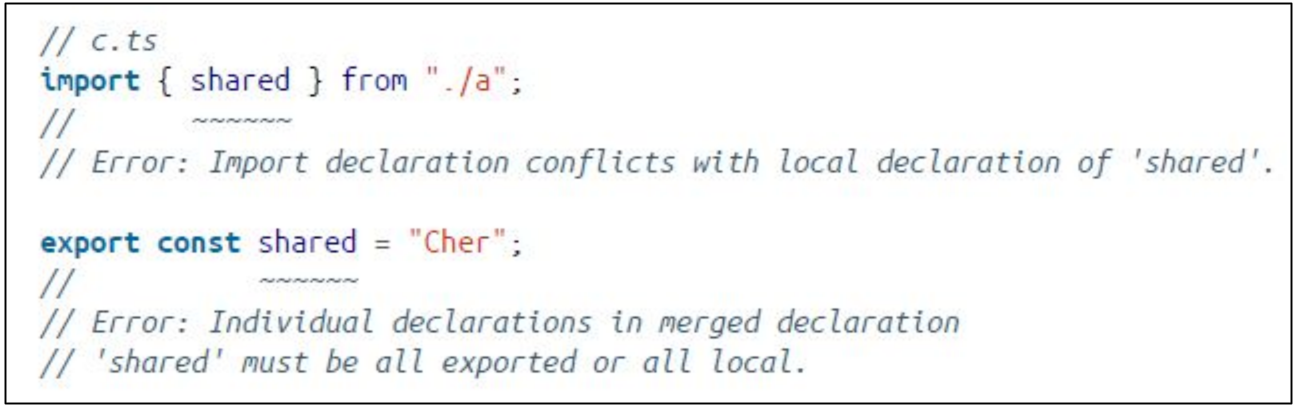


```
// a.ts
export const shared = "Cher";

// b.ts
export const shared = "Cher";
```

Type Annotations

Modules

- c.ts file causes a type error because it has a naming conflict between an imported shared and its own value:

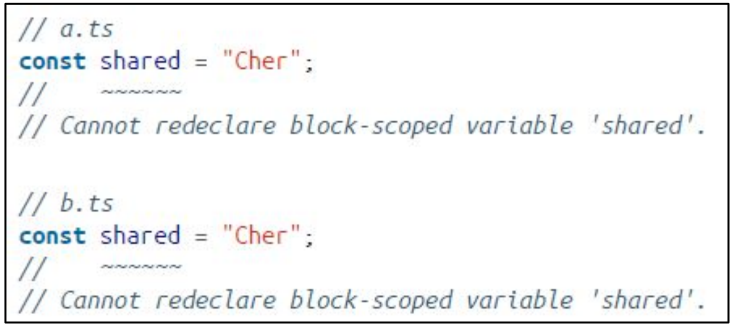


```
// c.ts
import { shared } from "./a";
//       ~~~~~~
// Error: Import declaration conflicts with local declaration of 'shared'.

export const shared = "Cher";
//           ~~~~~~
// Error: Individual declarations in merged declaration
// 'shared' must be all exported or all local.
```

Type Annotations

Modules

- If a file is a script, all scripts have access to its contents.
- That means variables declared in a script file cannot have the same name as variables declared in other script files.



```
// a.ts
const shared = "Cher";
//    ~~~~~~
// Cannot redeclare block-scoped variable 'shared'.

// b.ts
const shared = "Cher";
//    ~~~~~~
// Cannot redeclare block-scoped variable 'shared'.
```

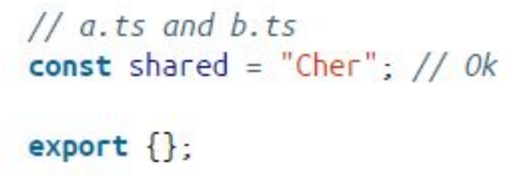
The a.ts and b.ts files are considered scripts because they do not have module-style export or import statements.

That means their variables of the same name conflict with each other as if they were declared in the same file:

Type Annotations

Modules

if you need a file to be a module without an export or import statement, you can add an export { }; somewhere in the file to force it to be a module:



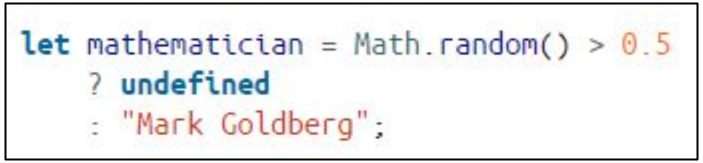
```
// a.ts and b.ts
const shared = "Cher"; // Ok

export {};
```

Unions and Literals

Chapter 3

Union Types

- Take this mathematician variable:

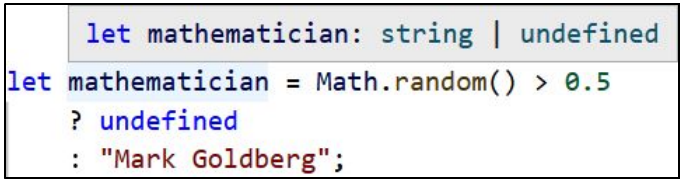


```
let mathematician = Math.random() > 0.5
    ? undefined
    : "Mark Goldberg";
```

What type is mathematician?

mathematician can be either undefined or string. This kind of "either or" type is called a **union**.

- handle code cases where we don't know exactly which type a value is, but do know it's one of two or more options.
- TypeScript represents union types using the | (pipe) operator between the possible values, or constituents.



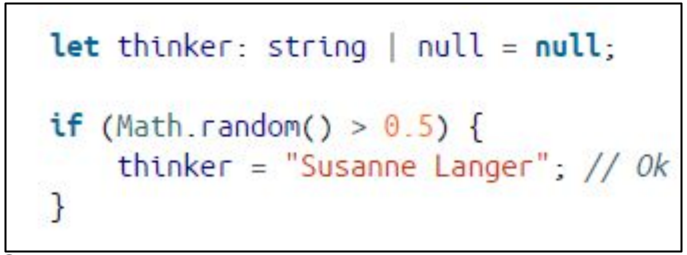
```
    let mathematician: string | undefined
let mathematician = Math.random() > 0.5
    ? undefined
    : "Mark Goldberg";
```

Union Types

Declaring Union Types

- Union types are an example of a situation when it might be useful to give an

explicit type annotation for a variable even though it has an initial value.



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

if (Math.random() > 0.5) {
    thinker = "Susanne Langer"; // Ok
}
```

- thinker starts off null but is known to potentially contain a string instead.

Giving it an explicit string | null type annotation means TypeScript will allow it to be assigned values of type string:

Union Types

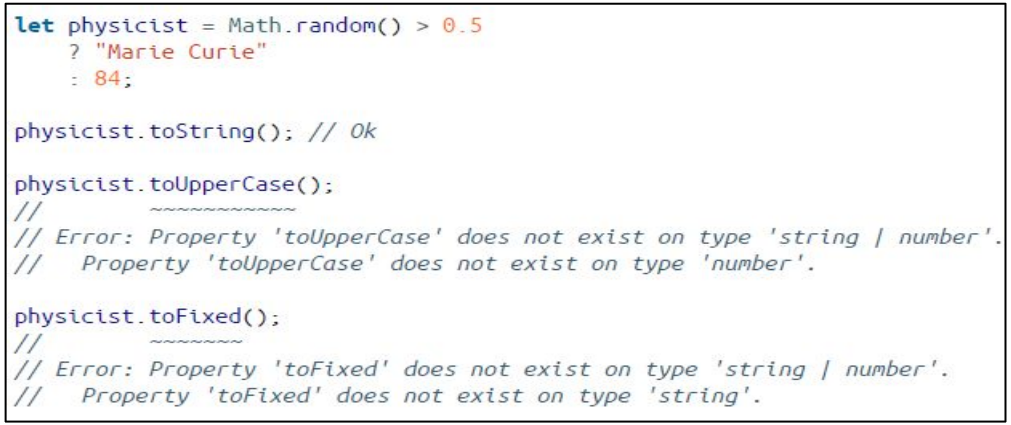
Union Properties

- TypeScript will only allow you to access member properties that exist on all possible types in the union.
- It will give you a type-checking error if you try to access a type that doesn’t exist on all possible types.

Union Types

Union Properties

Example



```
let physicist = Math.random() > 0.5
    ? "Marie Curie"
    : 84;

physicist.toString(); // Ok

physicist.toUpperCase();
//        ~~~~~~~~~~~~~
// Error: Property 'toUpperCase' does not exist on type 'string | number'.
//   Property 'toUpperCase' does not exist on type 'number'.

physicist.toFixed();
//        ~~~~~~~~
// Error: Property 'toFixed' does not exist on type 'string | number'.
//   Property 'toFixed' does not exist on type 'string'.
```

physicist is of type number | string. While .toString() exists in both types and is allowed to be used, (common properties)

.toUpperCase() and .toFixed() are not because .toUpperCase() is missing on the number type and .toFixed() is missing on the string type:

Narrowing

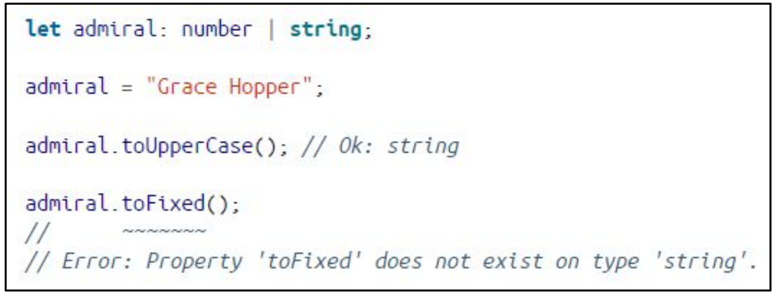
- Narrowing is when TypeScript infers from your code that a value is of a more specific type than what it was defined, declared, or previously inferred as.

- A logical check that can be used to narrow types is called a type guard.

Narrowing

Assignment Narrowing

If you directly assign a value to a variable, TypeScript will narrow the variable’s type to that value’s type.



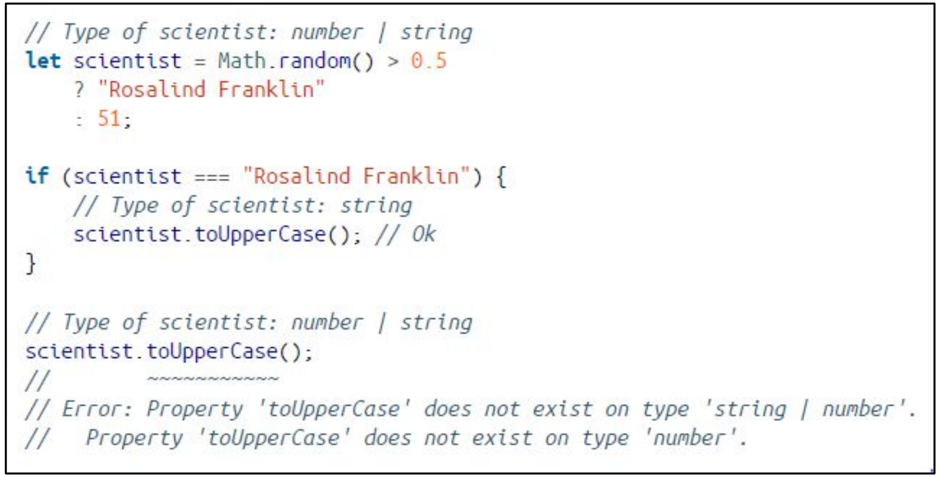
```
let admiral: number | string;

admiral = "Grace Hopper";

admiral.toUpperCase(); // Ok: string

admiral.toFixed();
//      ~~~~~~~
// Error: Property 'toFixed' does not exist on type 'string'.
```

admiral variable is declared initially as a number | string, but after being assigned the value "Grace Hopper", TypeScript knows it must be a string:

Narrowing

Conditional Checks

if statement checking the variable for being equal to a known value.



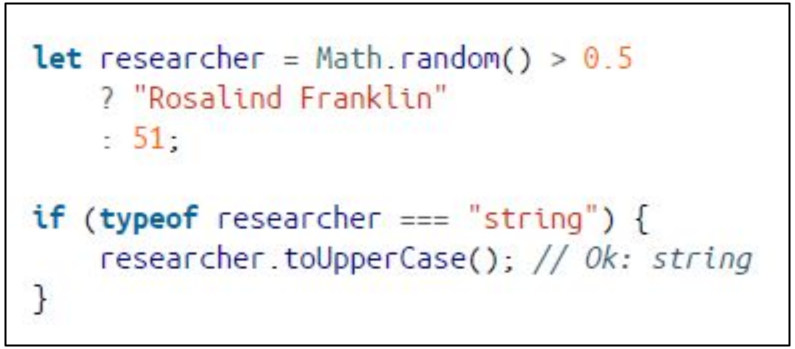
```
// Type of scientist: number | string
let scientist = Math.random() > 0.5
    ? "Rosalind Franklin"
    : 51;

if (scientist === "Rosalind Franklin") {
    // Type of scientist: string
    scientist.toUpperCase(); // Ok
}

// Type of scientist: number | string
scientist.toUpperCase();
//          ~~~~~~~~~~~
// Error: Property 'toUpperCase' does not exist on type 'string | number'.
//   Property 'toUpperCase' does not exist on type 'number'.
```

TypeScript is smart enough to understand that inside the body of that if statement, the variable must be the same type as the known value:

Narrowing

Typeof Checks

TypeScript also recognizes the typeof operator in narrowing down variable types.



```
let researcher = Math.random() > 0.5
    ? "Rosalind Franklin"
    : 51;

if (typeof researcher === "string") {
    researcher.toUpperCase(); // Ok: string
}
```

checking if typeof researcher is "string" indicates to TypeScript that the type of researcher must be string:

Literal Types

- When you declare a variable via var or let, you are telling the compiler that there is the chance that this variable will change its contents.
- In contrast, using const to declare a variable will inform TypeScript that this object will never change.