TypeScript Basics

Çağatay Sönmez

02.10.2021

Arcelik Electronics Plant, R&D Team – Software Design

Agenda

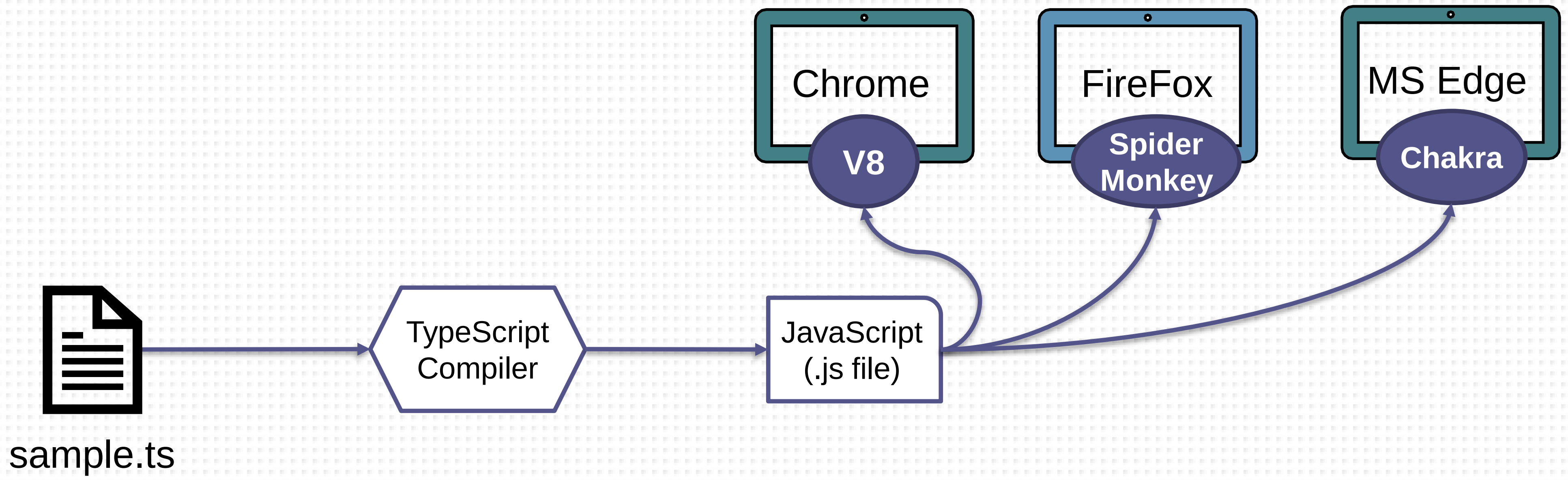
- What is TypeScript
- TypeScript Syntax
- TypeScript Variables & Types
- TypeScript Functions
- Compiling TypeScript Project
- TypeScript IDE - Visual Studio Code
- Using External (3rd Party) Libraries

Koç

What is TypeScript

- TypeScript is developed and maintained by Microsoft
- It is a superset of JavaScript
- Publicly released in October 2012
- TypeScript 1.0, 2.0, 3.0, 4.0 were released in 2014, 2016, 2018, 2020 respectivelly
- Can be used for both client-side and server-side applications, e.g.
  - ReactJS on client side
  - Node.js on server side

TypeScript Compiler



```
$ sudo apt install npm
$ npm install -g typescript
$ tsc --target es2015 sample.ts
```

TypeScript to JavaScript Example

```
class Greeter {
  greeting: string;

  constructor(message: string) {
    this.greeting = message;
  }

  greet() {
    console.log("Hello, " + this.greeting);
  }
}

let greeter = new Greeter("World");
greeter.greet();
```

TypeScript

Target: ES5

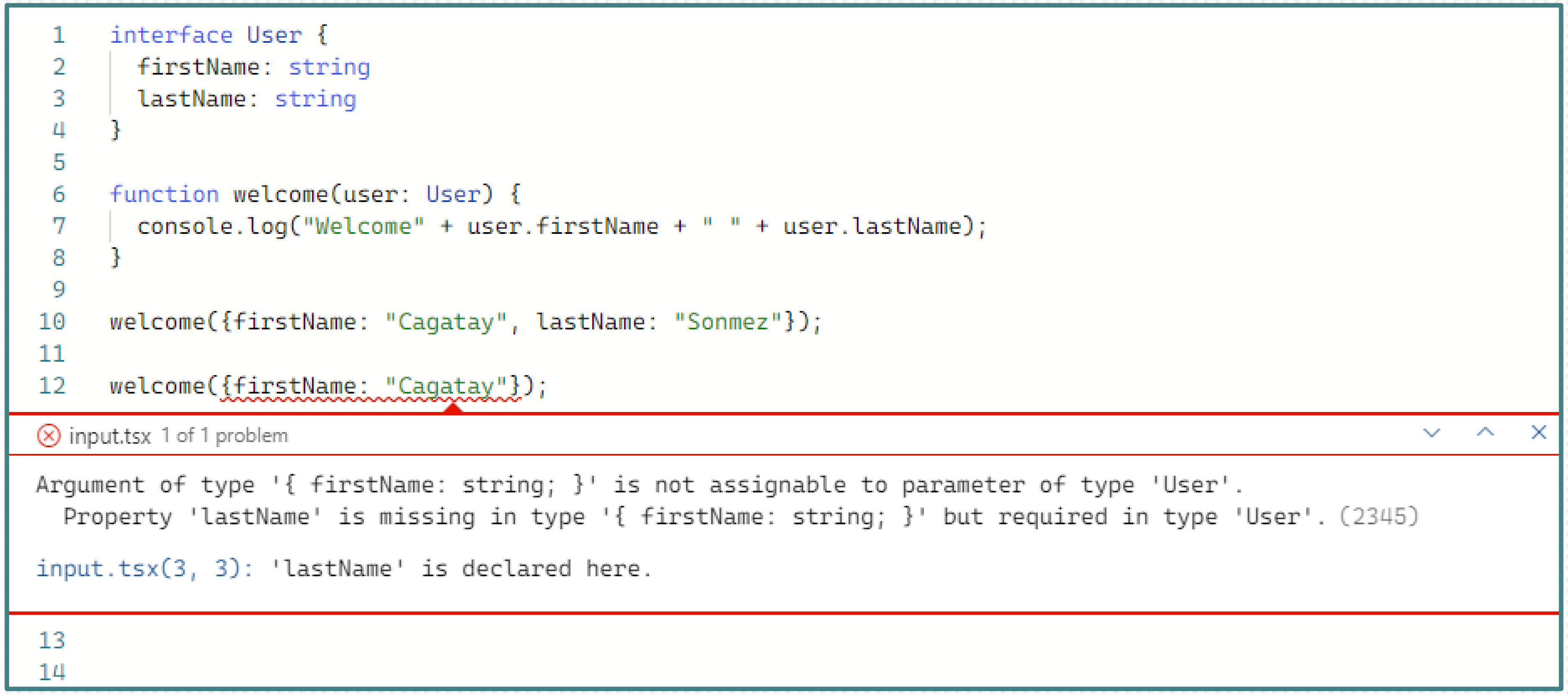
```
"use strict";
var Greeter = /** @class */ (function () {
    function Greeter(message) {
        this.greeting = message;
    }

    Greeter.prototype.greet = function () {
        console.log("Hello, " + this.greeting);
    };
    return Greeter;
}());

var greeter = new Greeter("World");
greeter.greet();
```

JavaScript

TypeScript vs JavaScript

TypeScript (TS)	JavaScript (JS)
Just-in-time compiled (interpreted)	Requires to be compiled
Highlights errors at compilation time	Shows errors at the runtime
Dynamic typing	Static typing
Object-oriented	Prototype-based
Supports class	Does not support class (until ES6)
Supports interface	Does not support interface
.ts file extension	.js file extension

Runtime Type Check



```
interface User {
  firstName: string
  lastName: string
}

function welcome(user: User) {
  console.log("Welcome" + user.firstName + " " + user.lastName);
}

welcome({firstName: "Cagatay", lastName: "Sonmez"});

welcome({firstName: "Cagatay"});
```

```
input.tsx 1 of 1 problem

Argument of type '{ firstName: string; }' is not assignable to parameter of type 'User'.
  Property 'lastName' is missing in type '{ firstName: string; }' but required in type 'User'. (2345)

input.tsx(3, 3): 'lastName' is declared here.
```

JavaScript

TypeScript

TypeScript Syntax

- TypeScript is a typed superset of JavaScript
  - All JavaScript code is valid TypeScript code
  - TypeScript adds a lot of new features on top of JavaScript

```
function myFunc(arg) {
    alert("argument is" + arg);
}
```

```
var flag = false;
while (flag == false) {
   alert("flag is true");
   flag = true;
}
```

```
if (x == y) {
                    alert("x is equal to y");
}
```

```
for (i=0; i<10; i++) {
   alert("i is: " + i);
}
```

```
switch (new Date().getDay()) {
   case 0:
      console.log("Sunday");
      break;
   default:
      console.log("Not Sunday");
}
```

```
var Shape = (function () {
   Shape.prototype.draw = function () {
      console.log("Drawing shape..");
   };
   return Shape;
}());
```

Declaring Variables and Type Annotations

- TypeScript doesn’t use “types on the left”-style declarations; Type annotations will always go after the thing being typed

```
var myName = "Alice";
var myName: string = "Alice";

const myId: number = 1234;

let myData: any;
myData = {1:false};
```

Variables declared with var has function scope

const variables cannot be changed

Variables declared with let has block scope

TypeScript tries to automatically infer the types, most of the time you don’t need to explicitly specify the type of the variable

Declare

Initialize

TypeScript Types - Primitives

- TypeScript uses the primitive types of JavaScript

boolean

```
var myBoolean = false;
var myBoolean: boolean = true;
```

inferred type (tsc marks this var as boolean)

explicit type (don't need to specify type)

number

```
var myFloat = 0.3555;
var myFloat: number;
```

string

```
var myString = "This is a string";
var myString: string;
```

Note: there is no int, double or float - everything is simply number

TypeScript Types - Arrays

- TypeScript uses Array type which is an Object in JavaScript

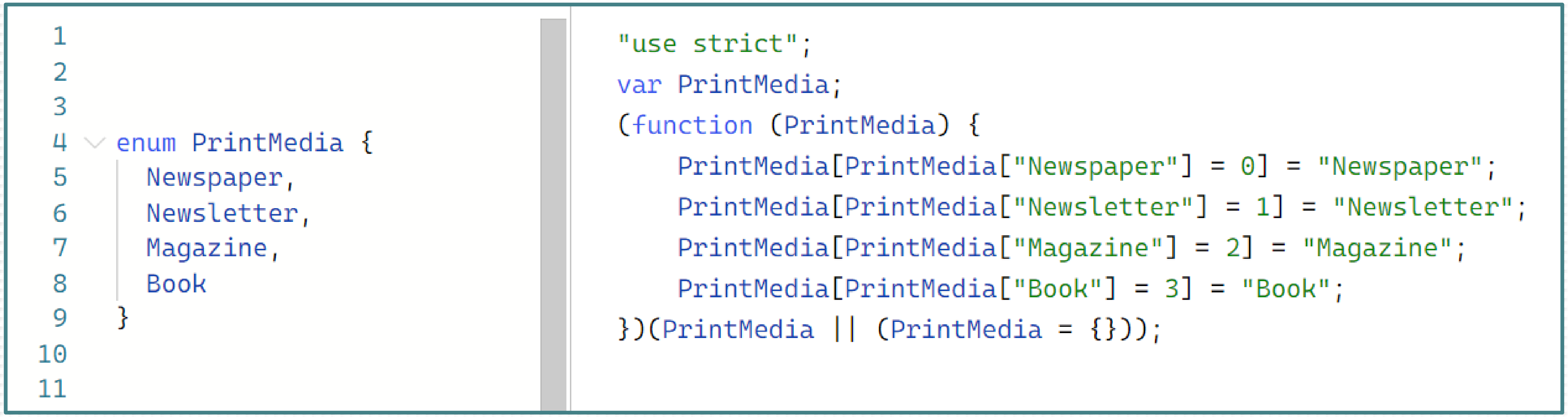
***type*[] or Array<*type*>**

```
var numberArray = [1, 2, 3];
var numberArray: number[];
var numberArray: number[] = [1, 2, 3];

var numberArray = Array (1, 2, 3);
var numberArray: Array<number>;
var numberArray: Array<number>(1,2,3);
```

declare and initialize number array

TypeScript Types - Enums

- Enum is a new data type supported in TypeScript
- Enums declare a set of named constants



```typescript



enum PrintMedia {
  Newspaper,
  Newsletter,
  Magazine,
  Book
}


```

```javascript
"use strict";
var PrintMedia;
(function (PrintMedia) {
    PrintMedia[PrintMedia["Newspaper"] = 0] = "Newspaper";
    PrintMedia[PrintMedia["Newsletter"] = 1] = "Newsletter";
    PrintMedia[PrintMedia["Magazine"] = 2] = "Magazine";
    PrintMedia[PrintMedia["Book"] = 3] = "Book";
})(PrintMedia || (PrintMedia = {}));
```

TypeScript Types - Tuples

- Tupple is a new data type supported in TypeScript
- Tuple can contain two values of different data types

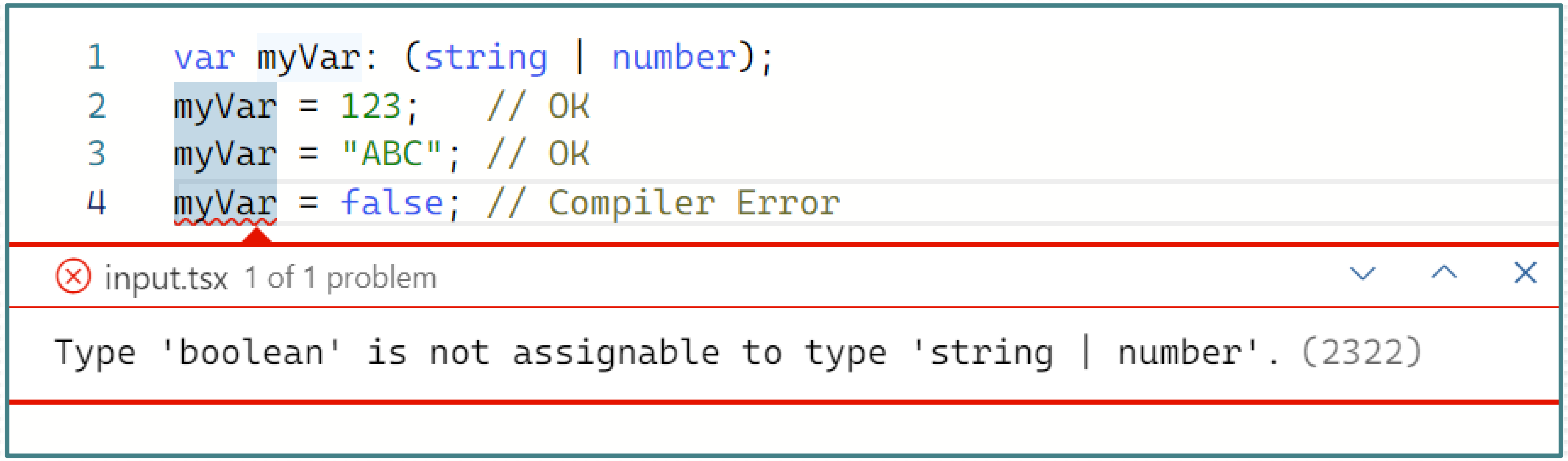
```
var employee: [number, string];
employee = [1, "Cagatay"]
```

$\rightarrow$ Declaring tupple

$\rightarrow$ Initializing tupple

TypeScript Types - Unions

- Union is used to declare more than one data type for a variable or a function parameter



```
var myVar: (string | number);
myVar = 123;    // OK
myVar = "ABC"; // OK
myVar = false; // Compiler Error
```

input.tsx 1 of 1 problem

```
Type 'boolean' is not assignable to type 'string | number'. (2322)
```

TypeScript Types - void

- Only undefined is assignable to void variables
- void type is aslo used to declare that a function returns nothing

```
let nothing: void = undefined;

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

nothing = greet();
```

Function returns nothing

TypeScript Types - any

- If you don’t know the type, or want to typechecking errors, you can use any
- Default type of TypeScript is any

any

```
var obj;
var obj: any;
```

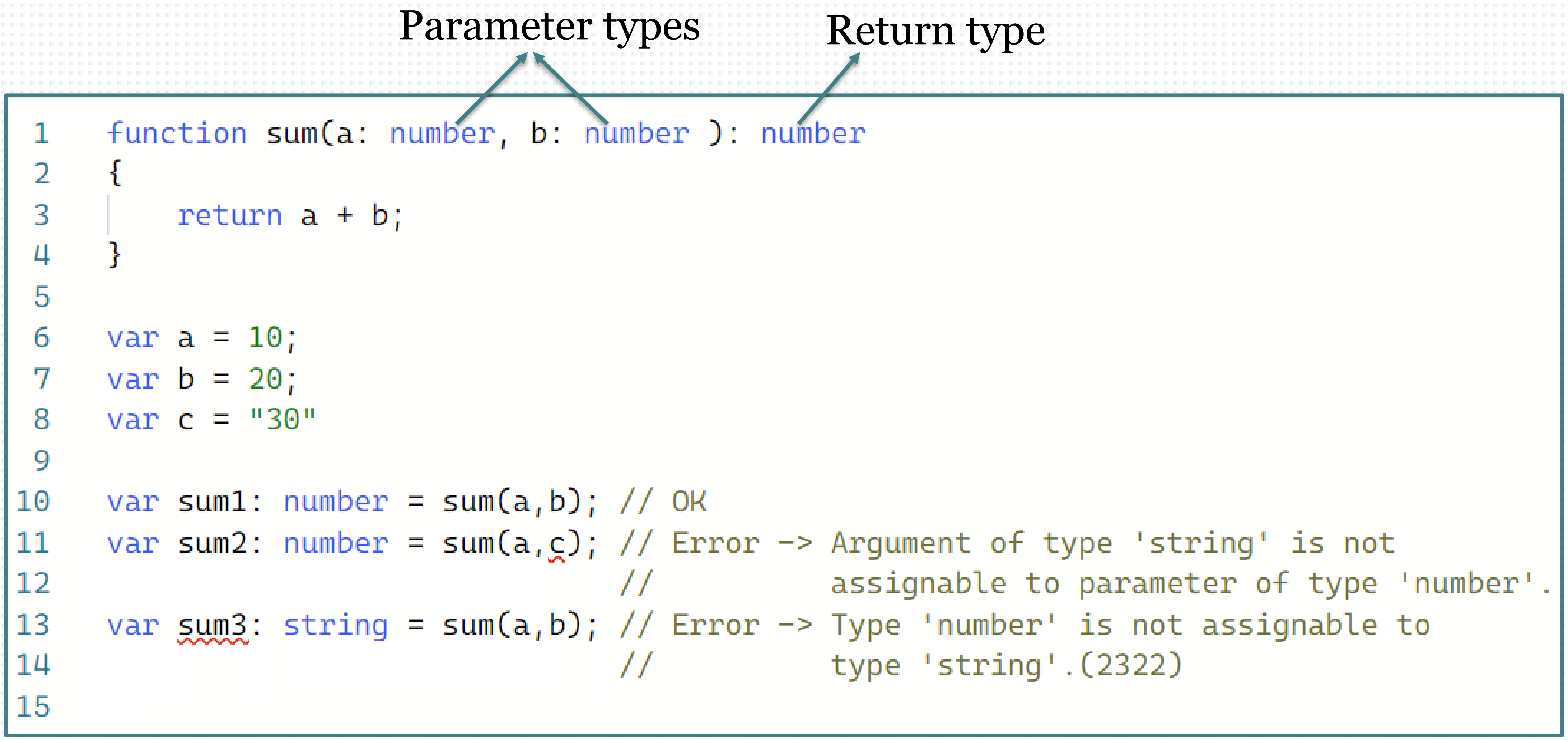
```
var obj: any = { x: 0 };

obj.foo();
obj();
obj.bar = 100;
obj = "hello";
```

No compile error

Runtime error (obj.foo is not a function)

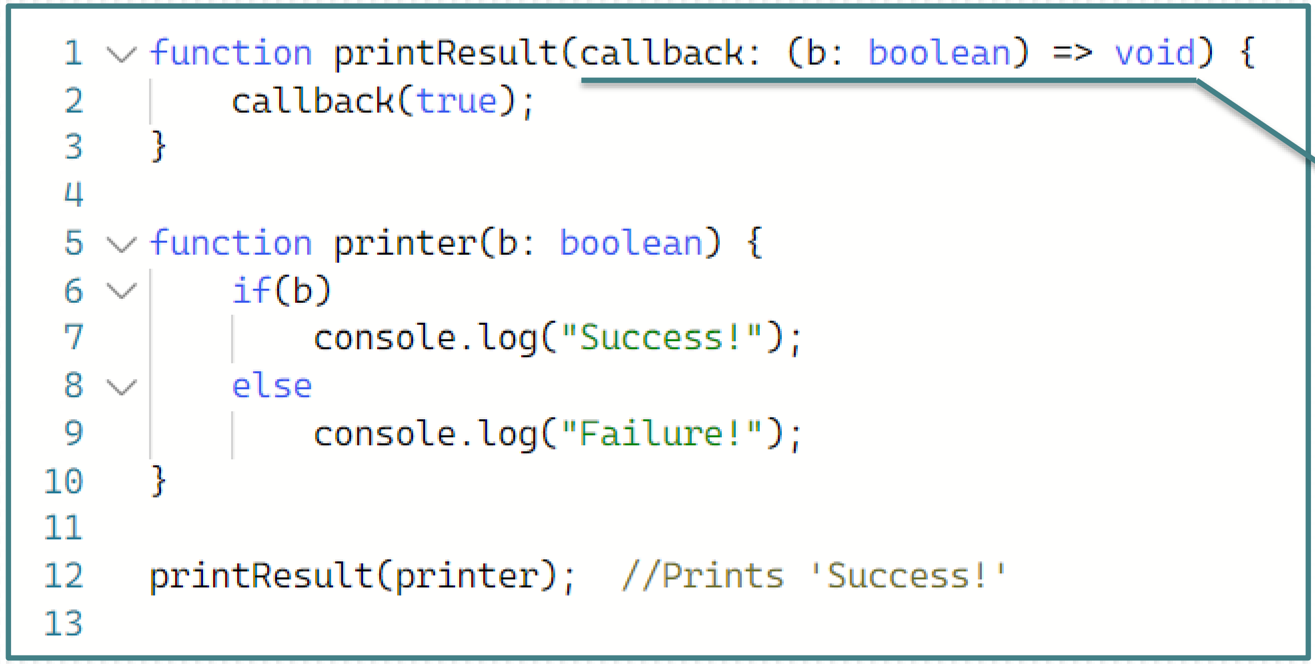
Runtime error (obj is not a function)

Function Annotations



```
function sum(a: number, b: number ): number
{
    return a + b;
}

var a = 10;
var b = 20;
var c = "30"

var sum1: number = sum(a,b); // OK
var sum2: number = sum(a,c); // Error -> Argument of type 'string' is not
                             //          assignable to parameter of type 'number'.
var sum3: string = sum(a,b); // Error -> Type 'number' is not assignable to
                             //          type 'string'.(2322)

```

Function as a Parameter

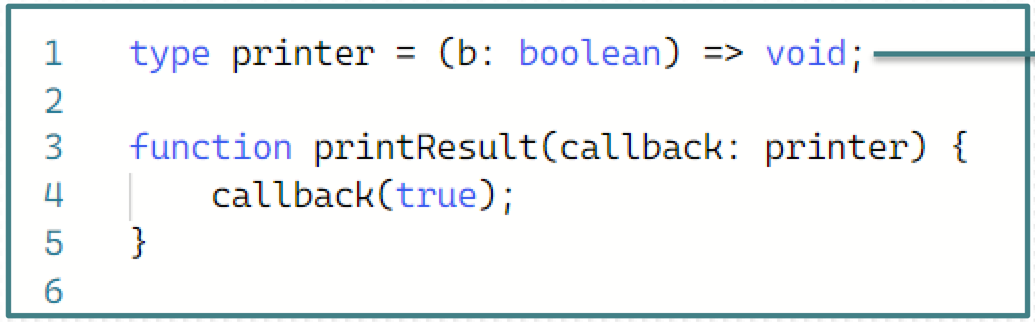


```
function printResult(callback: (b: boolean) => void) {
    callback(true);
}

function printer(b: boolean) {
    if(b)
        console.log("Success!");
    else
        console.log("Failure!");
}

printResult(printer);  //Prints 'Success!'

```

A function argument which has a boolean parameter and returns void (nothing)

Type Alias

- Use aliases for better readability

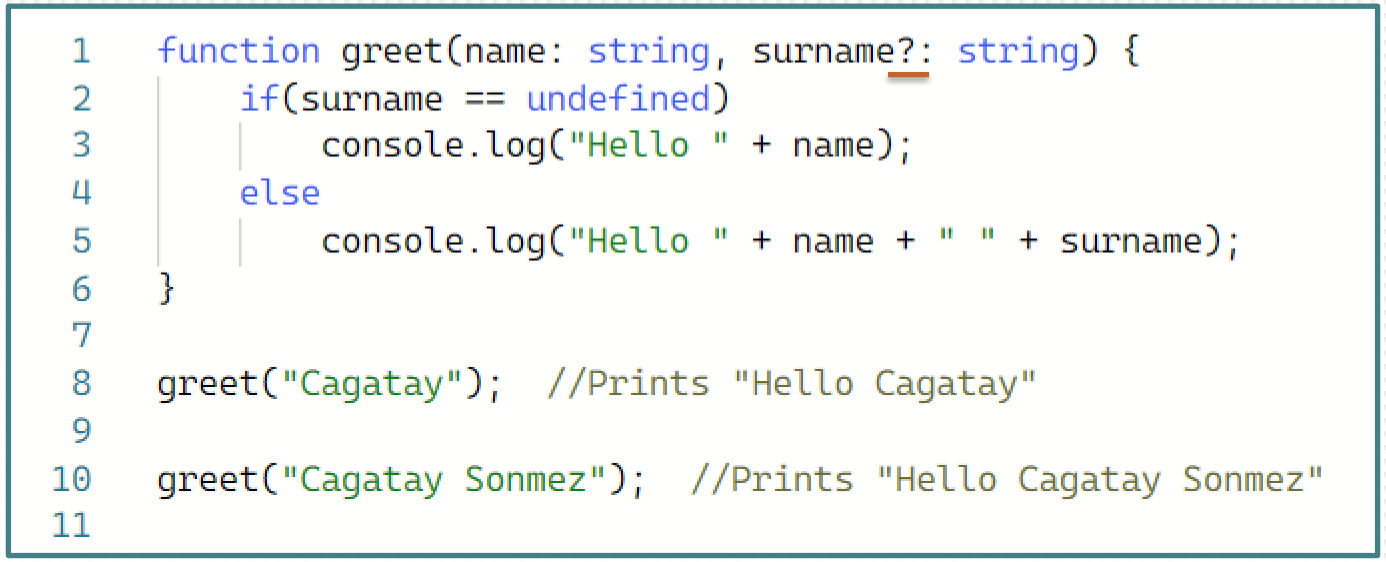


```
type printer = (b: boolean) => void;

function printResult(callback: printer) {
    callback(true);
}

```

Type alias

Optional Parameters

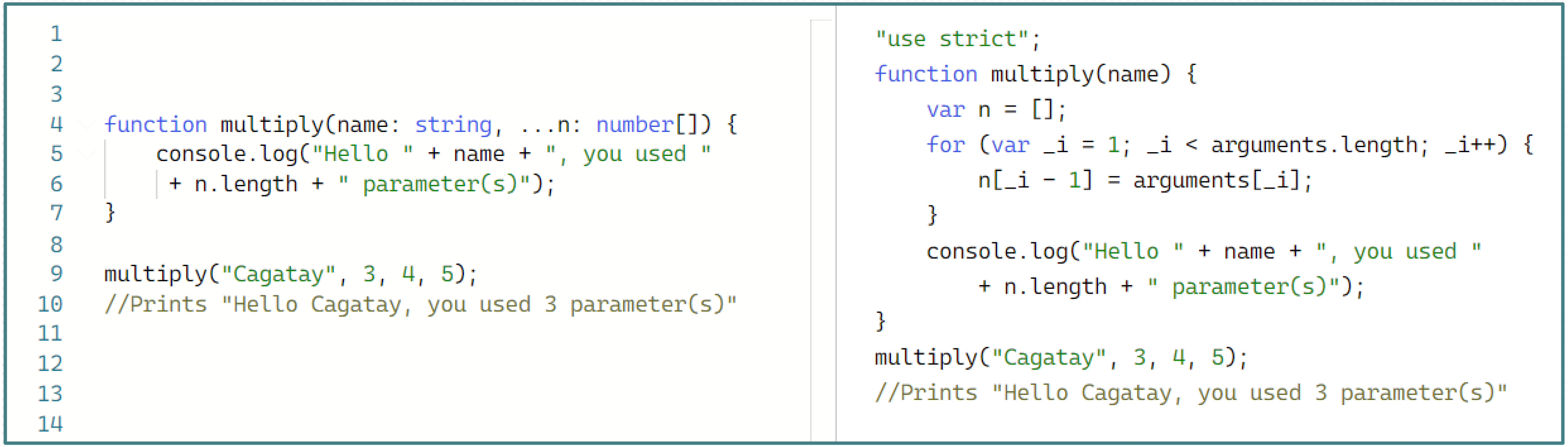
- Use ? to mark the parameter as optional
- A required parameter cannot follow an optional parameter.



```typescript
function greet(name: string, surname?: string) {
    if(surname == undefined)
        console.log("Hello " + name);
    else
        console.log("Hello " + name + " " + surname);
}

greet("Cagatay");  //Prints "Hello Cagatay"

greet("Cagatay Sonmez");  //Prints "Hello Cagatay Sonmez"

```

Rest Parameter

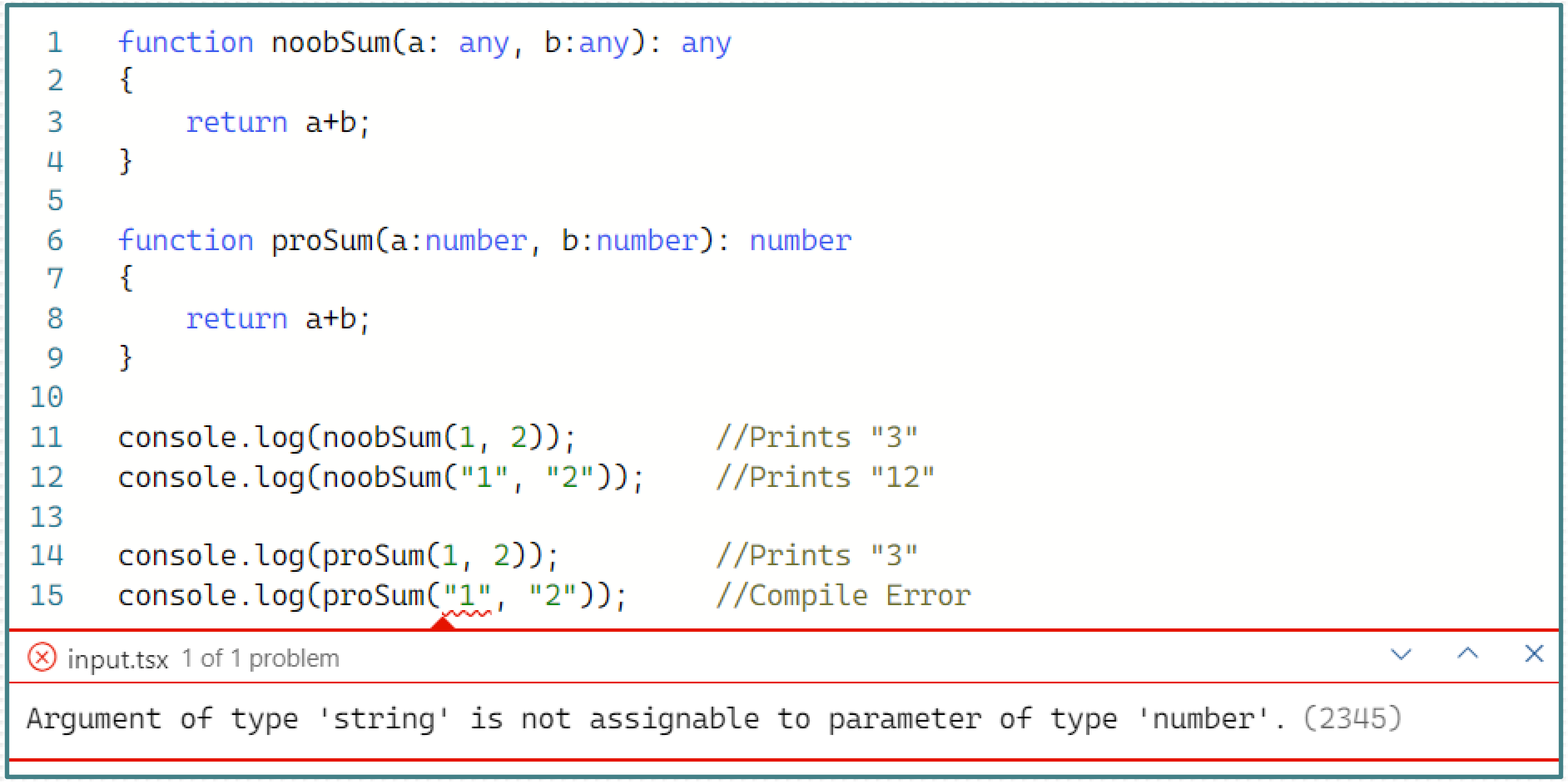
- Use … to declare rest parameter (variable number of arguments)
- A rest parameter must be of an array type
- A rest parameter must be last in a parameter list



```typescript



function multiply(name: string, ...n: number[]) {
    console.log("Hello " + name + ", you used "
      + n.length + " parameter(s)");
}

multiply("Cagatay", 3, 4, 5);
//Prints "Hello Cagatay, you used 3 parameter(s)"
```

```javascript
"use strict";
function multiply(name) {
    var n = [];
    for (var _i = 1; _i < arguments.length; _i++) {
        n[_i - 1] = arguments[_i];
    }
    console.log("Hello " + name + ", you used "
        + n.length + " parameter(s)");
}
multiply("Cagatay", 3, 4, 5);
//Prints "Hello Cagatay, you used 3 parameter(s)"
```

Why Types Matters?



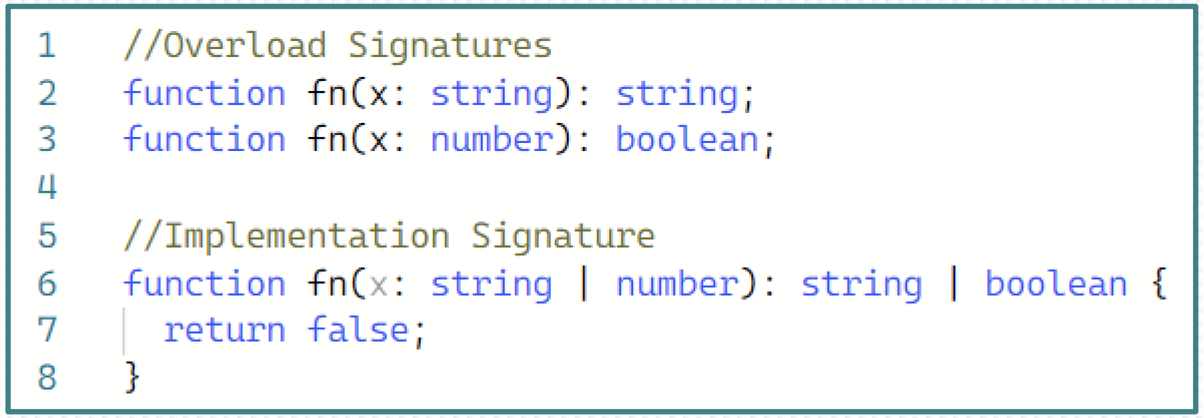
```
function noobSum(a: any, b:any): any
{
    return a+b;
}

function proSum(a:number, b:number): number
{
    return a+b;
}

console.log(noobSum(1, 2));         //Prints "3"
console.log(noobSum("1", "2"));     //Prints "12"

console.log(proSum(1, 2));          //Prints "3"
console.log(proSum("1", "2"));      //Compile Error
```

input.tsx 1 of 1 problem

```
Argument of type 'string' is not assignable to parameter of type 'number'. (2345)
```

Function Overloading

- Function overloading is supported by writing overload signatures
- There must be one implementation which is called implementation signature
- Overload signatures and the implementation signature should be compatible
- You should always have two or more signatures above the implementation of the function

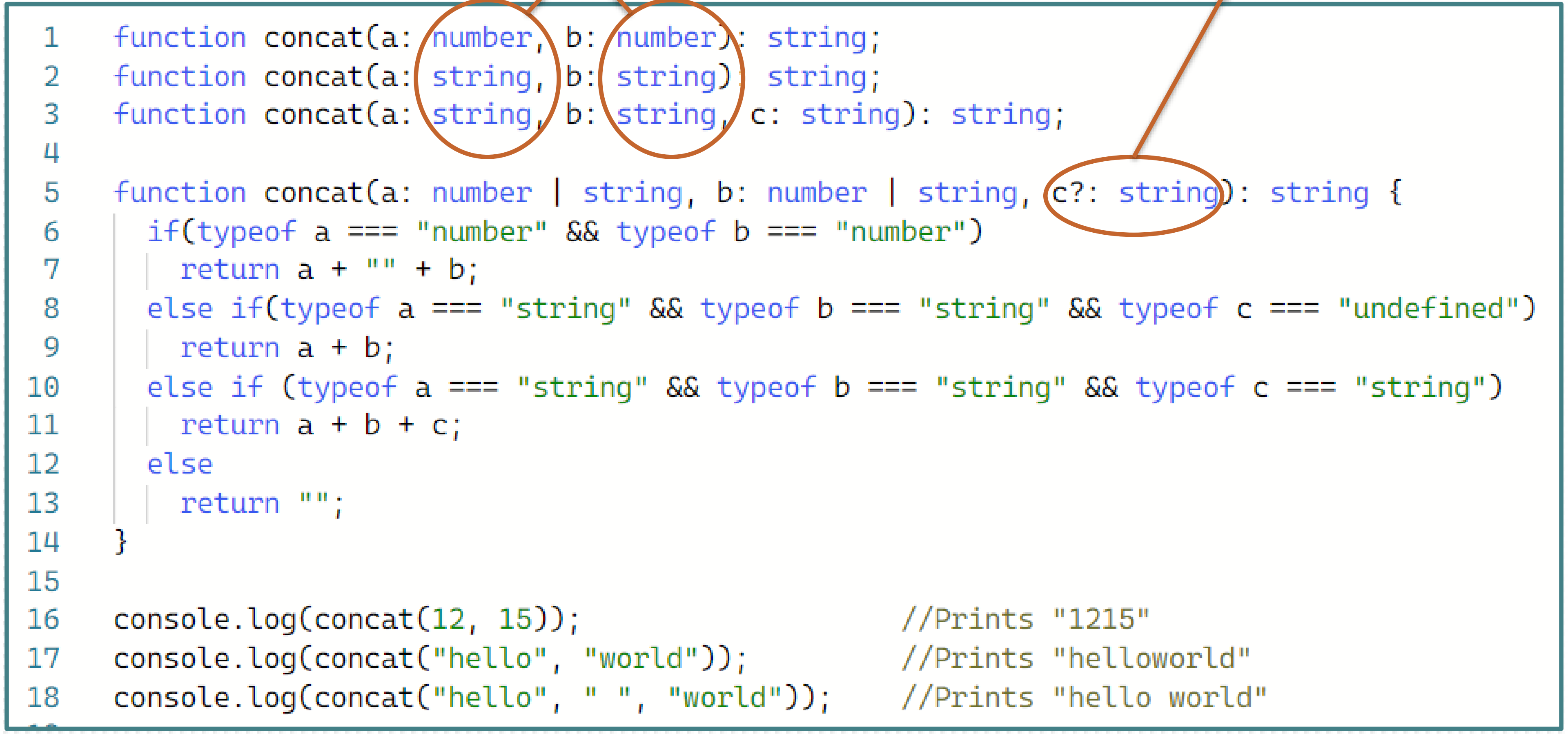


```typescript
//Overload Signatures
function fn(x: string): string;
function fn(x: number): boolean;

//Implementation Signature
function fn(x: string | number): string | boolean {
  return false;
}
```

Function Overloading Example

Type of *a* and *b* can be a number or string

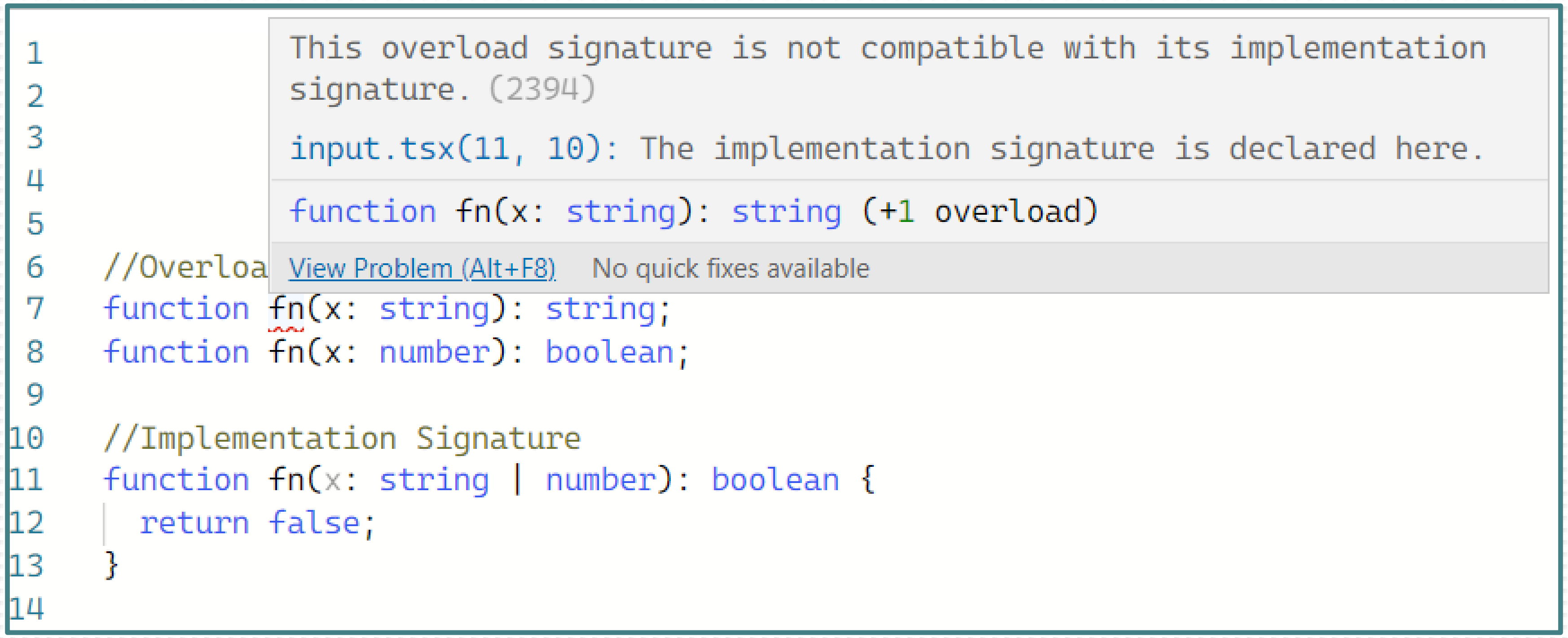
3rd argument is an optional string



```
function concat(a: number, b: number): string;
function concat(a: string, b: string): string;
function concat(a: string, b: string, c: string): string;

function concat(a: number | string, b: number | string, c?: string): string {
  if(typeof a === "number" && typeof b === "number")
    return a + "" + b;
  else if(typeof a === "string" && typeof b === "string" && typeof c === "undefined")
    return a + b;
  else if (typeof a === "string" && typeof b === "string" && typeof c === "string")
    return a + b + c;
  else
    return "";
}

console.log(concat(12, 15));                //Prints "1215"
console.log(concat("hello", "world"));      //Prints "helloworld"
console.log(concat("hello", " ", "world")); //Prints "hello world"
```

Function Overloading Error 1

- Overload signatures and the implementation signature should be compatible!

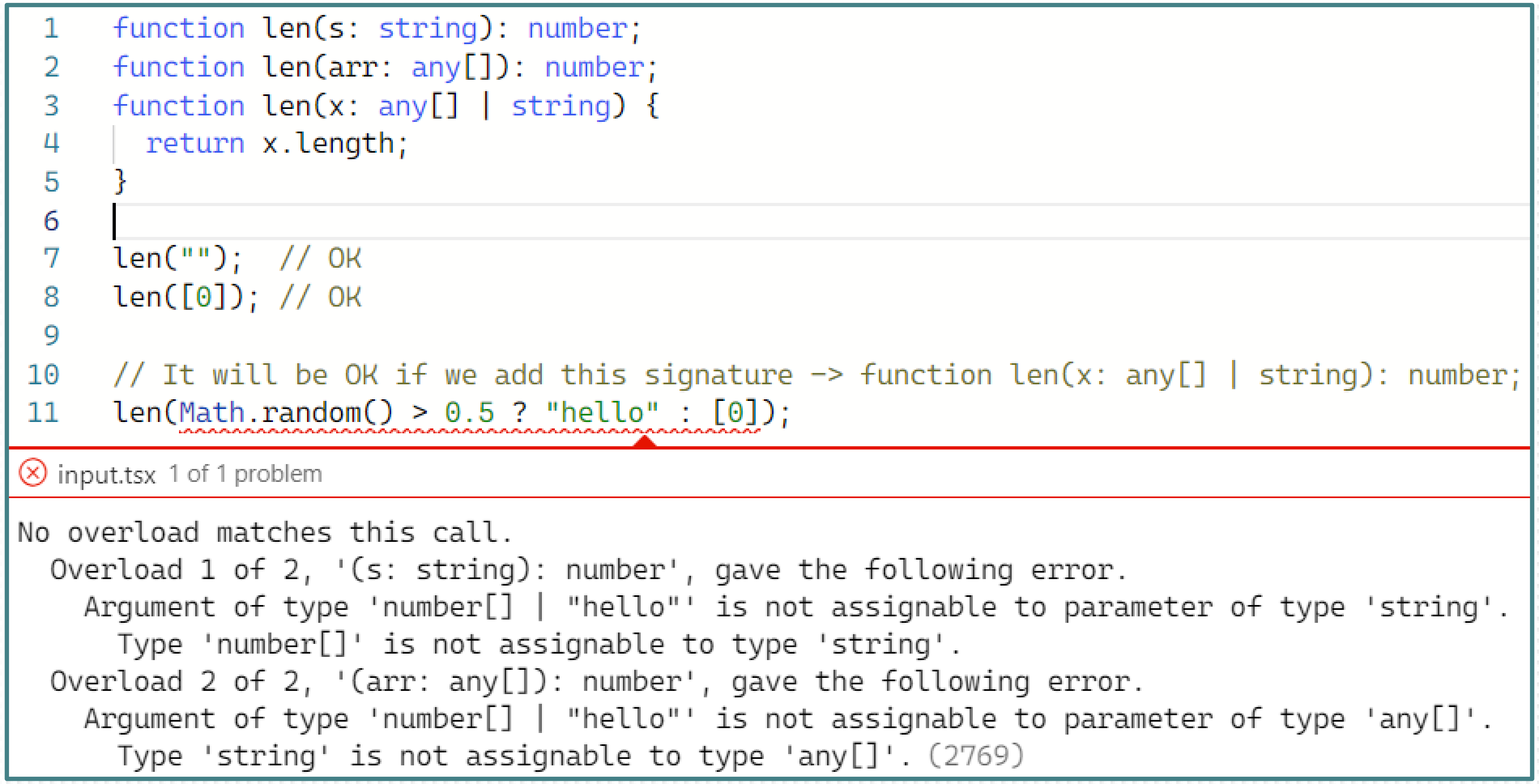


```
This overload signature is not compatible with its implementation
signature. (2394)

input.tsx(11, 10): The implementation signature is declared here.

function fn(x: string): string (+1 overload)
```

```
//Overloa
function fn(x: string): string;
function fn(x: number): boolean;

//Implementation Signature
function fn(x: string | number): boolean {
  return false;
}
```

Function Overloading Error

- Prefer parameters with union types instead of overloads when possible!



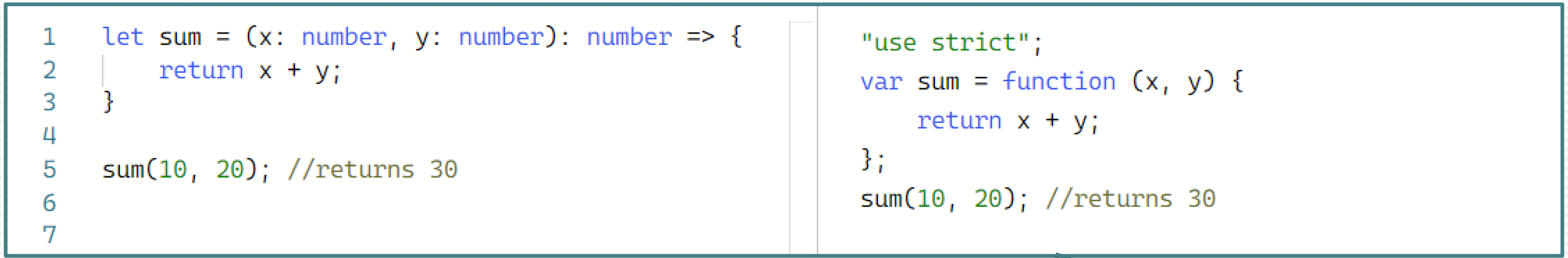
```
function len(s: string): number;
function len(arr: any[]): number;
function len(x: any[] | string) {
  return x.length;
}

len("");  // OK
len([0]); // OK

// It will be OK if we add this signature -> function len(x: any[] | string): number;
len(Math.random() > 0.5 ? "hello" : [0]);
```

input.tsx 1 of 1 problem

```
No overload matches this call.
  Overload 1 of 2, '(s: string): number', gave the following error.
    Argument of type 'number[] | "hello"' is not assignable to parameter of type 'string'.
      Type 'number[]' is not assignable to type 'string'.
  Overload 2 of 2, '(arr: any[]): number', gave the following error.
    Argument of type 'number[] | "hello"' is not assignable to parameter of type 'any[]'.
      Type 'string' is not assignable to type 'any[]'. (2769)
```

Arrow Functions

- A compact alternative to a traditional function expression
- Remove "function" keyword and place arrow between the argument and opening body bracket
  - function (a){ … } ➡ (a) => { … }

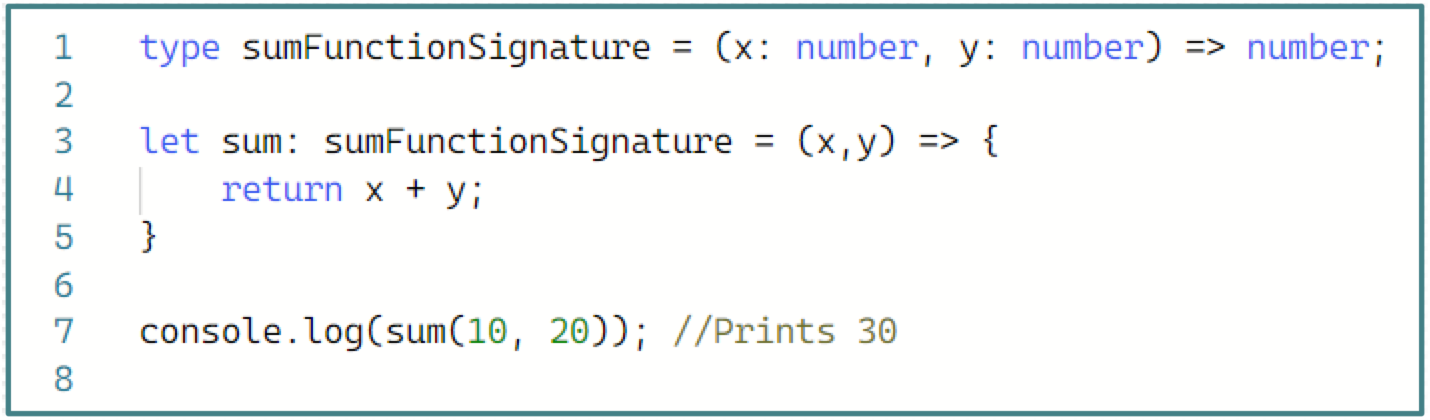


```
let sum = (x: number, y: number): number => {
    return x + y;
}

sum(10, 20); //returns 30


```

```
"use strict";
var sum = function (x, y) {
    return x + y;
};
sum(10, 20); //returns 30
```

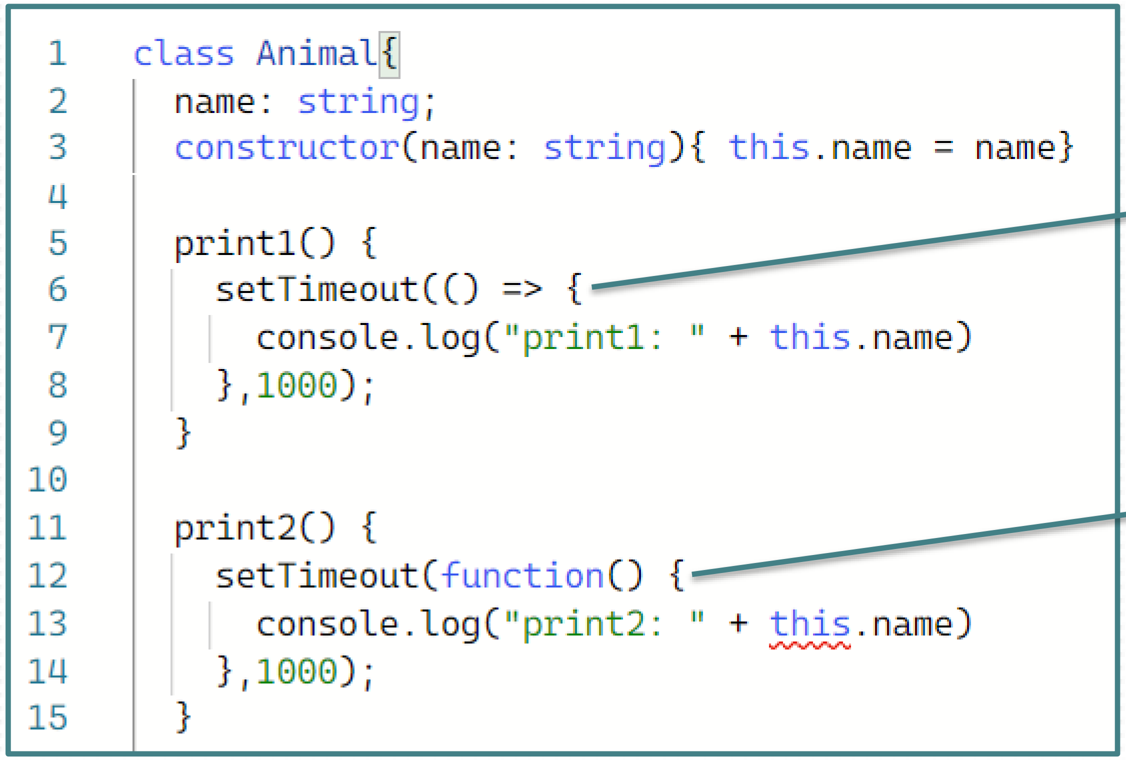
Target: ES5

Arrow Functions with Type Alias



```
type sumFunctionSignature = (x: number, y: number) => number;

let sum: sumFunctionSignature = (x,y) => {
    return x + y;
}

console.log(sum(10, 20)); //Prints 30

```

Arrow Function vs Regular Function

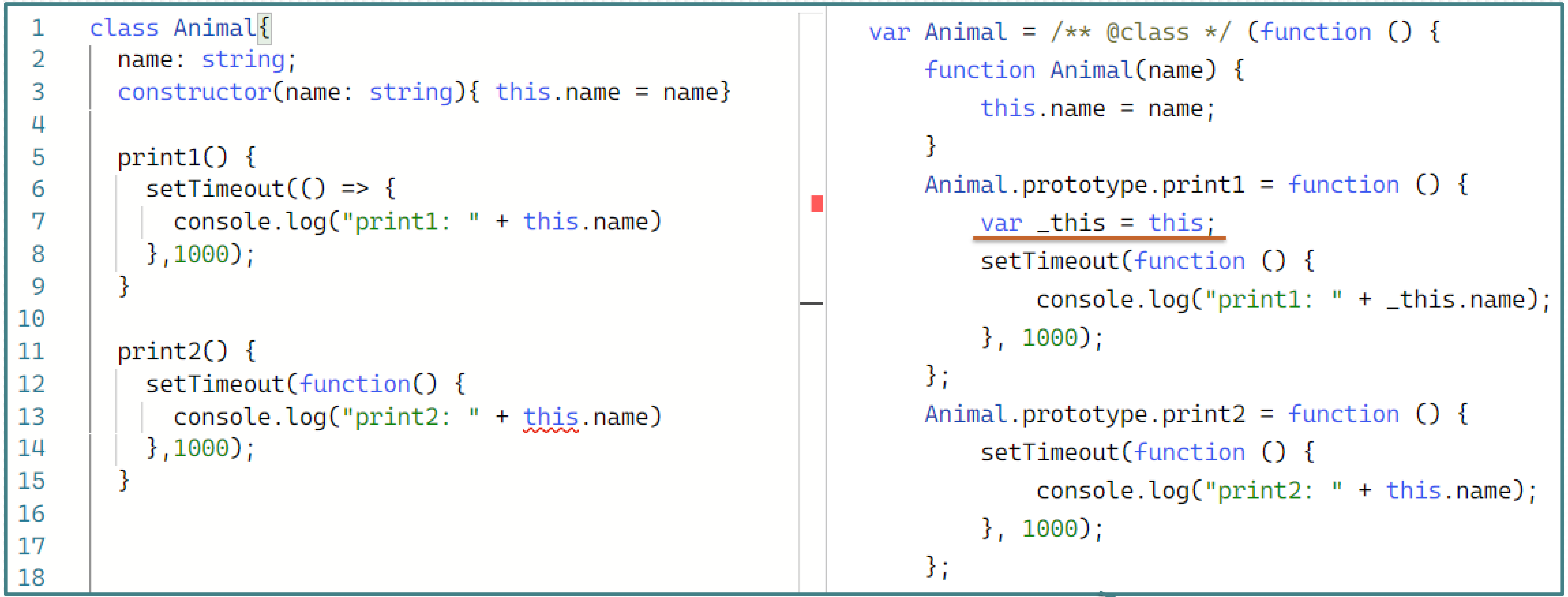


```
class Animal{
  name: string;
  constructor(name: string){ this.name = name}

  print1() {
    setTimeout(() => {
      console.log("print1: " + this.name)
    },1000);
  }

  print2() {
    setTimeout(function() {
      console.log("print2: " + this.name)
    },1000);
  }
```

Uses the scope of Animal class

Uses the scope of the block from which print2 function is called!

Arrow Function vs Regular Function Cont.

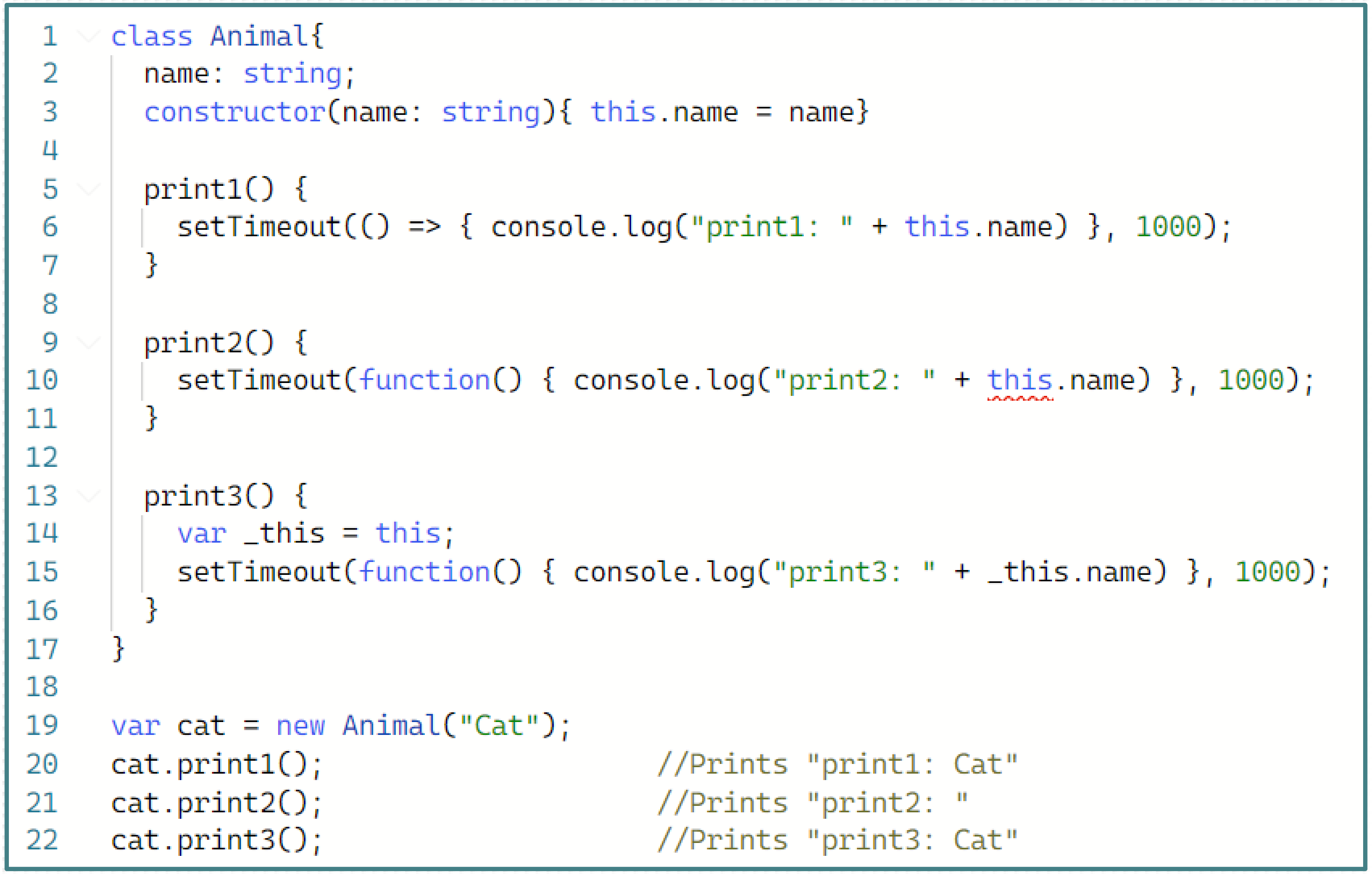


```
class Animal{
  name: string;
  constructor(name: string){ this.name = name}

  print1() {
    setTimeout(() => {
      console.log("print1: " + this.name)
    },1000);
  }

  print2() {
    setTimeout(function() {
      console.log("print2: " + this.name)
    },1000);
  }
```

```
var Animal = /** @class */ (function () {
    function Animal(name) {
        this.name = name;
    }
    Animal.prototype.print1 = function () {
        var _this = this;
        setTimeout(function () {
            console.log("print1: " + _this.name);
        }, 1000);
    };
    Animal.prototype.print2 = function () {
        setTimeout(function () {
            console.log("print2: " + this.name);
        }, 1000);
    };
```

Target: ES5

Arrow Function vs Regular Function Cont.



```typescript
class Animal{
  name: string;
  constructor(name: string){ this.name = name}

  print1() {
    setTimeout(() => { console.log("print1: " + this.name) }, 1000);
  }

  print2() {
    setTimeout(function() { console.log("print2: " + this.name) }, 1000);
  }

  print3() {
    var _this = this;
    setTimeout(function() { console.log("print3: " + _this.name) }, 1000);
  }
}

var cat = new Animal("Cat");
cat.print1();                    //Prints "print1: Cat"
cat.print2();                    //Prints "print2: "
cat.print3();                    //Prints "print3: Cat"
```

Compiling Project

- TypeScript files can be compiled using the tsc <file name>.ts command
- Compiling a large project with multiple files with tsc command is difficult
- TypeScript supports tsconfig.json to compile whole project at once
- If tsc is invoked with no input files, the compiler searches for the tsconfig.json file starting in the current directory and continuing up the parent directory chain
- See all compile options below
  - https://www.typescriptlang.org/tsconfig

tsconfig.json Example

Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es6', or 'es2015'.

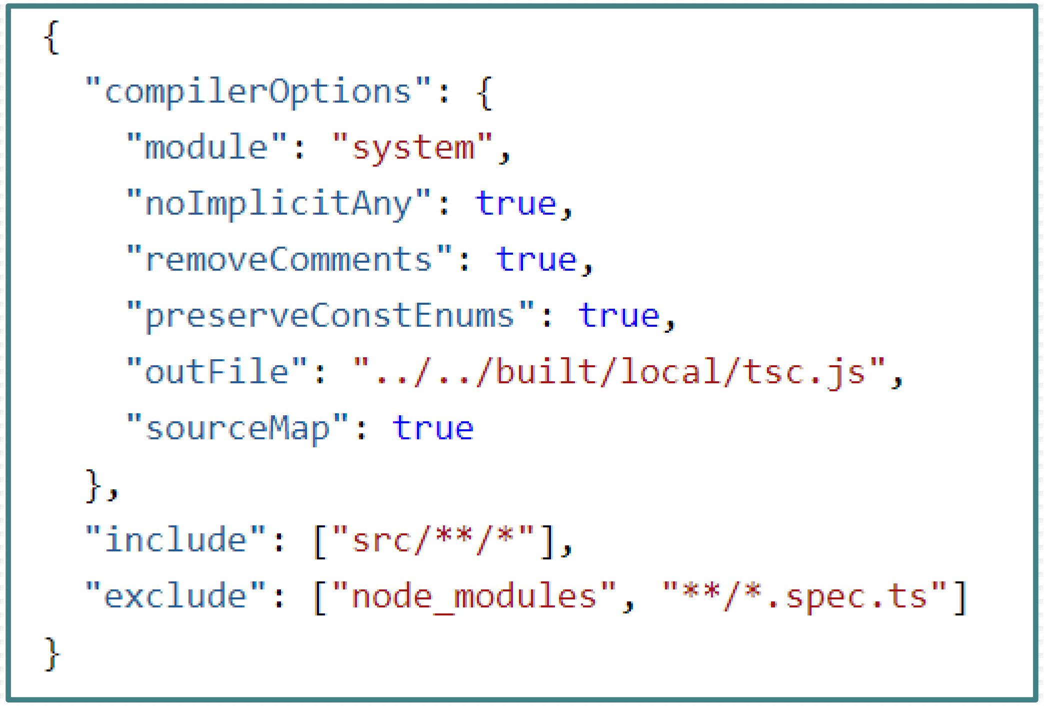
```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "removeComments": true,
    "sourceMap": true
  },
}
```

Specify ECMAScript target version: 'es3' (default), 'es5', or 'es6'.

tsconfig.json – files property

```json
{
  "compilerOptions": {
    "module": "commonjs",
    "noImplicitAny": true,
    "removeComments": true,
    "preserveConstEnums": true,
    "sourceMap": true
  },
  "files": [
    "core.ts",
    "sys.ts",
    "types.ts",
    "scanner.ts",
    "parser.ts",
    "utilities.ts",
  ]
}
```

tsconfig.json – include & exclude properties

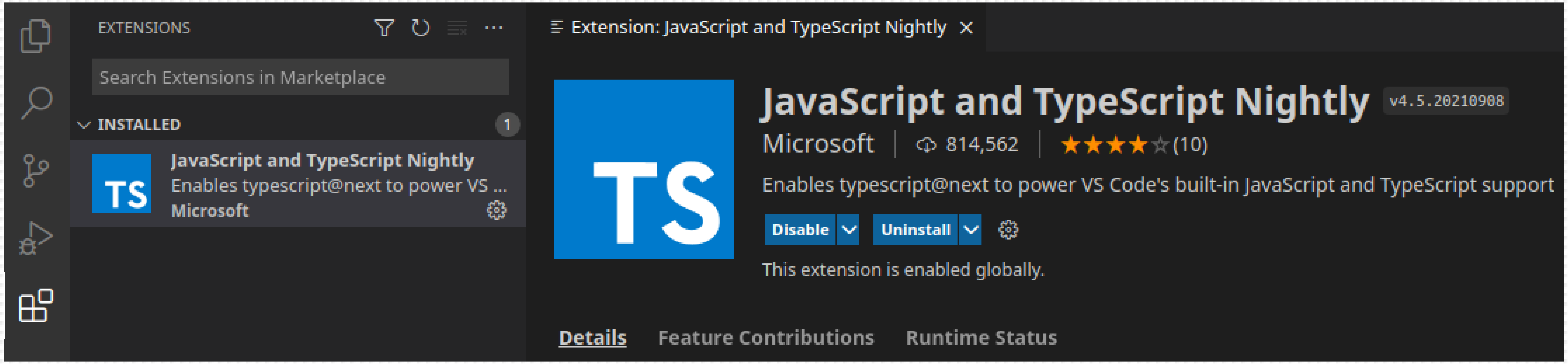


```json
{
  "compilerOptions": {
    "module": "system",
    "noImplicitAny": true,
    "removeComments": true,
    "preserveConstEnums": true,
    "outFile": "../../built/local/tsc.js",
    "sourceMap": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "**/*.spec.ts"]
}
```

Note: 'amd' and 'system’ module options can be used in conjunction with --outFile.

TypeScript IDE - Visual Studio Code

- Visual Studio Code includes TypeScript language support
- However, typescript compiler is not included by default
- You can consider installing TS extensions

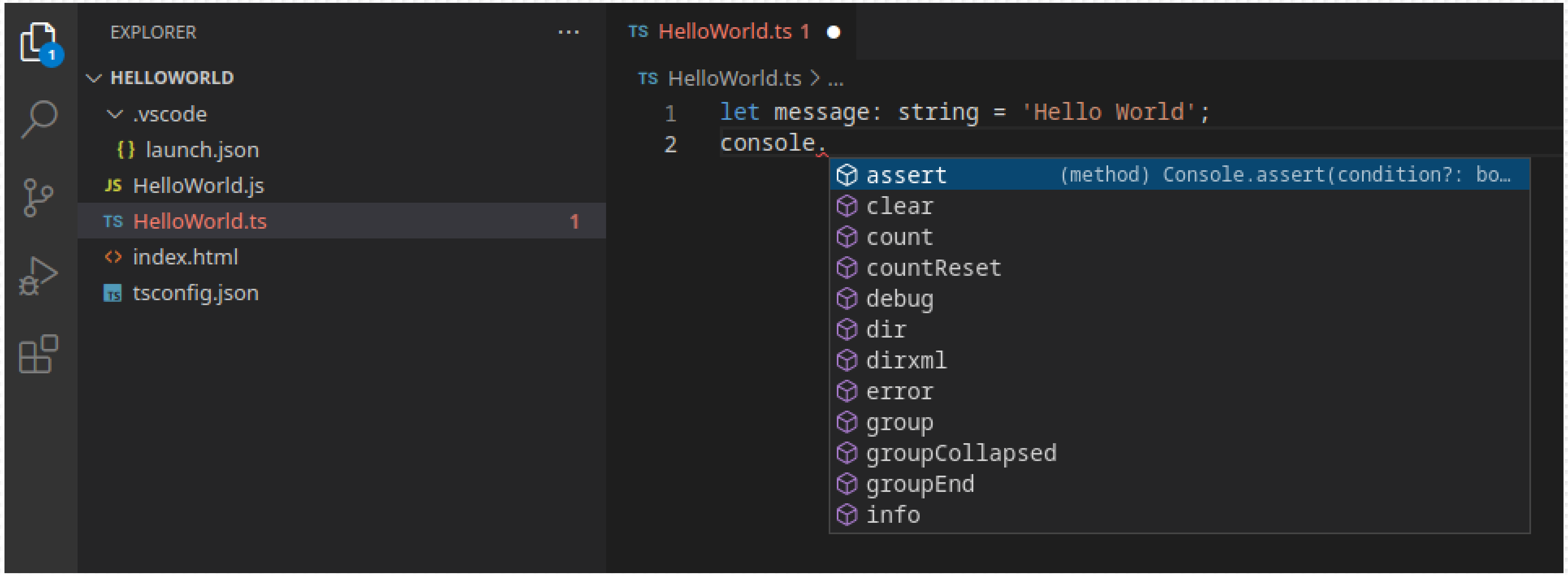


Extension: JavaScript and TypeScript Nightly

JavaScript and TypeScript Nightly
Enables typescript@next to power VS ...
Microsoft

JavaScript and TypeScript Nightly v4.5.20210908
Microsoft | 814,562 | ★★★★☆ (10)
Enables typescript@next to power VS Code's built-in JavaScript and TypeScript support
This extension is enabled globally.

Why VS Code - Intellisense

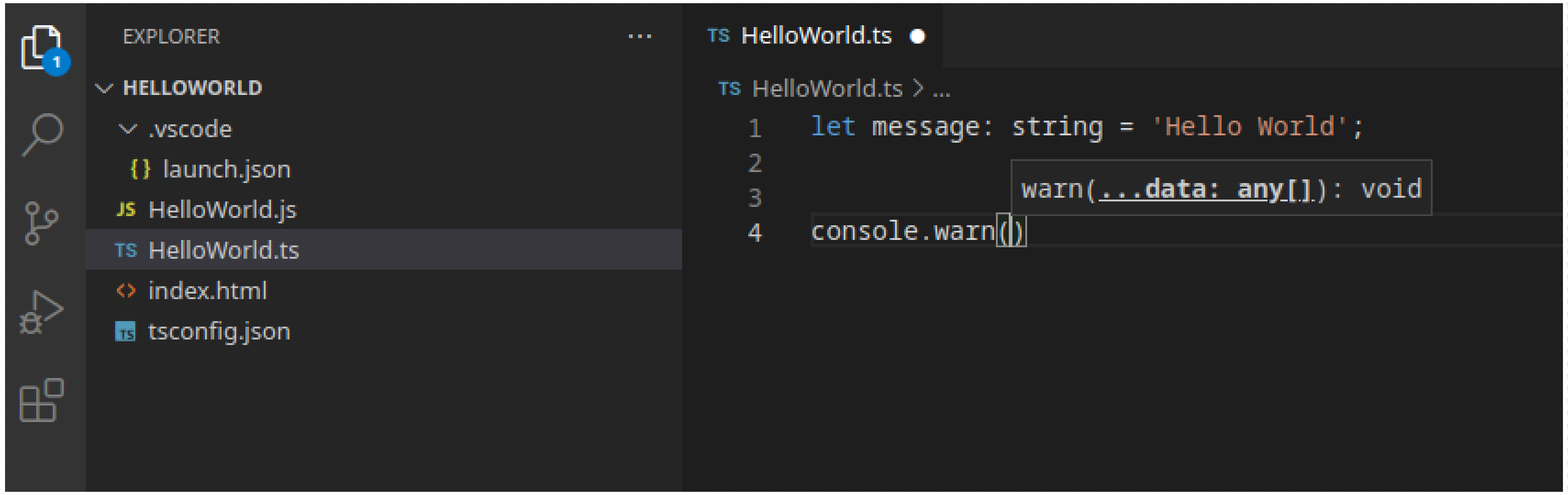


```
let message: string = 'Hello World';
console.
```

```
assert          (method) Console.assert(condition?: bo…
clear
count
countReset
debug
dir
dirxml
error
group
groupCollapsed
groupEnd
info
```

Why VS Code – Hover Information

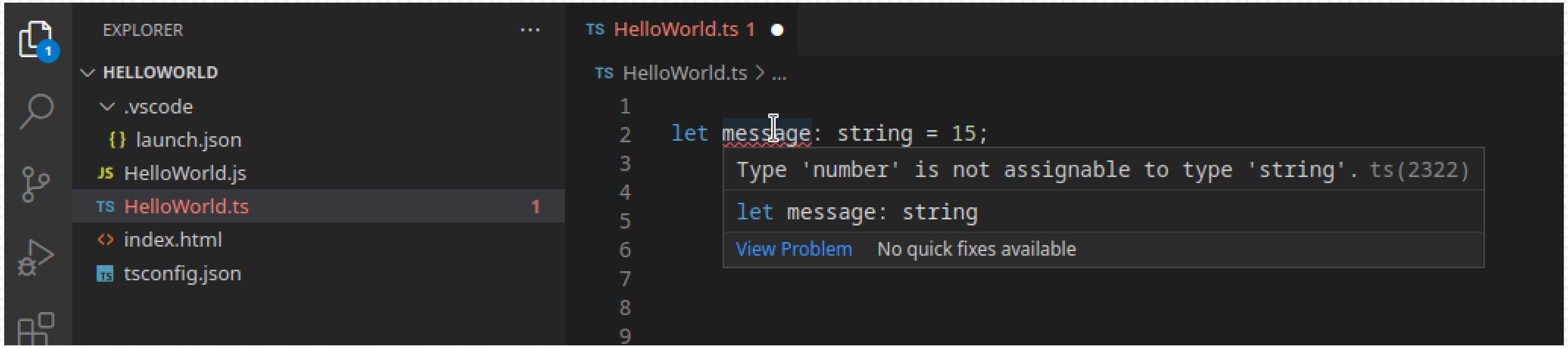
- Hover information will be shown when you select a method to get parameter help



```
let message: string = 'Hello World';
            warn(...data: any[]): void

console.warn()
```

Why VS Code – Error Checking

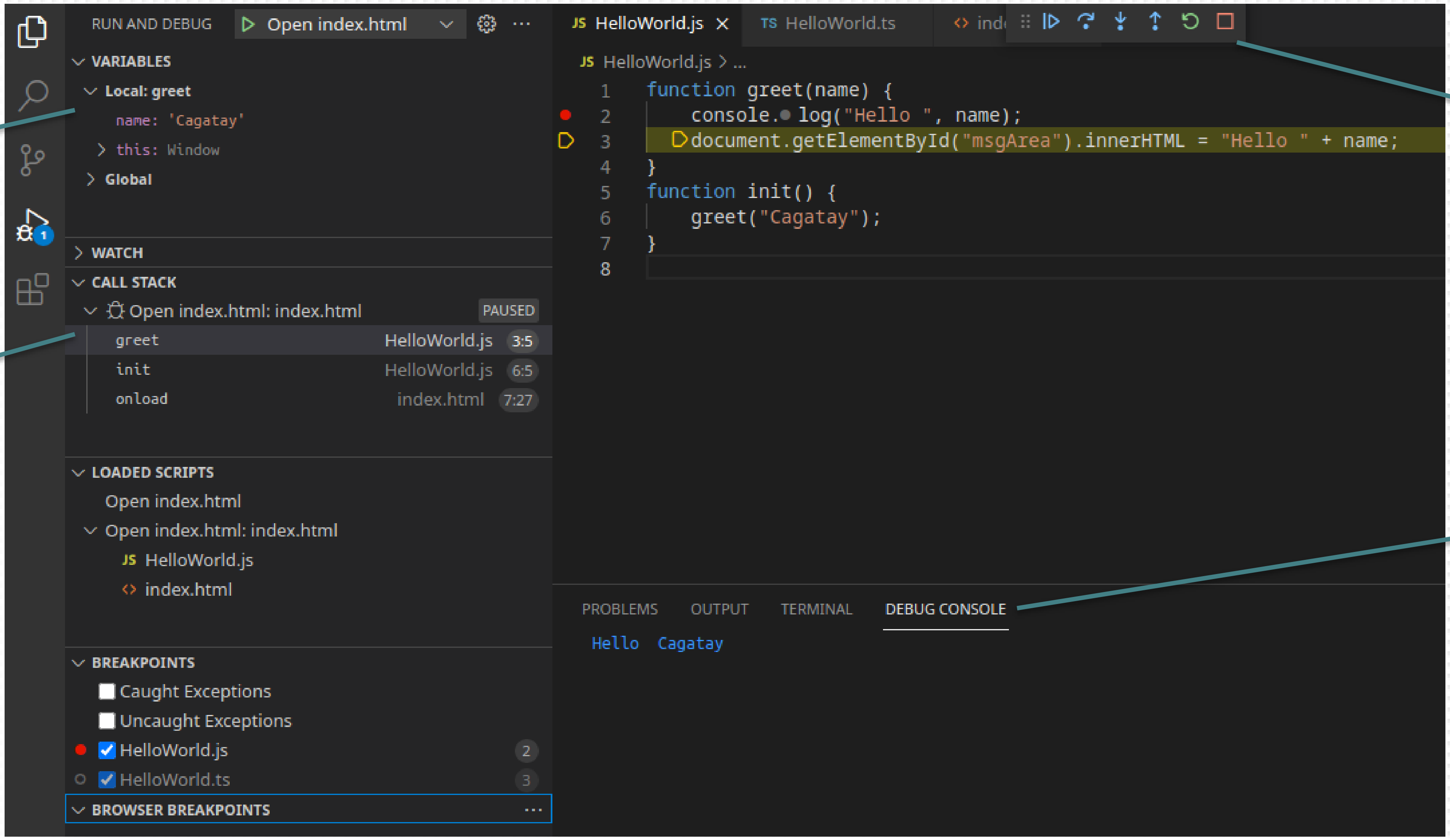
- Strong type checking helps you avoid common programming mistakes



```
let message: string = 15;
```

```
Type 'number' is not assignable to type 'string'. ts(2322)
let message: string
```

Why VS Code - Debugging



```
Local: greet
  name: 'Cagatay'
  > this: Window
> Global
```

```
Open index.html: index.html        PAUSED
  greet      HelloWorld.js   3:5
  init       HelloWorld.js   6:5
  onload     index.html      7:27
```

```
Open index.html
Open index.html: index.html
  HelloWorld.js
  index.html
```

```
Caught Exceptions
Uncaught Exceptions
HelloWorld.js   2
HelloWorld.ts   3
```

```javascript
function greet(name) {
    console.log("Hello ", name);
  document.getElementById("msgArea").innerHTML = "Hello " + name;
}
function init() {
    greet("Cagatay");
}
```

```
Hello  Cagatay
```

Variables

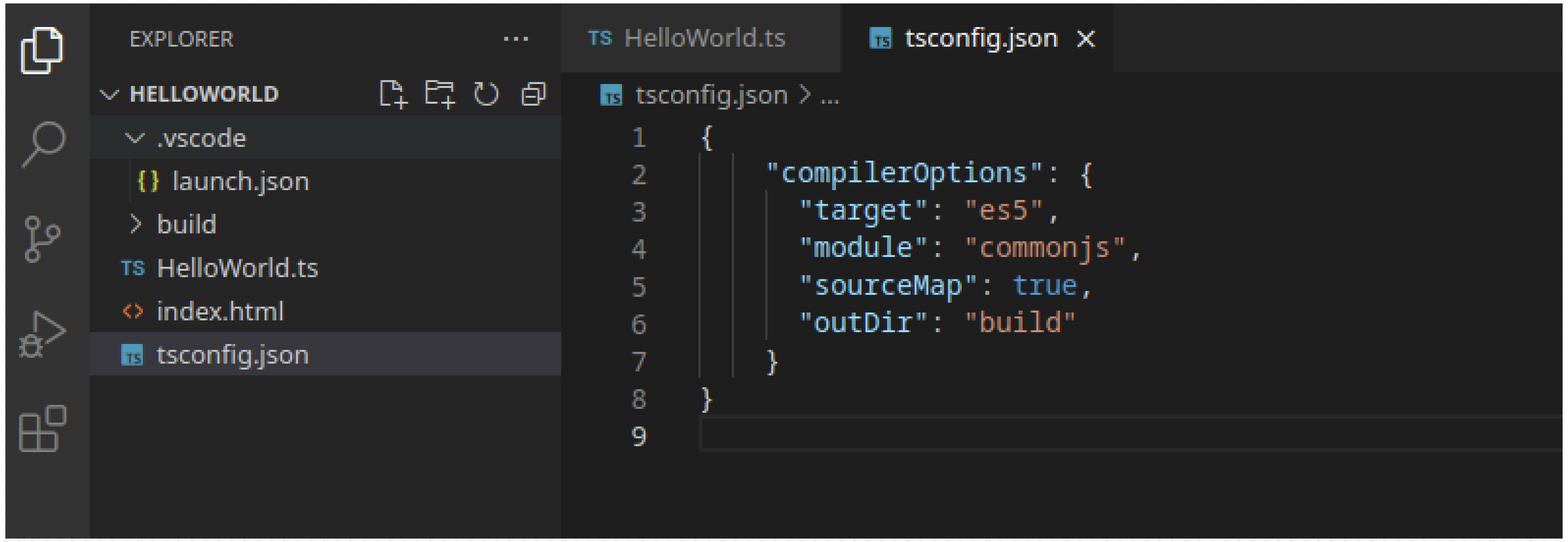
Call Stack

Debug Panel

Console

Why VS Code – Debugging with ts File

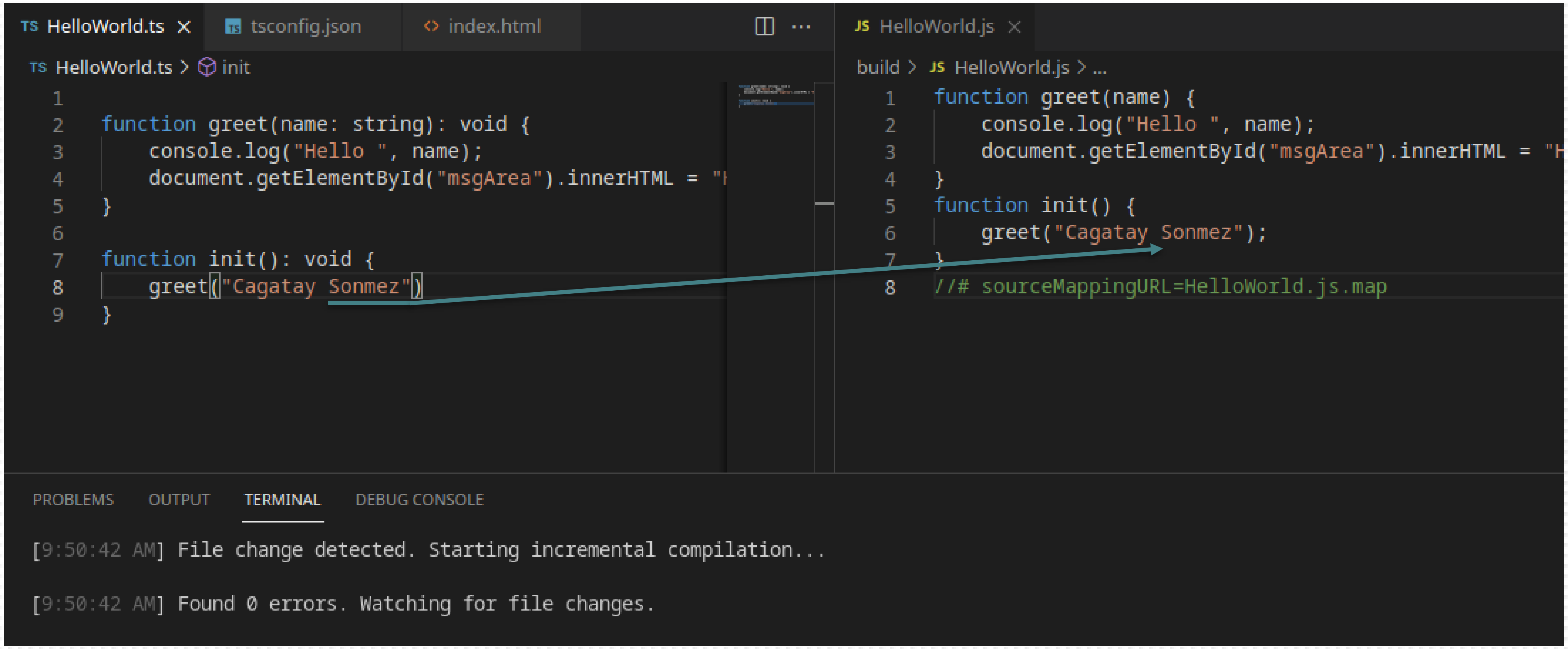
- VS Code relies on source maps for the debugger to map between the original TypeScript source code and the running JavaScript
- You can create source maps by setting "sourceMap": true in the tsconfig.json



```
{
    "compilerOptions": {
      "target": "es5",
      "module": "commonjs",
      "sourceMap": true,
      "outDir": "build"
    }
}
```

Why VS Code – Auto Generate JS File

- Your changes will be automatically compiled when you save a file if you run built task in watch mode using «ctrl + shift + B» shortcut



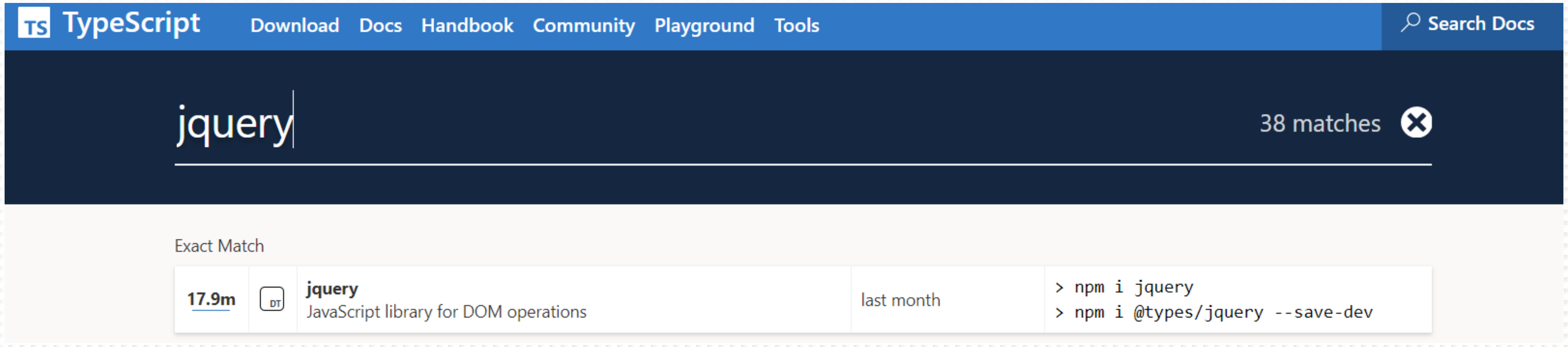
```typescript

function greet(name: string): void {
    console.log("Hello ", name);
    document.getElementById("msgArea").innerHTML = "H

}

function init(): void {
    greet("Cagatay Sonmez")
}
```

```javascript
function greet(name) {
    console.log("Hello ", name);
    document.getElementById("msgArea").innerHTML = "H
}
function init() {
    greet("Cagatay Sonmez");
}
//# sourceMappingURL=HelloWorld.js.map
```

```
[9:50:42 AM] File change detected. Starting incremental compilation...

[9:50:42 AM] Found 0 errors. Watching for file changes.
```

External (3rd Party) Libraries

- TypeScript cannot know type information of an existing javascript library
- A declaration file (with the extension .d.ts) is used to provide type information of an external API
- Declaration files are provided from a GitHub repository
  - https://github.com/DefinitelyTyped/DefinitelyTyped/
- You can also search for external libraries from offical TypeScript web page
  - https://www.typescriptlang.org/dt/search?search=



jquery

38 matches

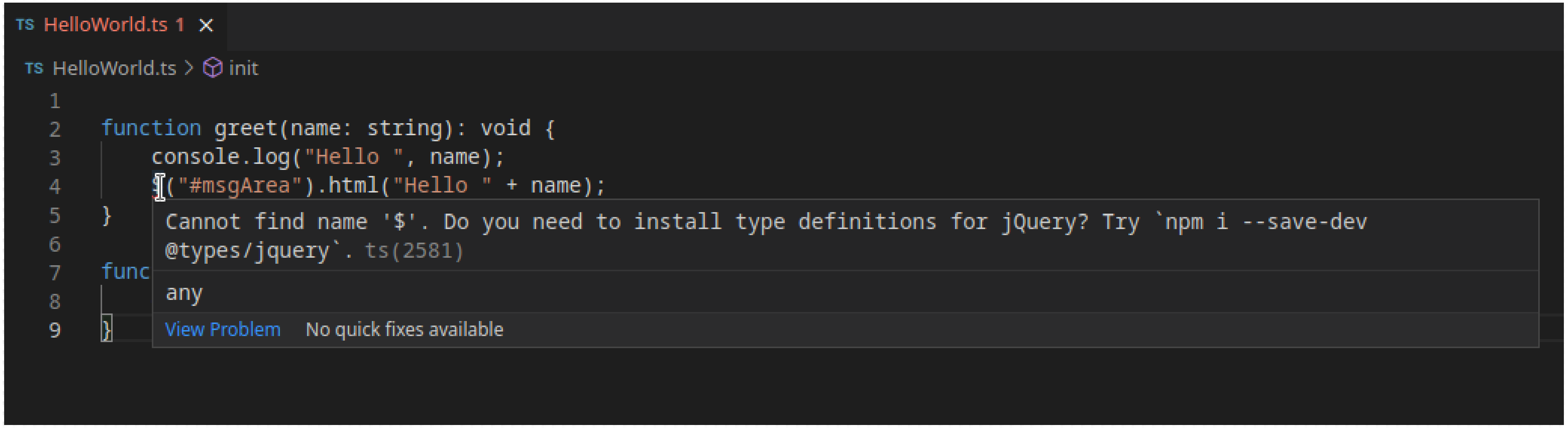
Exact Match

17.9m | jquery JavaScript library for DOM operations | last month

```
> npm i jquery
> npm i @types/jquery --save-dev
```

External Library Example I

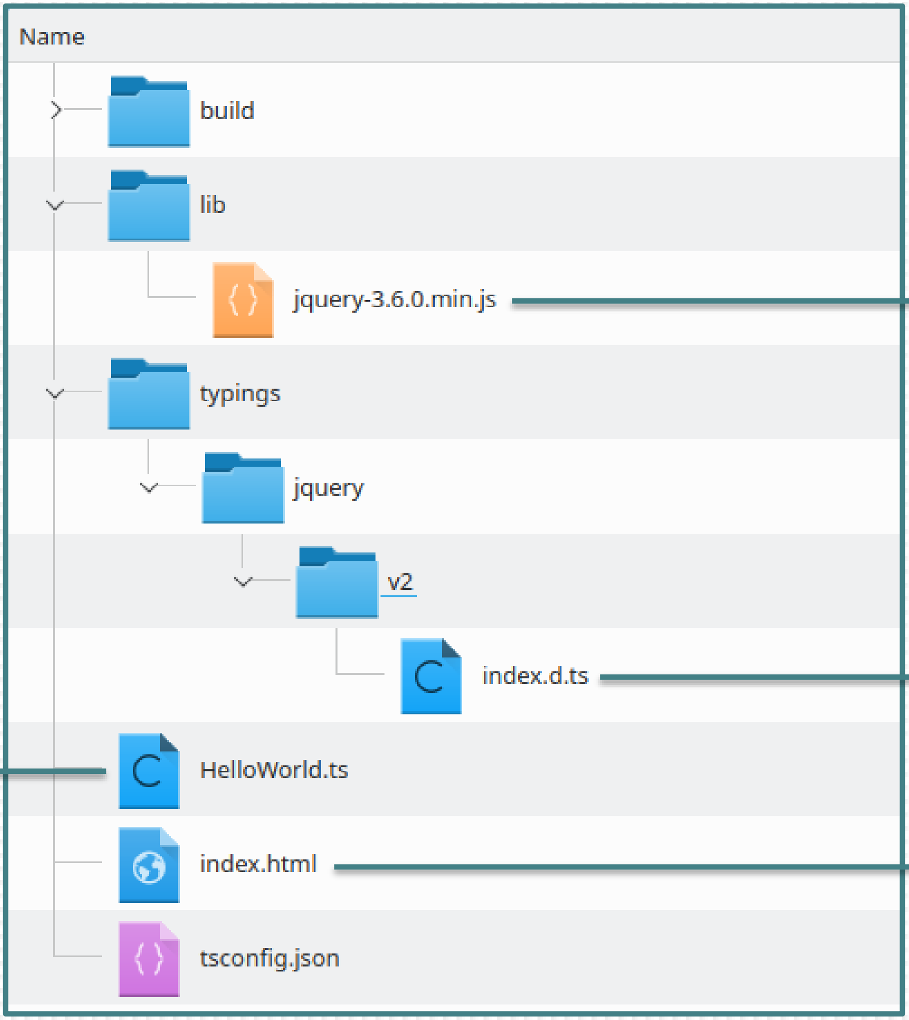
- You will get compile error if you don’t give d.ts file to VS Code
- Intellisense will not work either



```
function greet(name: string): void {
    console.log("Hello ", name);
    $("#msgArea").html("Hello " + name);
}

func

}
```

```
Cannot find name '$'. Do you need to install type definitions for jQuery? Try `npm i --save-dev
@types/jquery`. ts(2581)
any
```

External Library Example II



1- download the library

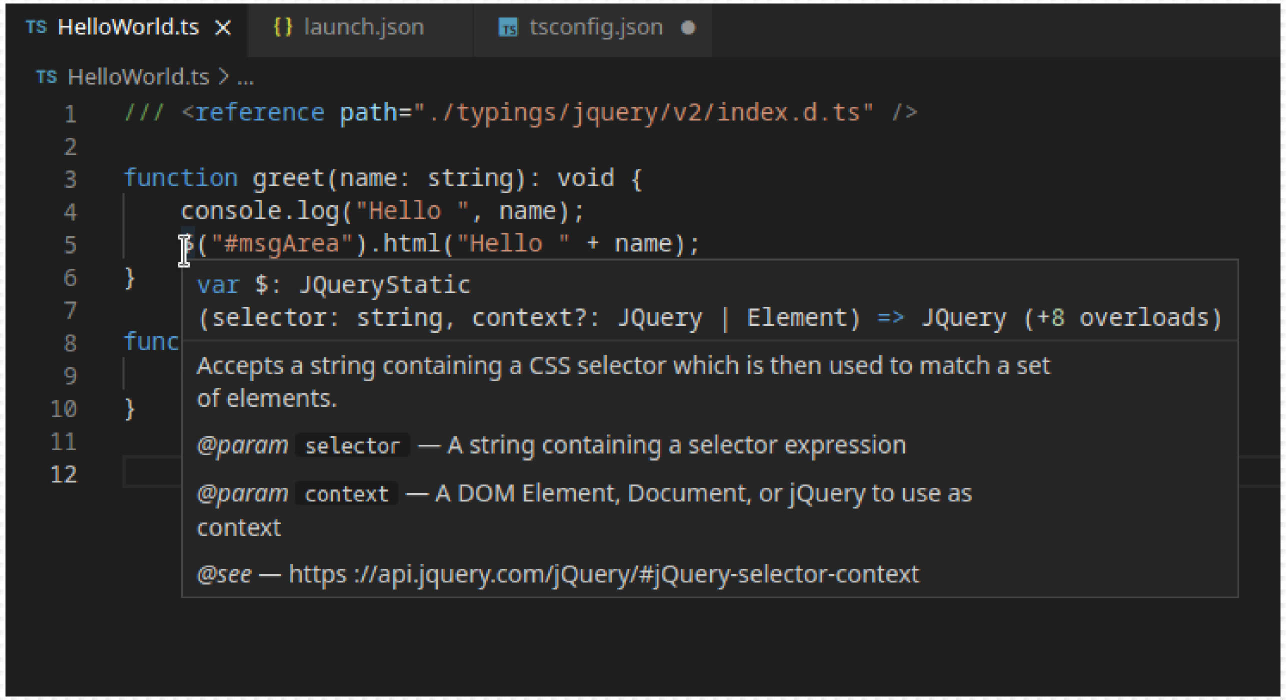
2- download the d.ts file

3- use reference path
to declare dependency

4- add js file of library to
<script> tag of html

Reference Path

- VS Code automatically finds .d.ts files, if you don’t have "files" section in your tsconfig.json file
- Otherwise, declare the dependency by using reference path which is a triple-slash directive



```
/// <reference path="./typings/jquery/v2/index.d.ts" />

function greet(name: string): void {
    console.log("Hello ", name);
    $("#msgArea").html("Hello " + name);
}

func

}

```

```
var $: JQueryStatic
(selector: string, context?: JQuery | Element) => JQuery (+8 overloads)
```

Accepts a string containing a CSS selector which is then used to match a set of elements.

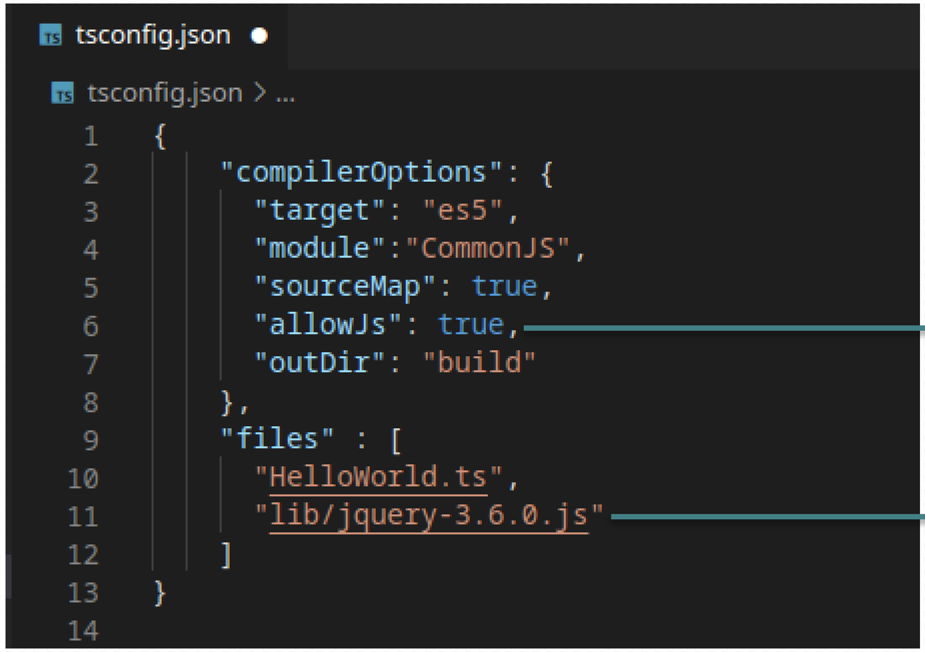
@param `selector` — A string containing a selector expression

@param `context` — A DOM Element, Document, or jQuery to use as context

@see — https ://api.jquery.com/jQuery/#jQuery-selector-context

Including JavaScript files

- You can also use existing (maybe 3rd party) JavaScript file in you project
- Enable "allowJs" options in compiler options



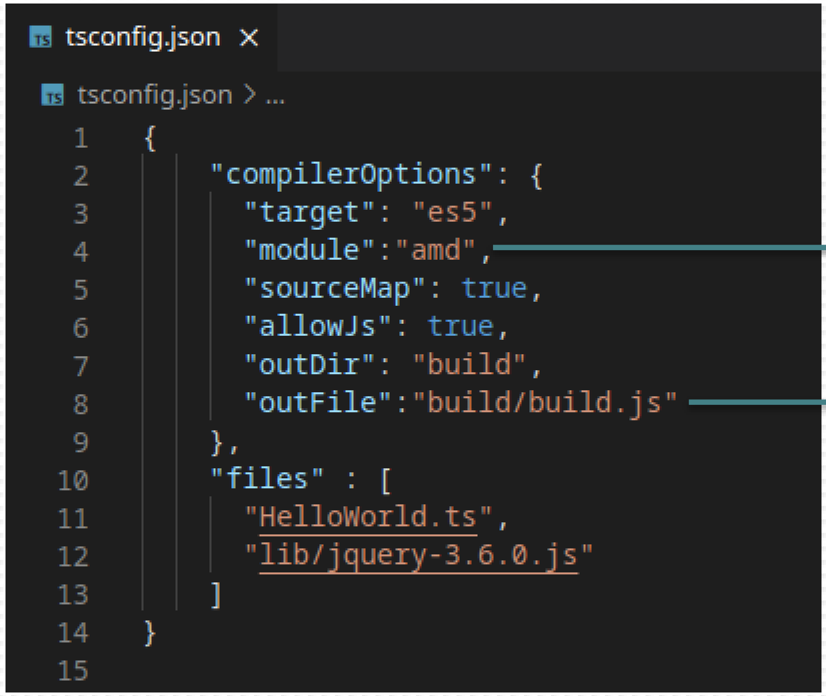
```json
{
    "compilerOptions": {
      "target": "es5",
      "module":"CommonJS",
      "sourceMap": true,
      "allowJs": true,
      "outDir": "build"
    },
    "files" : [
      "HelloWorld.ts",
      "lib/jquery-3.6.0.js"
    ]
}
```

1- allow using JS

2- add JS file

Compile to Single Output File

- We can produce a single JS file for all TS and JS files
- To make this use the "outFile" parameter
- Please note that outFile parameters uses either *amd* or *system* "module" options



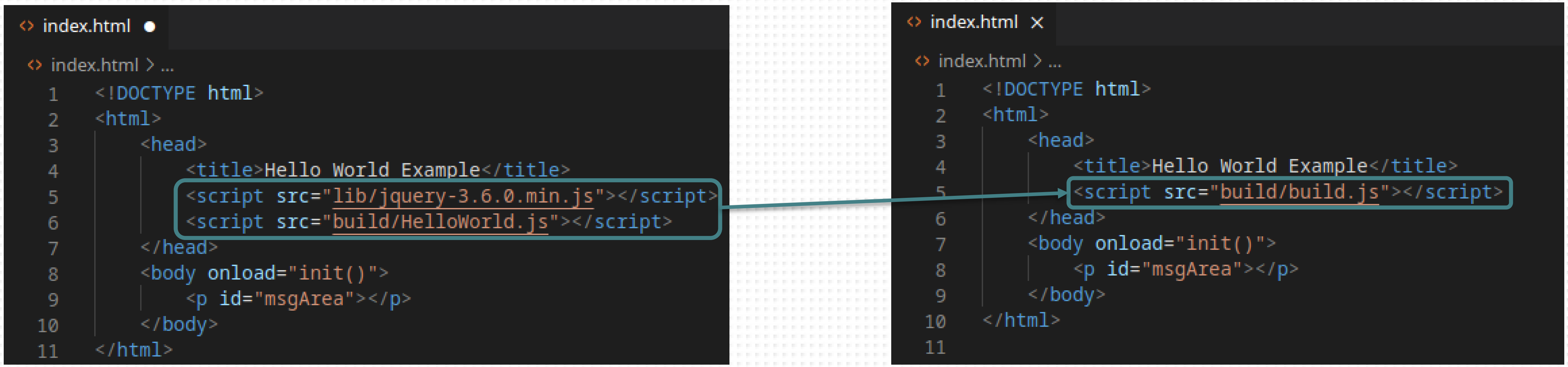
```
{
    "compilerOptions": {
      "target": "es5",
      "module":"amd",
      "sourceMap": true,
      "allowJs": true,
      "outDir": "build",
      "outFile":"build/build.js"
    },
    "files" : [
      "HelloWorld.ts",
      "lib/jquery-3.6.0.js"
    ]
}
```

1- use proper module

2- define an outFile

Why Compile to Single Output File?

- It is better to manage dependencies with build system



index.html ●

index.html > ...

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World Example</title>
        <script src="lib/jquery-3.6.0.min.js"></script>
        <script src="build/HelloWorld.js"></script>
    </head>
    <body onload="init()">
        <p id="msgArea"></p>
    </body>
</html>
```

index.html ×

index.html > ...

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World Example</title>
        <script src="build/build.js"></script>
    </head>
    <body onload="init()">
        <p id="msgArea"></p>
    </body>
</html>
```

QUESTIONS?