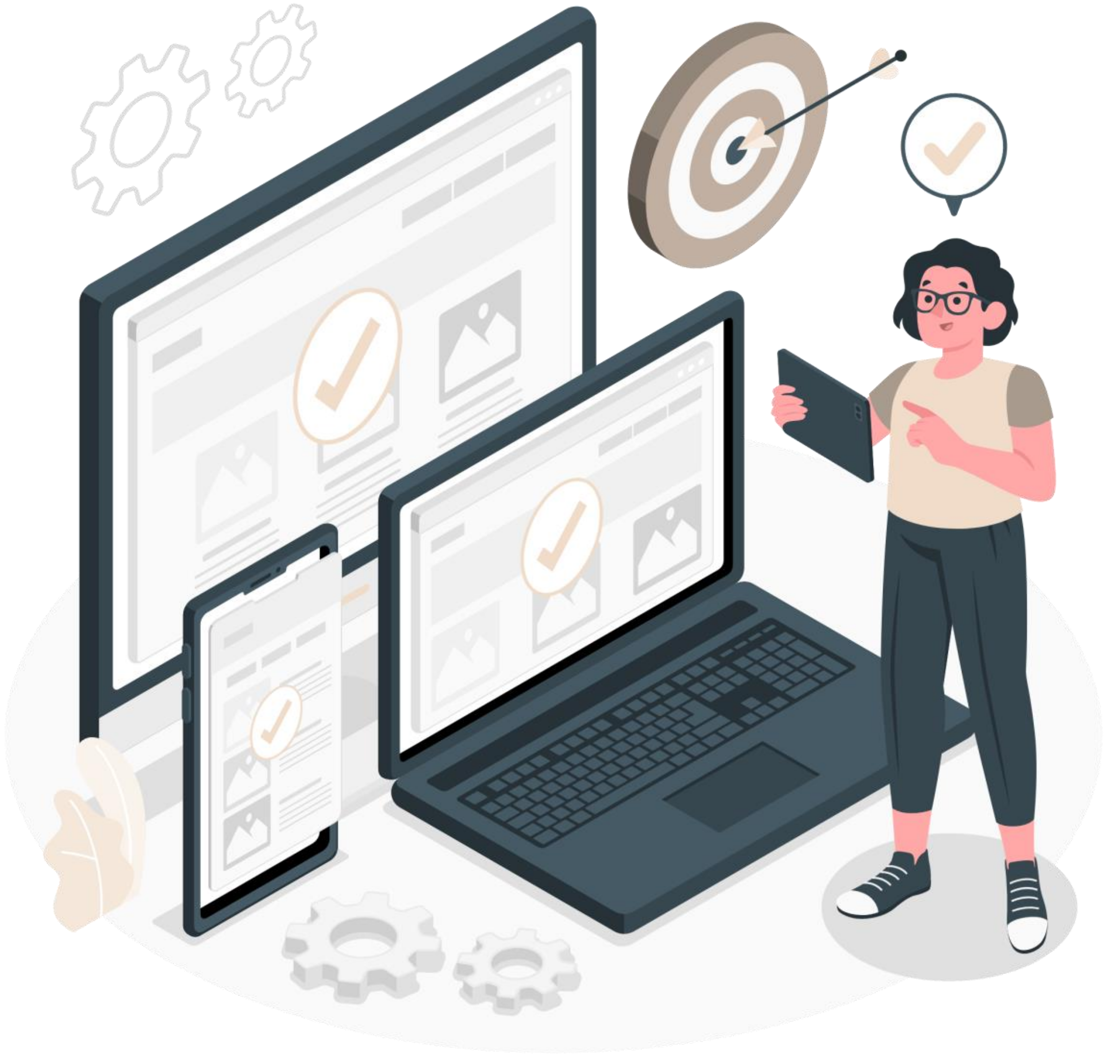
Intro to TypeScript

IT 331: APPLICATION DEVELOPMENT AND EMERGING TECHNOLOGIES



In this Presentation

TypeScript in a Nutshell

A JAVASCRIPT WITH STATIC TYPING

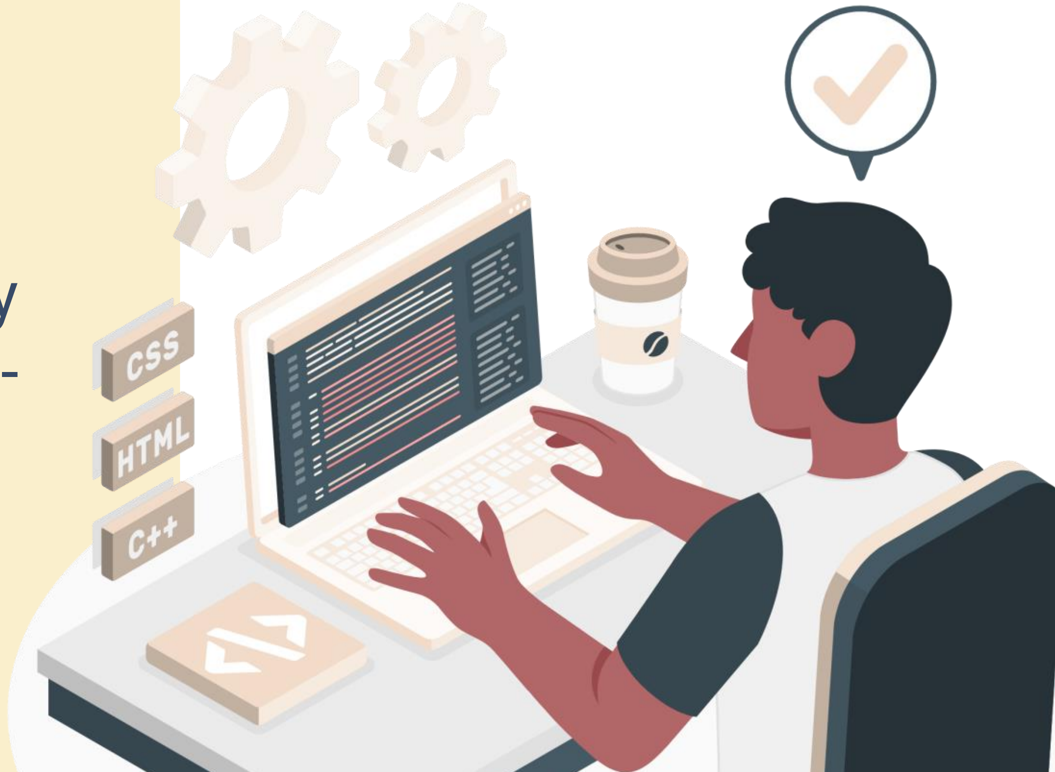
TypeScript adds syntax on top of JavaScript allowing developers to add types.

TYPESCRIPT USES COMPILE TIME CHECKING

TypeScript uses compile time checking. Which means it checks ***if the specified types match before running the code***, not while running the program.

What is TypeScript

- Designed for large-scale applications.
- Can be compiled to plain JavaScript for any browser.
- Developed and maintained by Microsoft and under open-source license.



Why TypeScript?

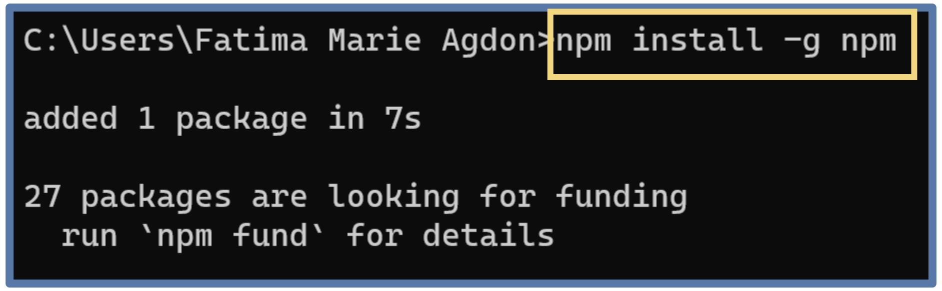
- Has **static typing**, which is the most important feature.
- Developed for large-scale applications.
- Prevents common JavaScript mistake.
- Learning TS is not learning a new language, it was built on top of JavaScript. **JS code is valid in TS.**

Installing TypeScript

What you will need

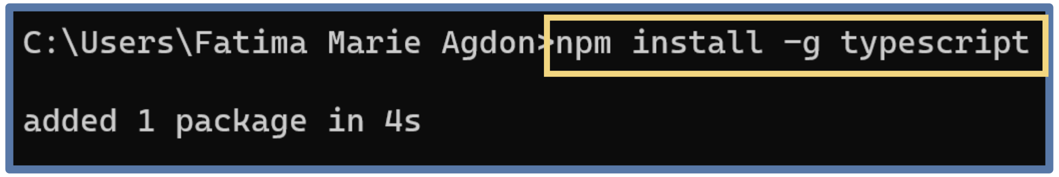
- Node JS (LTS Version)
- Node Package Manager (npm)

Install npm via cmd:



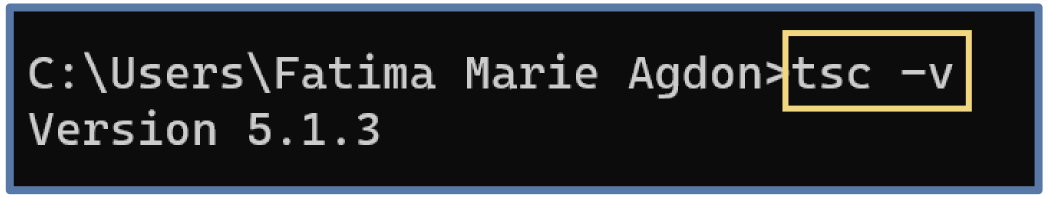
```
C:\Users\Fatima Marie Agdon>npm install -g npm

added 1 package in 7s

27 packages are looking for funding
  run `npm fund` for details
```

How to install?



```
C:\Users\Fatima Marie Agdon>npm install -g typescript

added 1 package in 4s
```

Verify installation using:



```
C:\Users\Fatima Marie Agdon>tsc -v
Version 5.1.3
```

Returns the current version of TypeScript installed.

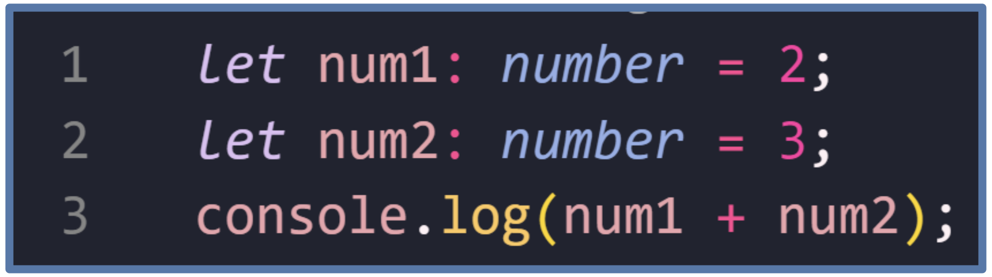
First TypeScript Program

What you will need

Any IDE

Creating your TypeScript file

- Create a file with name *Test.ts*
- Write the following code:



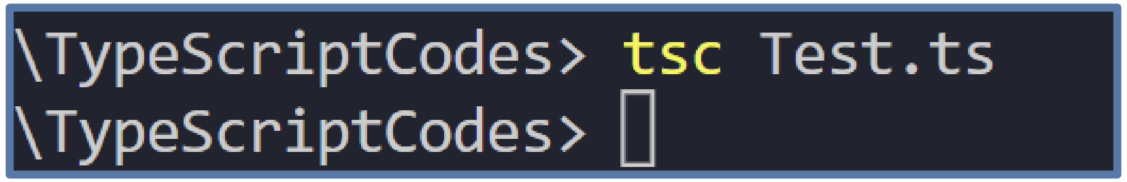
```
let num1: number = 2;
let num2: number = 3;
console.log(num1 + num2);
```

- Do not forget to save your file.

First TypeScript Program

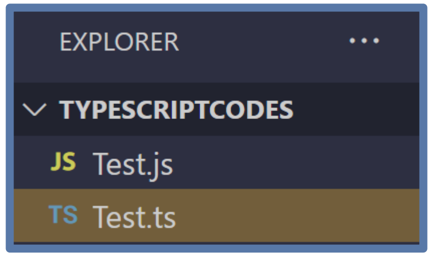
Compiling the Program

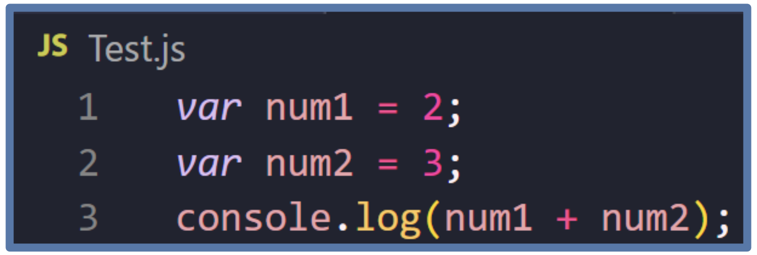
- Before you run your program, compile it first using the command tsc <filename.ts>.



```
\TypeScriptCodes> tsc Test.ts
\TypeScriptCodes> 
```

- In our case, it will be tsc Test.ts this will create an new .js file.

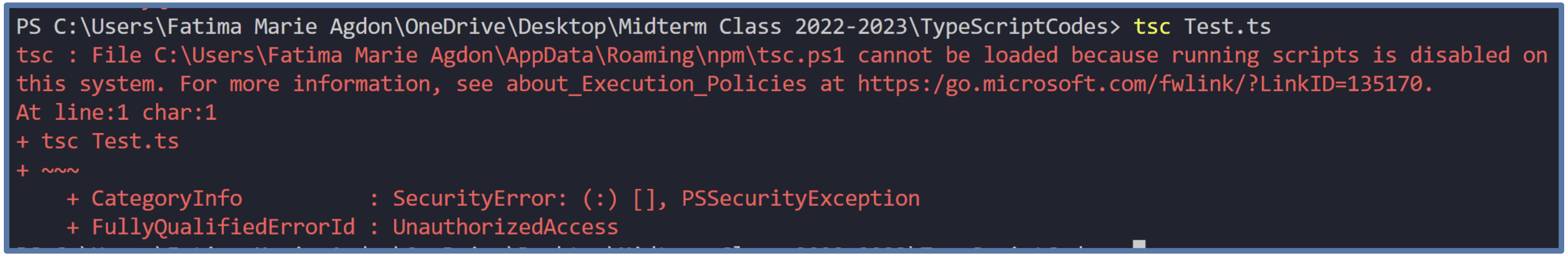




Test.js

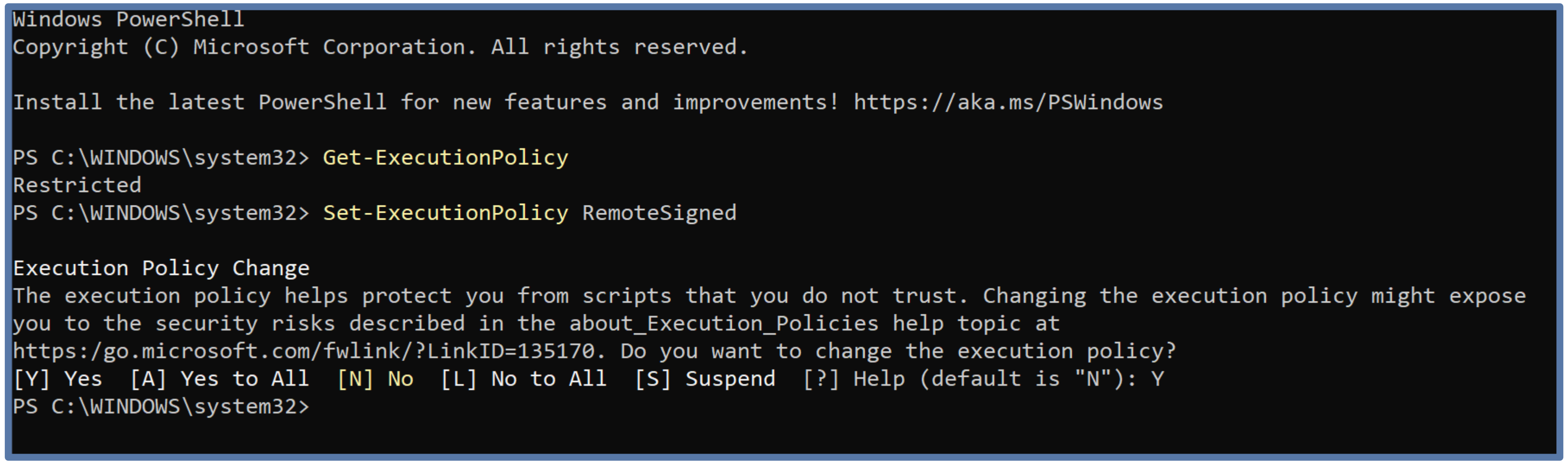
```
var num1 = 2;
var num2 = 3;
console.log(num1 + num2);
```

- **In case this is encountered during compilation:**



```
PS C:\Users\Fatima Marie Agdon\OneDrive\Desktop\Midterm Class 2022-2023\TypeScriptCodes> tsc Test.ts
tsc : File C:\Users\Fatima Marie Agdon\AppData\Roaming\npm\tsc.ps1 cannot be loaded because running scripts is disabled on
this system. For more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ tsc Test.ts
+ ~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

- Search for **PowerShell** on your device, and select Run as Administrator, follow as indicated in the screenshot:

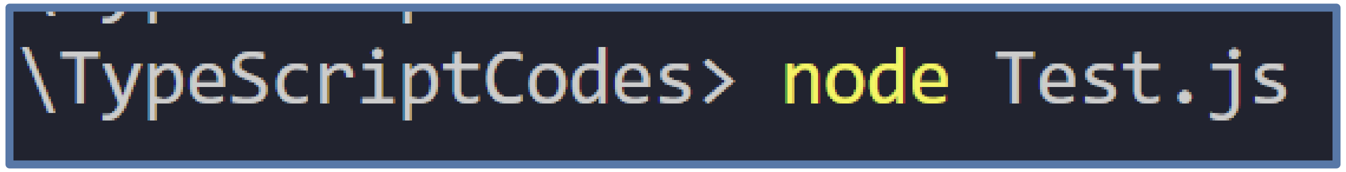


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> Get-ExecutionPolicy
Restricted
PS C:\WINDOWS\system32> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): Y
PS C:\WINDOWS\system32>
```

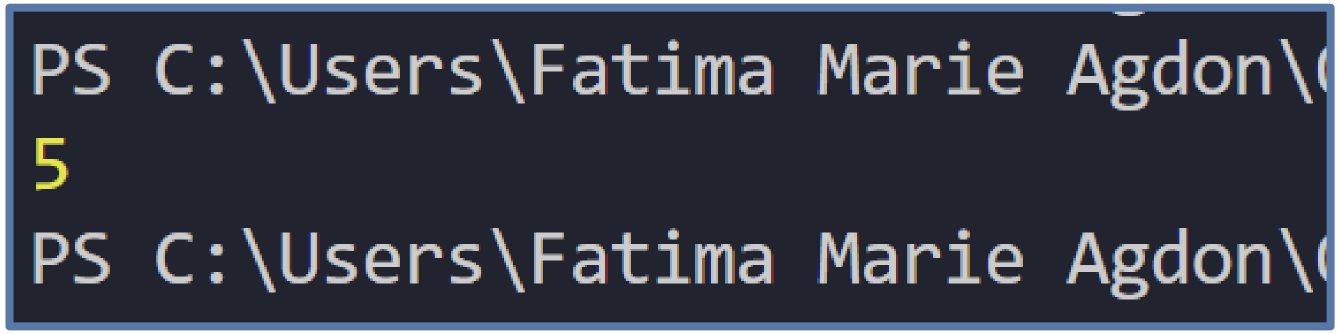
First TypeScript Program

Execution

- Run your program with NodeJS using the command:



```
\TypeScriptCodes> node Test.js
```



```
PS C:\Users\Fatima Marie Agdon\
5
PS C:\Users\Fatima Marie Agdon\
```

TypeScript Fundamentals

1. Simple and Special Types
2. Arrays and Tuples
3. Enums
4. Functions
5. Object Types
6. Classes

1

SIMPLE TYPES

There are three main primitive types in JavaScript and in TypeScript.

BOOLEAN

Holds the value *True* or *False*

NUMBER

It can be a whole number or floating point types.

STRING

Consists of one or more characters enclosed in double quotes. (“ and ”).

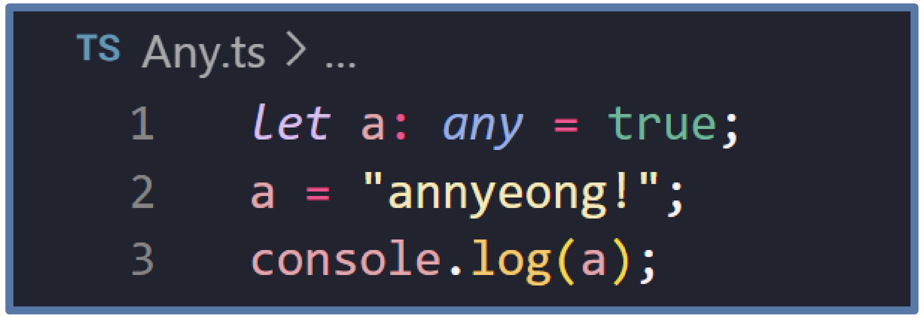
1

SPECIAL TYPES

TypeScript has special types that does not refer to any specific data type.

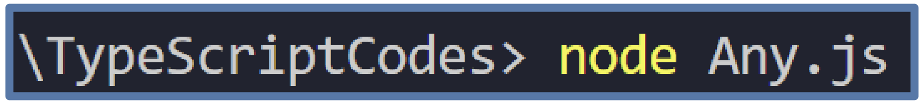
ANY

Disables all the type checking and allows all types to be used.

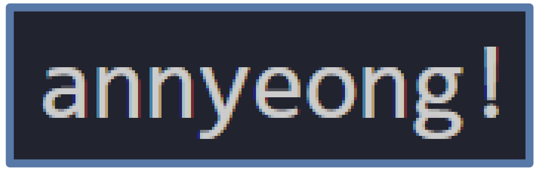


TS Any.ts > ...

```typescript
let a: any = true;
a = "annyeong!";
console.log(a);
```



```
\TypeScriptCodes> node Any.js
```



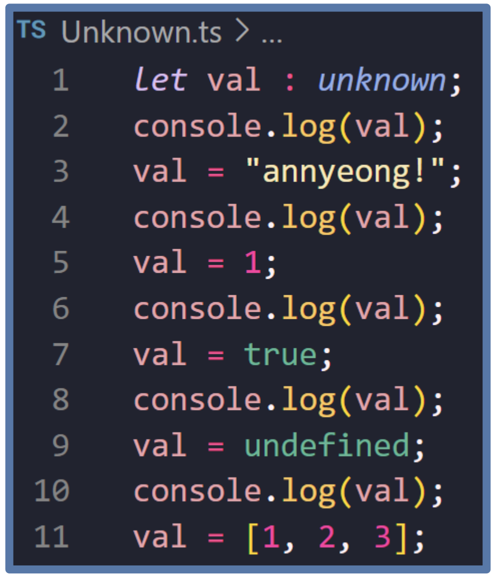
```
annyeong!
```

1

SPECIAL TYPES

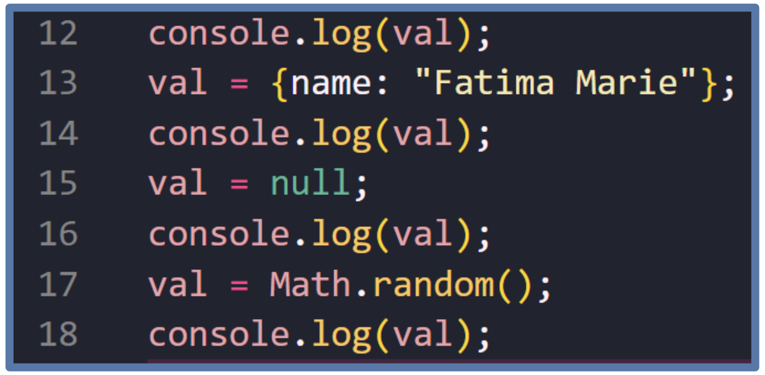
UNKNOWN

Similar to any but safer, it prevents unknown types to be used.

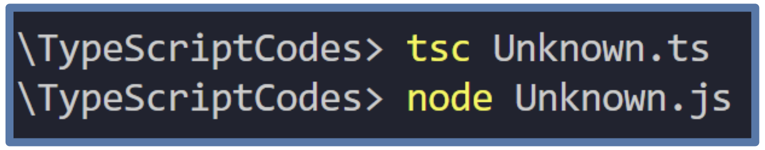


TS Unknown.ts > ...

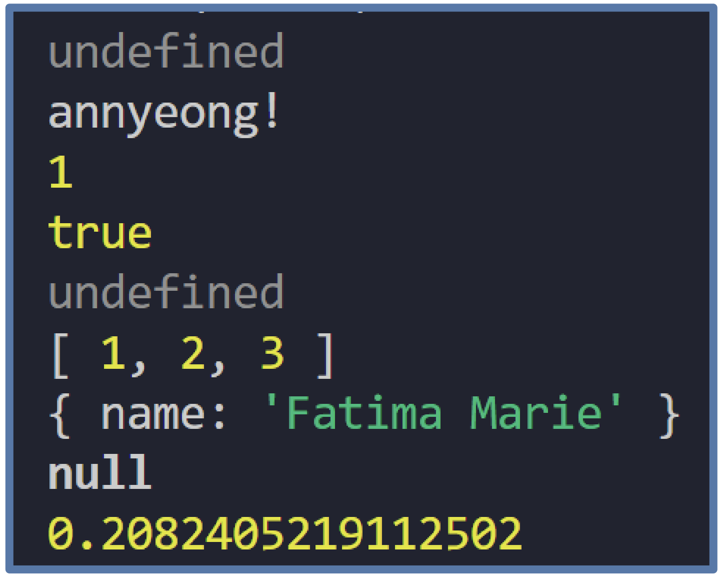
```typescript
let val : unknown;
console.log(val);
val = "annyeong!";
console.log(val);
val = 1;
console.log(val);
val = true;
console.log(val);
val = undefined;
console.log(val);
val = [1, 2, 3];
```



```typescript
console.log(val);
val = {name: "Fatima Marie"};
console.log(val);
val = null;
console.log(val);
val = Math.random();
console.log(val);
```



```
\TypeScriptCodes> tsc Unknown.ts
\TypeScriptCodes> node Unknown.js
```



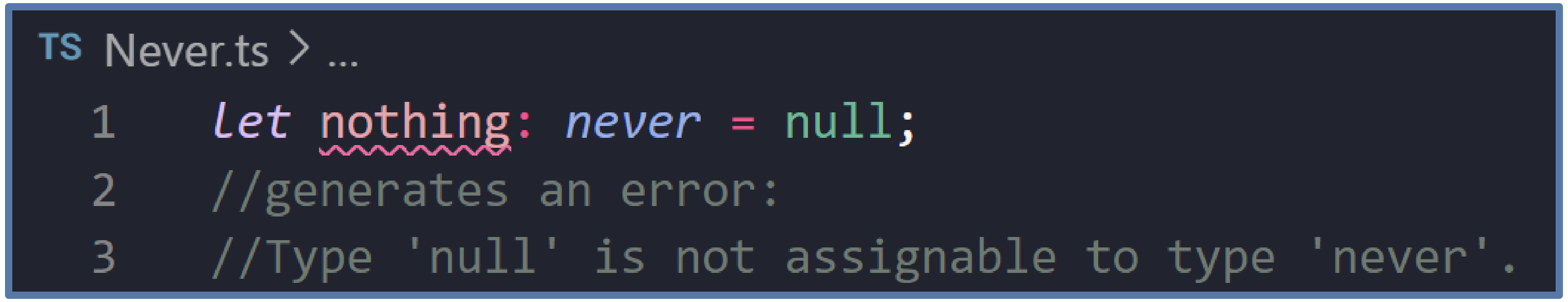
```
undefined
annyeong!
1
true
undefined
[ 1, 2, 3 ]
{ name: 'Fatima Marie' }
null
0.2082405219112502
```

1

SPECIAL TYPES

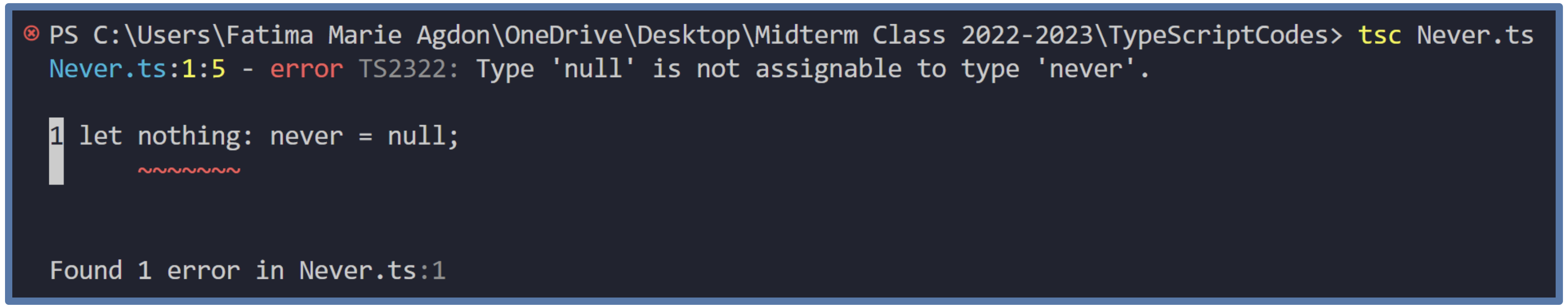
NEVER

Effectively throws an error whenever it is defined.



TS Never.ts > ...

```
let nothing: never = null;
//generates an error:
//Type 'null' is not assignable to type 'never'.
```



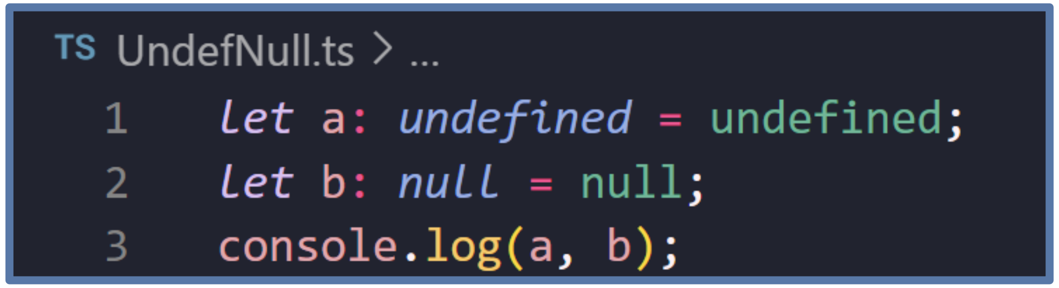
```
PS C:\Users\Fatima Marie Agdon\OneDrive\Desktop\Midterm Class 2022-2023\TypeScriptCodes> tsc Never.ts
Never.ts:1:5 - error TS2322: Type 'null' is not assignable to type 'never'.

1 let nothing: never = null;
      ~~~~~~~


Found 1 error in Never.ts:1
```

1

SPECIAL TYPES

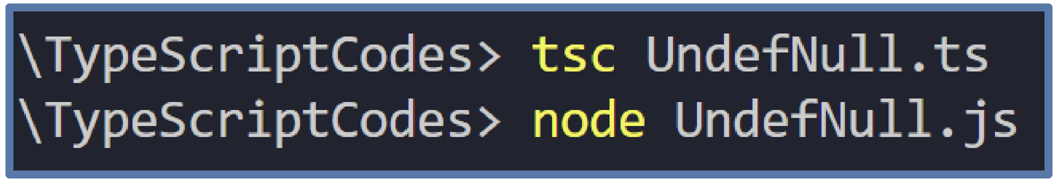
UNDEFINED AND NULL

The JavaScript primitive types undefined and null respectively

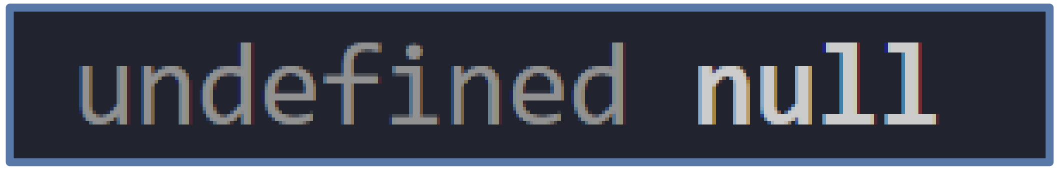


TS UndefNull.ts > ...

```typescript
let a: undefined = undefined;
let b: null = null;
console.log(a, b);
```



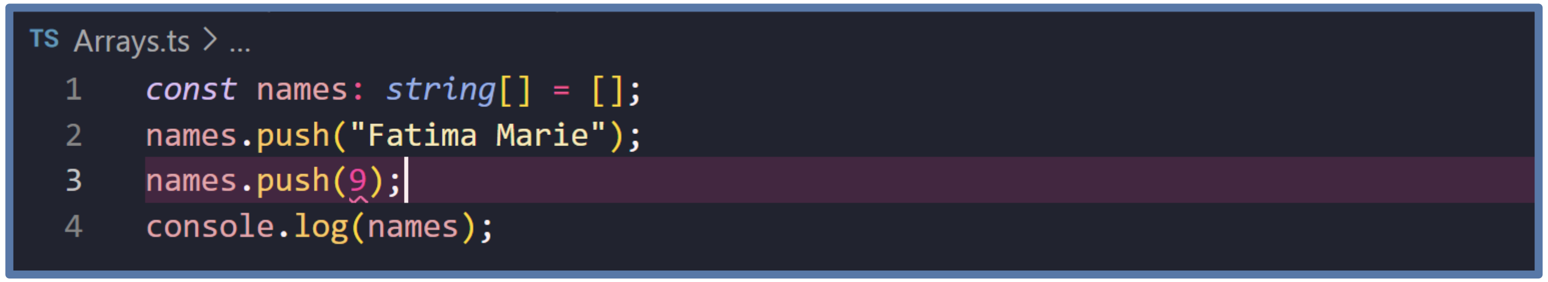
```
\TypeScriptCodes> tsc UndefNull.ts
\TypeScriptCodes> node UndefNull.js
```



```
undefined null
```

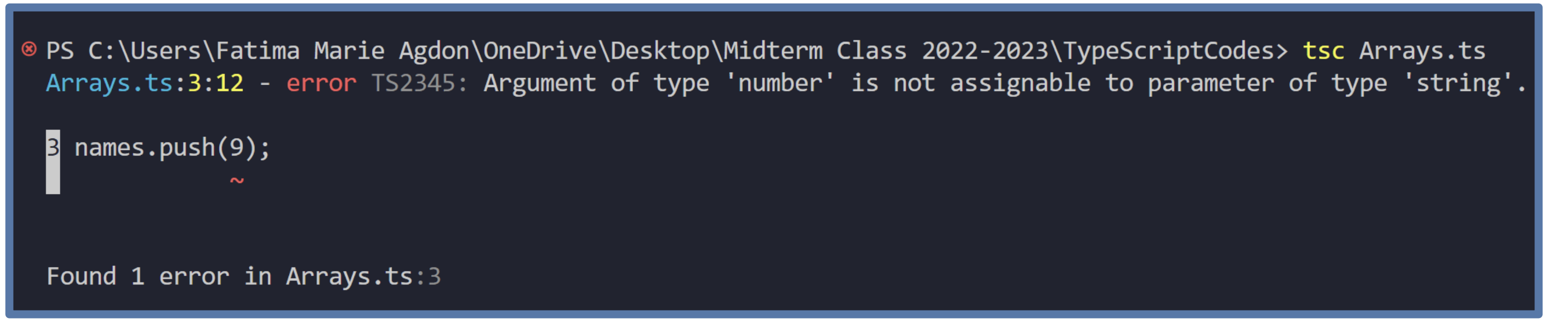
2 ARRAYS

TypeScript has as specific syntax for array types. It includes the modifiers *readonly* modifier and *inference*.



Arrays.ts > ...

```
const names: string[] = [];
names.push("Fatima Marie");
names.push(9);
console.log(names);
```



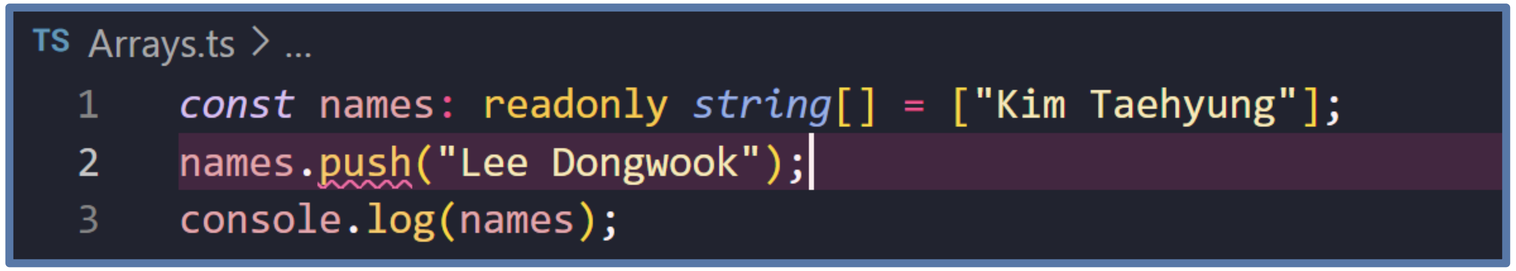
```
PS C:\Users\Fatima Marie Agdon\OneDrive\Desktop\Midterm Class 2022-2023\TypeScriptCodes> tsc Arrays.ts
Arrays.ts:3:12 - error TS2345: Argument of type 'number' is not assignable to parameter of type 'string'.

3 names.push(9);
             ~


Found 1 error in Arrays.ts:3
```

2

ARRAYS

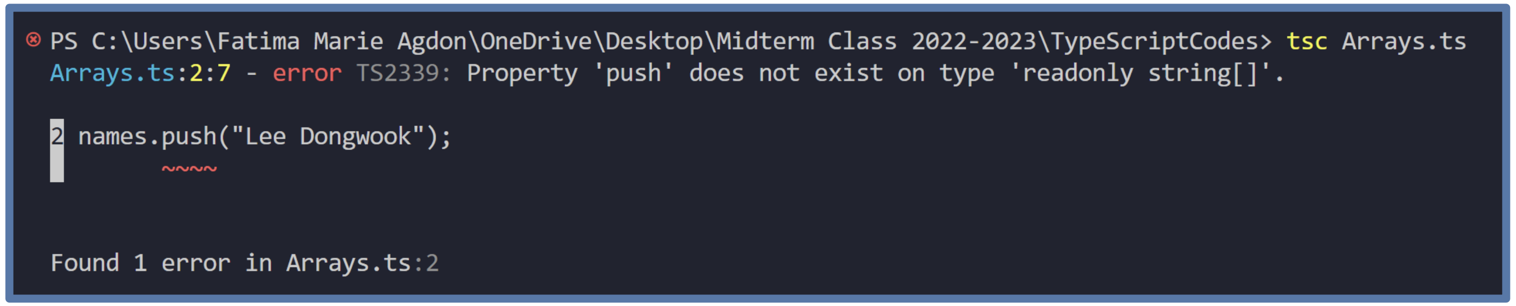
READONLY

A modifier prevents changes in your array.



TS Arrays.ts > ...

```
const names: readonly string[] = ["Kim Taehyung"];
names.push("Lee Dongwook");
console.log(names);
```



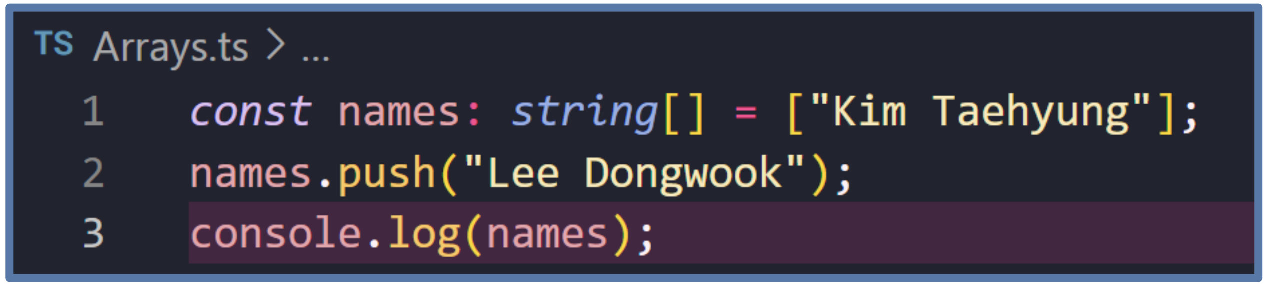
```
⊗ PS C:\Users\Fatima Marie Agdon\OneDrive\Desktop\Midterm Class 2022-2023\TypeScriptCodes> tsc Arrays.ts
  Arrays.ts:2:7 - error TS2339: Property 'push' does not exist on type 'readonly string[]'.

  2 names.push("Lee Dongwook");
          ~~~~


  Found 1 error in Arrays.ts:2
```

You have to remove readonly modifier in order to make the changes in your array.

2 ARRAYS

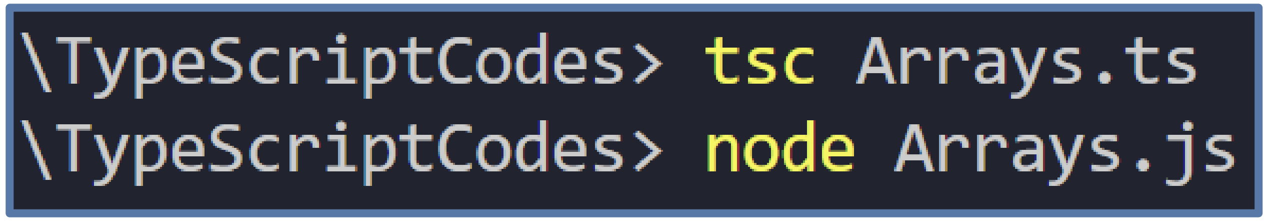
READONLY

After removing readonly:

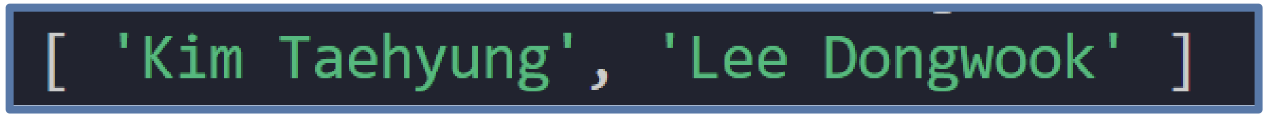


Arrays.ts > ...

```
const names: string[] = ["Kim Taehyung"];
names.push("Lee Dongwook");
console.log(names);
```



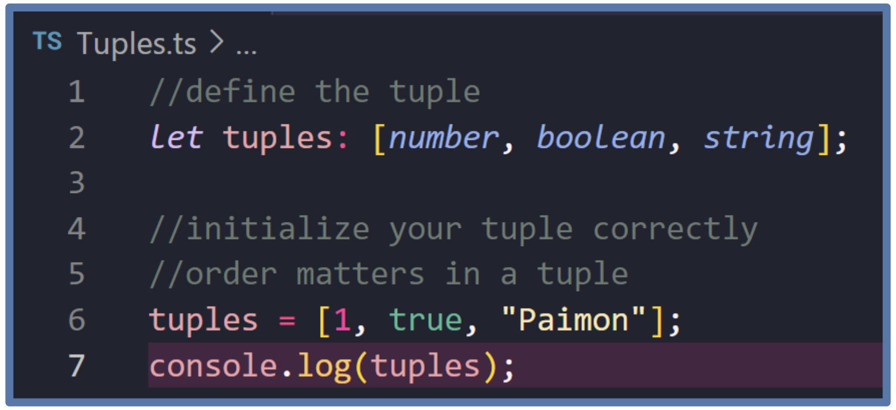
```
\TypeScriptCodes> tsc Arrays.ts
\TypeScriptCodes> node Arrays.js
```



```
[ 'Kim Taehyung', 'Lee Dongwook' ]
```

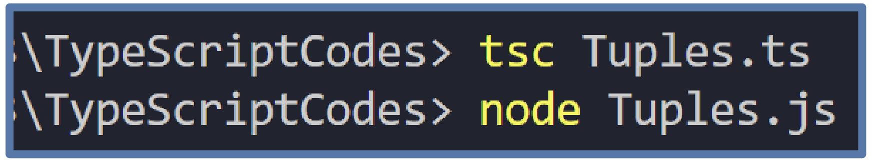
2 TUPLES

- A type array with a ***pre-defined length and type*** for each index.
- It allows each element in the array to be a known type of value.
- To ***define a tuple*** you need to ***define each type in the array***.

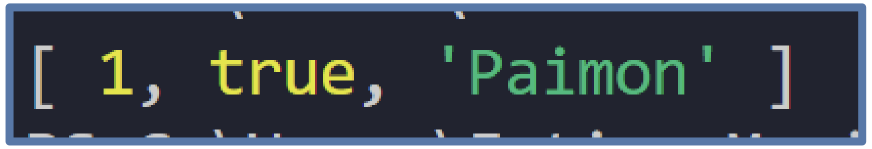


TS Tuples.ts > ...

```
//define the tuple
let tuples: [number, boolean, string];

//initialize your tuple correctly
//order matters in a tuple
tuples = [1, true, "Paimon"];
console.log(tuples);
```



```
\TypeScriptCodes> tsc Tuples.ts
\TypeScriptCodes> node Tuples.js
```



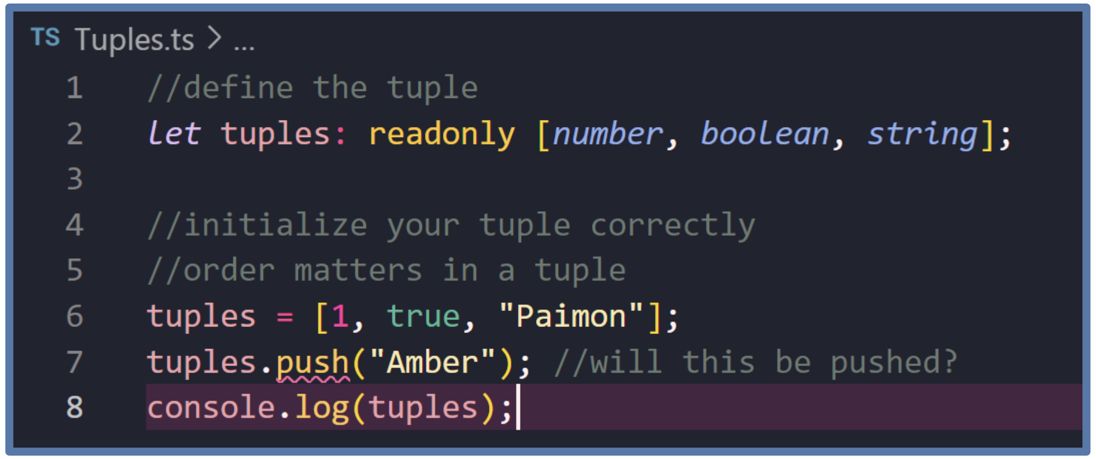
```
[ 1, true, 'Paimon' ]
```

2

TUPLES

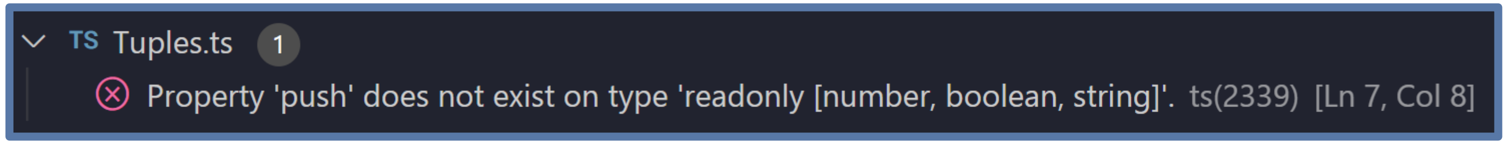
READONLY

Tuples only have strongly defined to initialized values, so it is recommended to make your tuple *readonly*.



TS Tuples.ts > ...

```typescript
//define the tuple
let tuples: readonly [number, boolean, string];

//initialize your tuple correctly
//order matters in a tuple
tuples = [1, true, "Paimon"];
tuples.push("Amber"); //will this be pushed?
console.log(tuples);
```



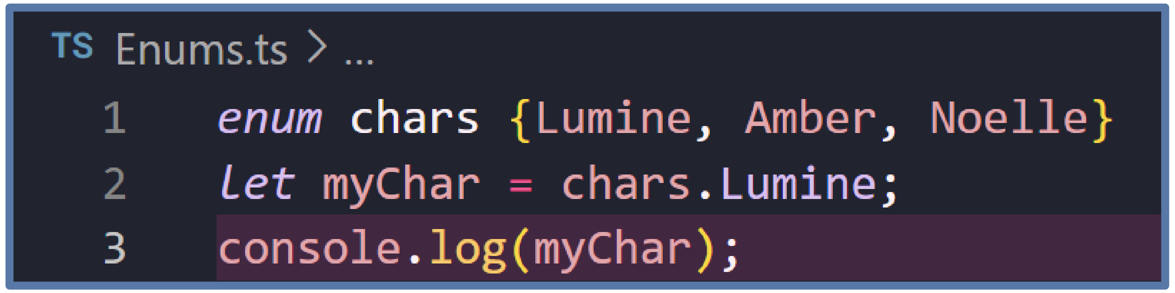
TS Tuples.ts 1

```
Property 'push' does not exist on type 'readonly [number, boolean, string]'. ts(2339) [Ln 7, Col 8]
```

3

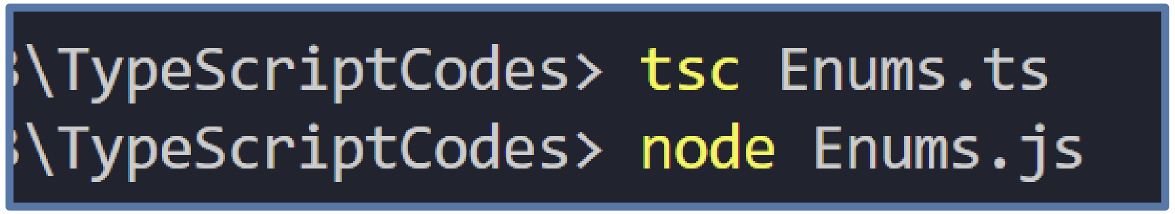
ENUMS

- A special class that represents a group of constants and it comes in two flavors: number and string.
- By default, enum will initialize the first value to 0 then add 1 to each additional value.

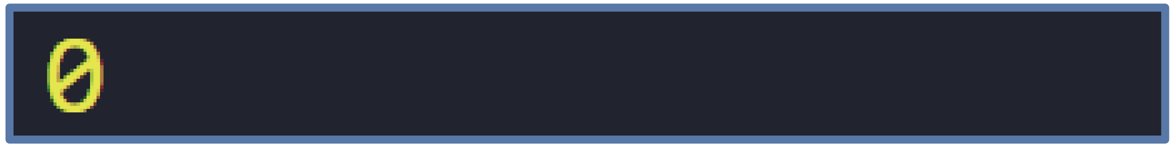


TS Enums.ts > ...

```typescript
enum chars {Lumine, Amber, Noelle}
let myChar = chars.Lumine;
console.log(myChar);
```



```
\TypeScriptCodes> tsc Enums.ts
\TypeScriptCodes> node Enums.js
```



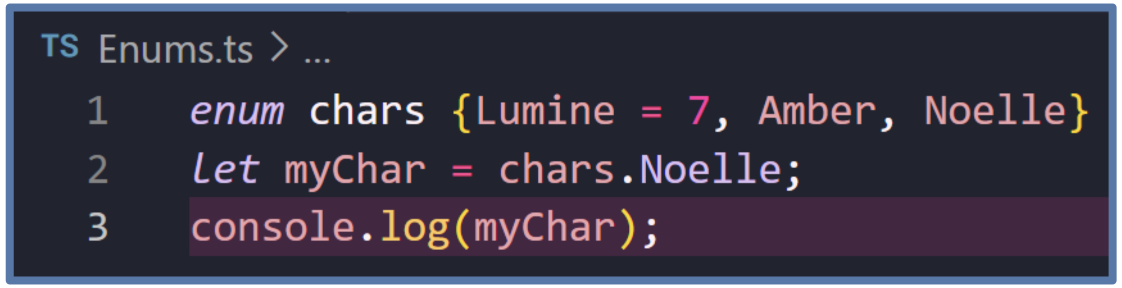
```
0
```

3

ENUMS

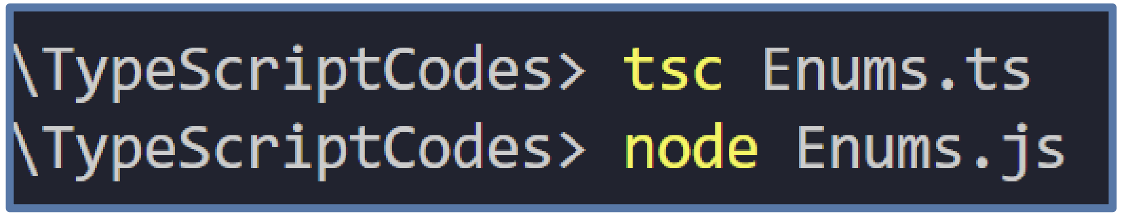
INITIALIZED

You can set the value of your first numeric enum and it will increment automatically.

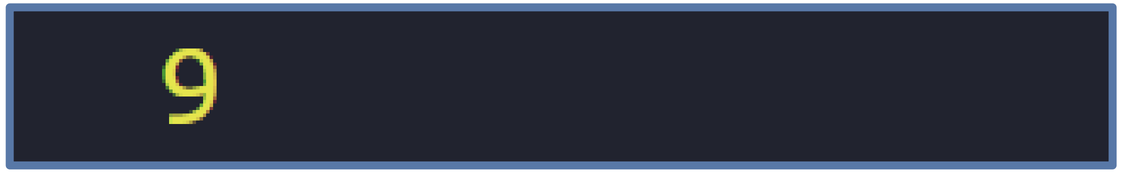


TS Enums.ts > ...

```
enum chars {Lumine = 7, Amber, Noelle}
let myChar = chars.Noelle;
console.log(myChar);
```



```
\TypeScriptCodes> tsc Enums.ts
\TypeScriptCodes> node Enums.js
```



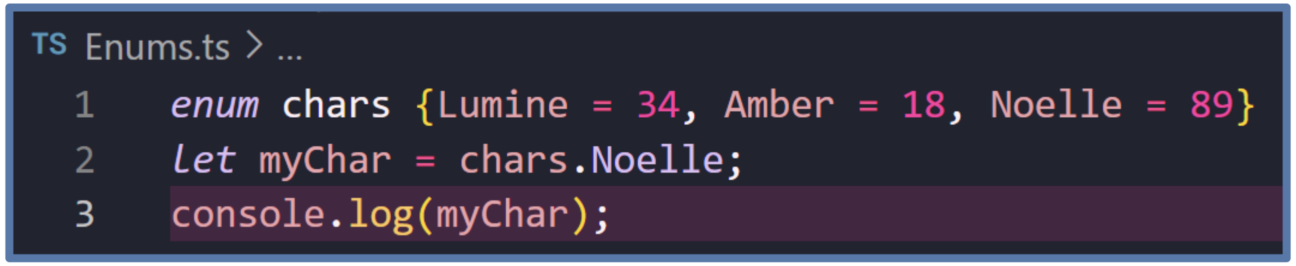
```
9
```

3

ENUMS

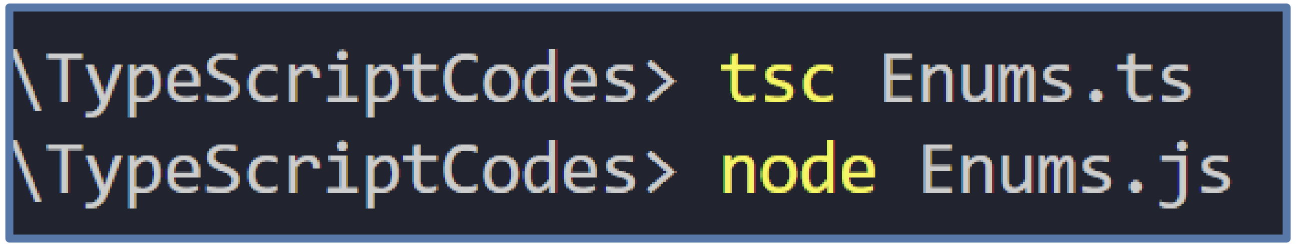
FULLY INITIALIZED

You can explicitly assign number value in each enum value, take note that **these values will not increment**.

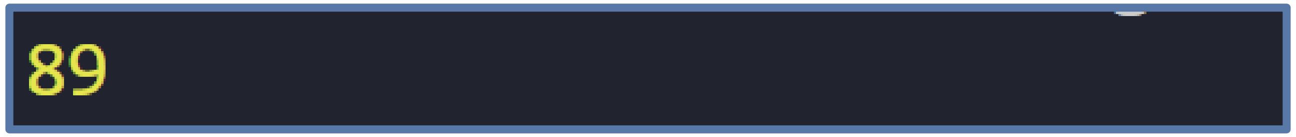


TS Enums.ts > ...

```
enum chars {Lumine = 34, Amber = 18, Noelle = 89}
let myChar = chars.Noelle;
console.log(myChar);
```



```
\TypeScriptCodes> tsc Enums.ts
\TypeScriptCodes> node Enums.js
```

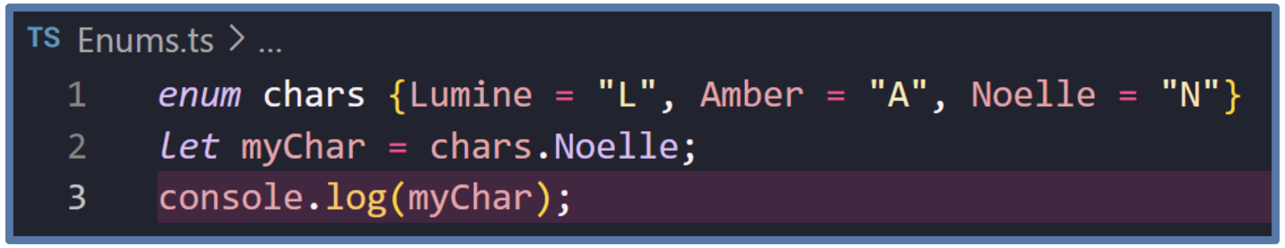


```
89
```

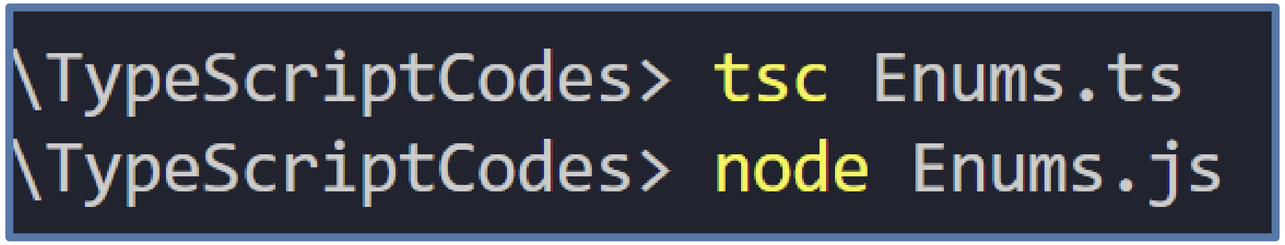
ENUMS

STRING

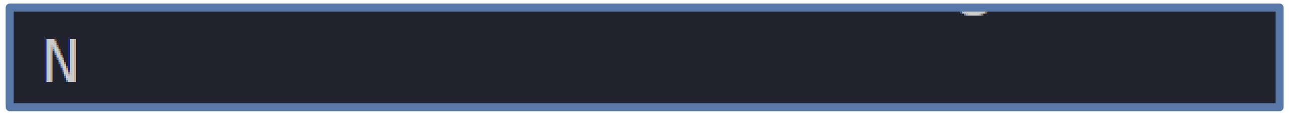
enums can also contain strings and it is more common than numbers because of its readability and intent.



```
TS Enums.ts > ...
enum chars {Lumine = "L", Amber = "A", Noelle = "N"}
let myChar = chars.Noelle;
console.log(myChar);
```



```
\TypeScriptCodes> tsc Enums.ts
\TypeScriptCodes> node Enums.js
```



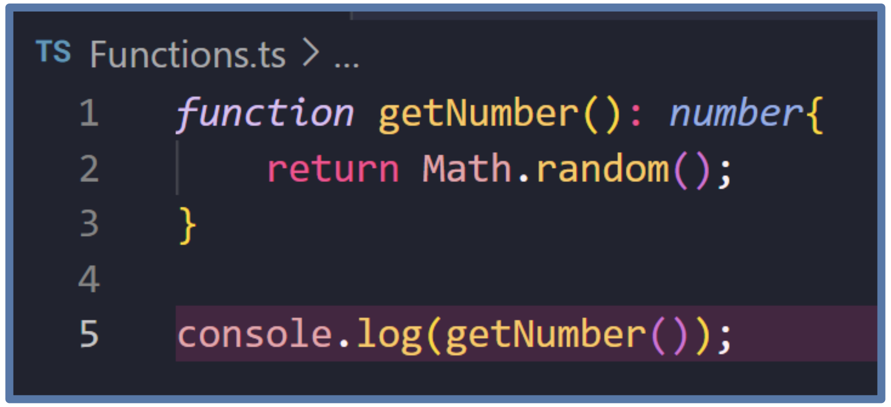
```
N
```

4

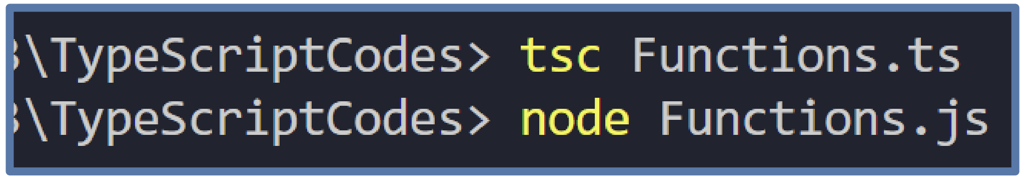
FUNCTIONS

RETURN TYPE

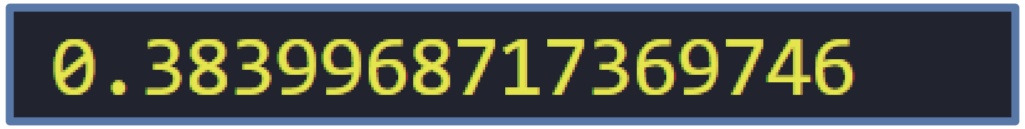
- TypeScript has a specific syntax for typing function parameters and return values.
- In return type the type of value returned by the function can be defined explicitly.


TS Functions.ts > ...

```
function getNumber(): number{
    return Math.random();
}

console.log(getNumber());
```



```
\TypeScriptCodes> tsc Functions.ts
\TypeScriptCodes> node Functions.js
```



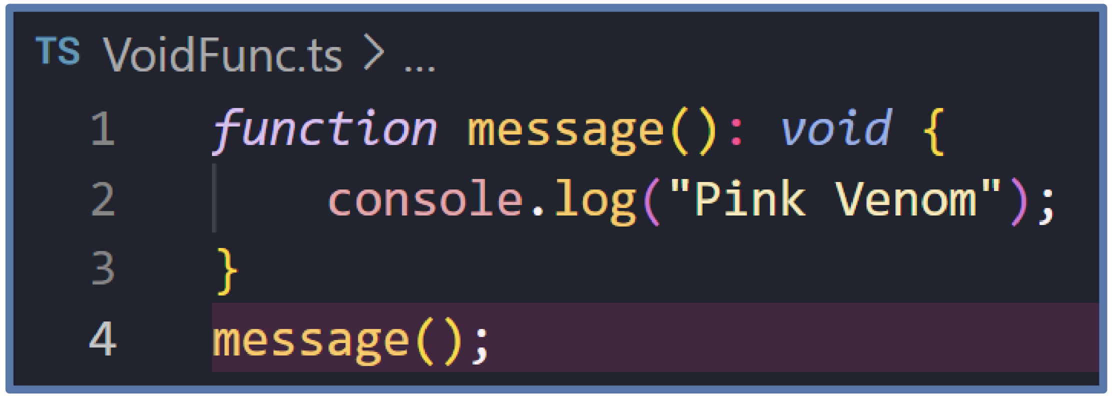
```
0.3839968717369746
```

4

FUNCTIONS

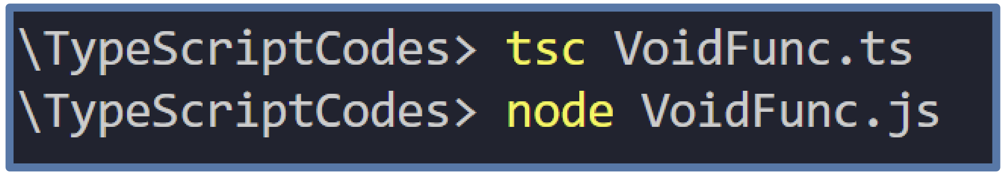
VOID RETURN TYPE

A void type indicates that a function does not return any value.

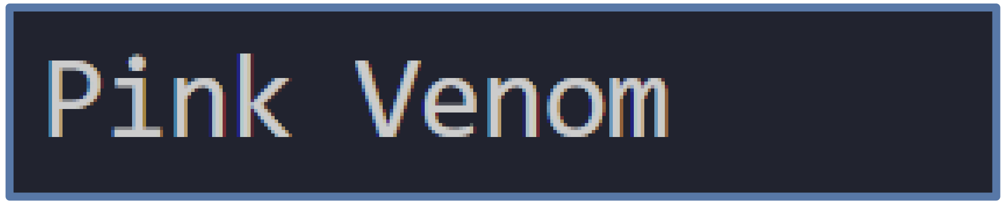


TS VoidFunc.ts > ...

```typescript
function message(): void {
    console.log("Pink Venom");
}
message();
```



```
\TypeScriptCodes> tsc VoidFunc.ts
\TypeScriptCodes> node VoidFunc.js
```



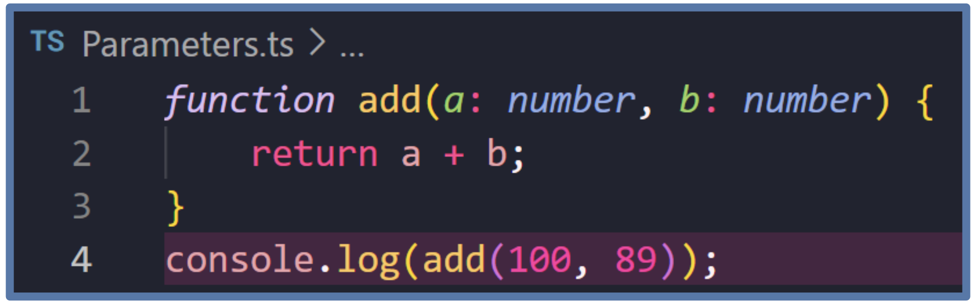
```
Pink Venom
```

4

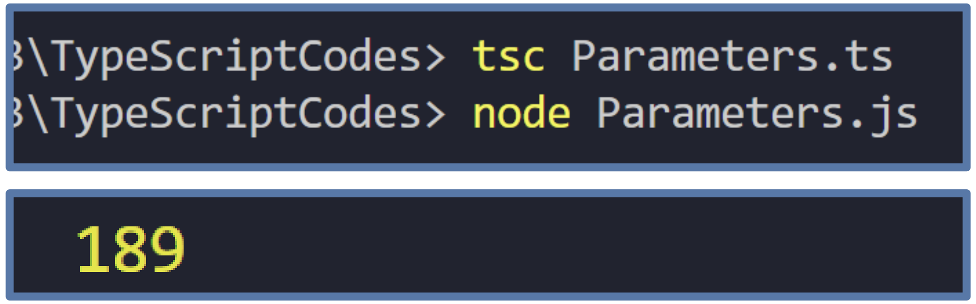
FUNCTIONS

PARAMETERS

Type with similar syntax as variable declarations. If no parameter is defined TypeScript will use any as default.


TS Parameters.ts > ...

```typescript
function add(a: number, b: number) {
    return a + b;
}
console.log(add(100, 89));
```



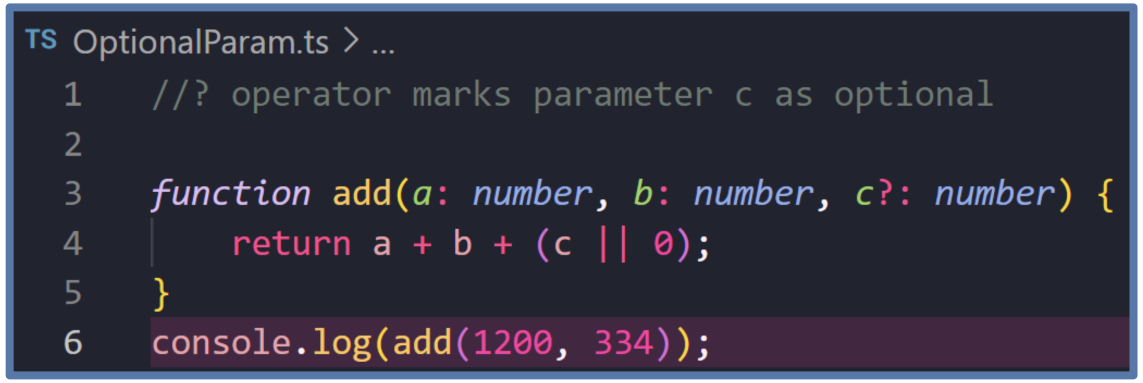
```
3\TypeScriptCodes> tsc Parameters.ts
3\TypeScriptCodes> node Parameters.js
```

```
189
```

FUNCTIONS

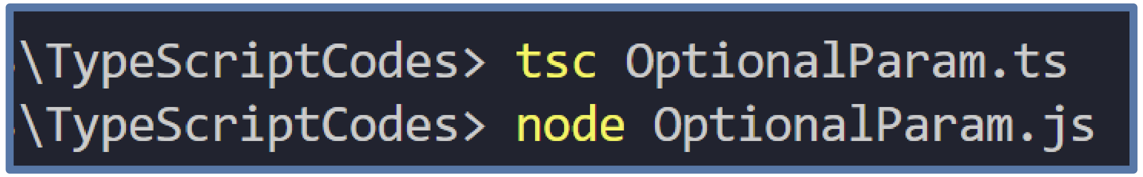
OPTIONAL PARAMETERS

TypeScript assume that all parameters are required but **can be specified as optional**.

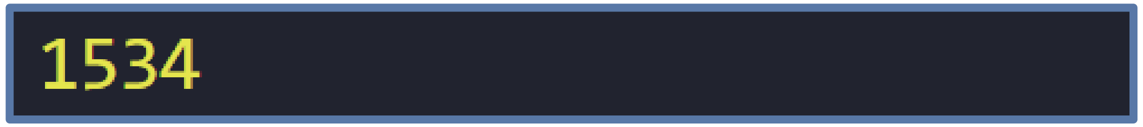


TS OptionalParam.ts > ...

```
//? operator marks parameter c as optional

function add(a: number, b: number, c?: number) {
    return a + b + (c || 0);
}
console.log(add(1200, 334));
```



```
\TypeScriptCodes> tsc OptionalParam.ts
\TypeScriptCodes> node OptionalParam.js
```

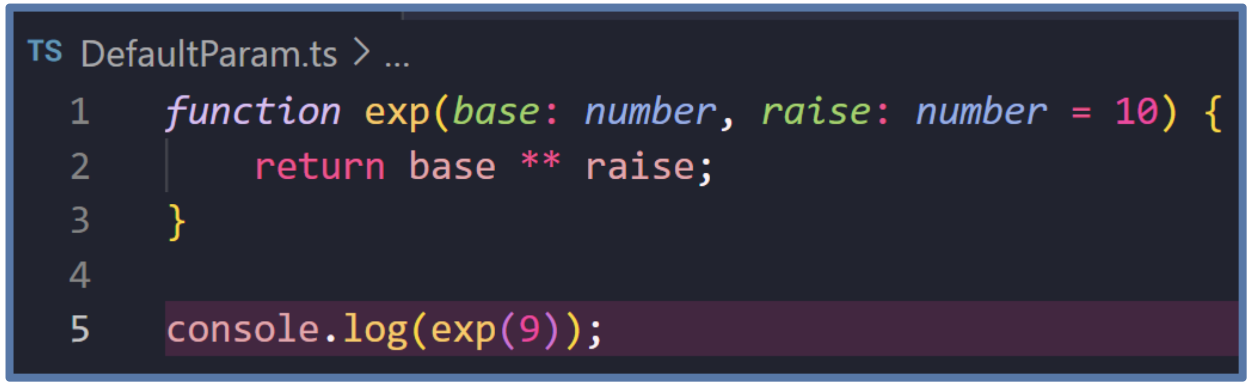


```
1534
```

4 FUNCTIONS

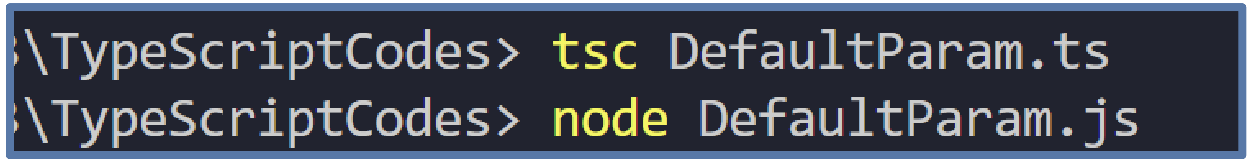
DEFAULT PARAMETERS

Parameters with default values, default value goes after the annotation.

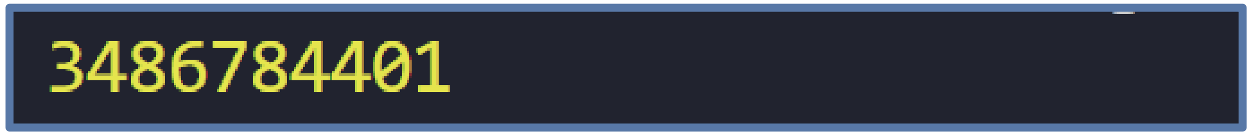


TS DefaultParam.ts > ...

```typescript
function exp(base: number, raise: number = 10) {
    return base ** raise;
}

console.log(exp(9));
```



```
\TypeScriptCodes> tsc DefaultParam.ts
\TypeScriptCodes> node DefaultParam.js
```

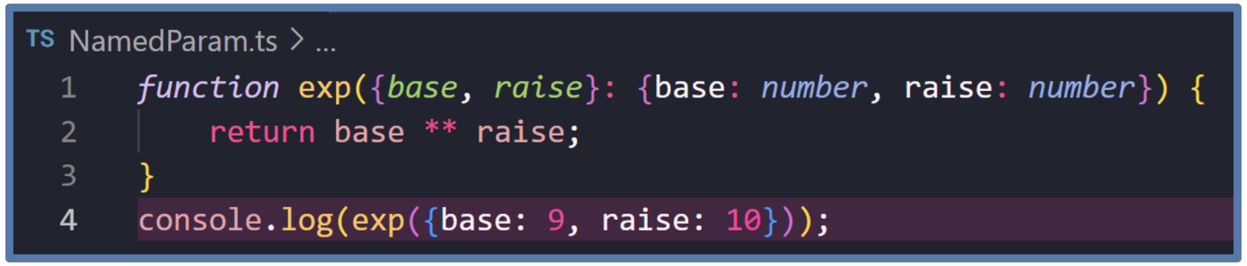


```
3486784401
```

4 FUNCTIONS

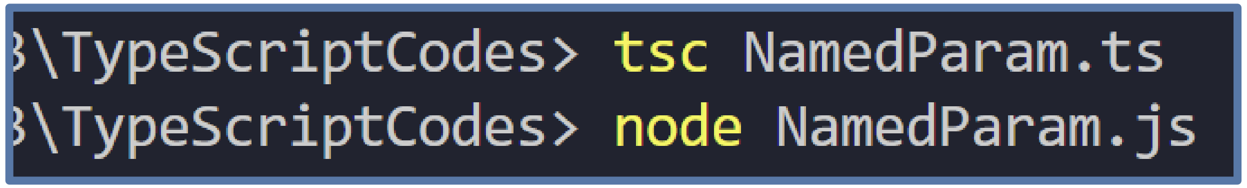
NAMED PARAMETERS

Typing named parameters follows the same pattern as normal parameters.

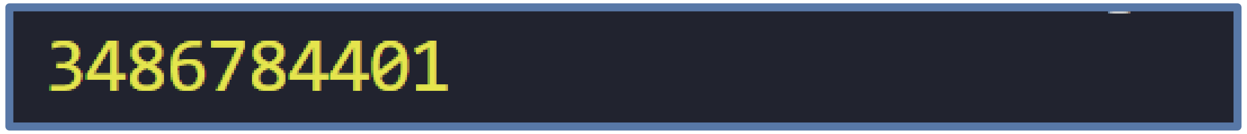


NamedParam.ts > ...

```typescript
function exp({base, raise}: {base: number, raise: number}) {
    return base ** raise;
}
console.log(exp({base: 9, raise: 10}));
```



```
3\TypeScriptCodes> tsc NamedParam.ts
3\TypeScriptCodes> node NamedParam.js
```

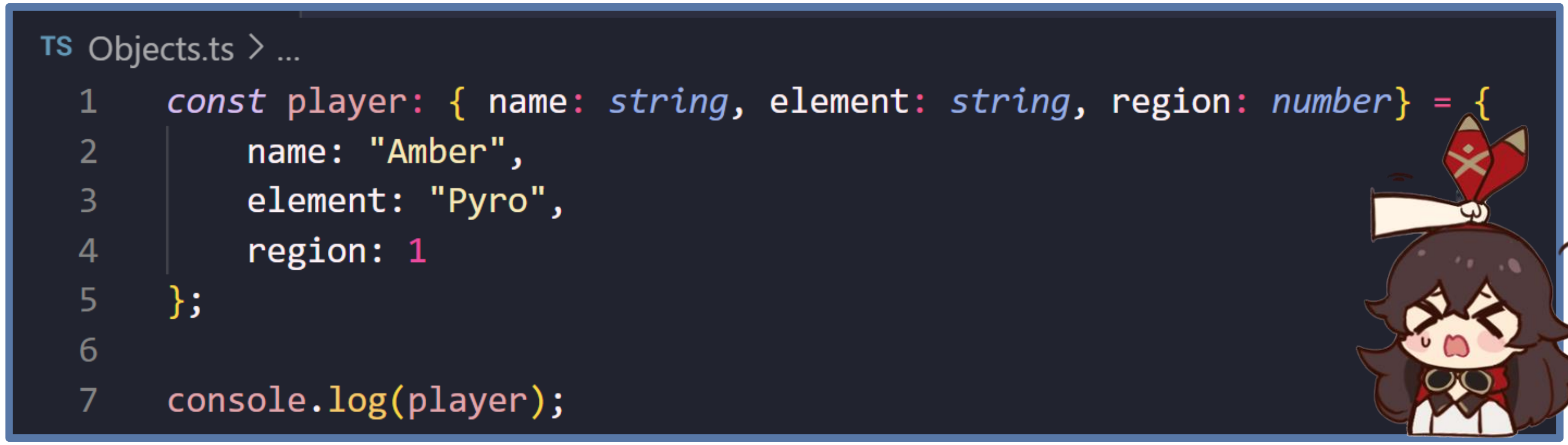


```
3486784401
```

5

OBJECT TYPES

TypeScript has specific syntax for typing objects.


TS Objects.ts > ...

```
const player: { name: string, element: string, region: number} = {
    name: "Amber",
    element: "Pyro",
    region: 1
};

console.log(player);
```

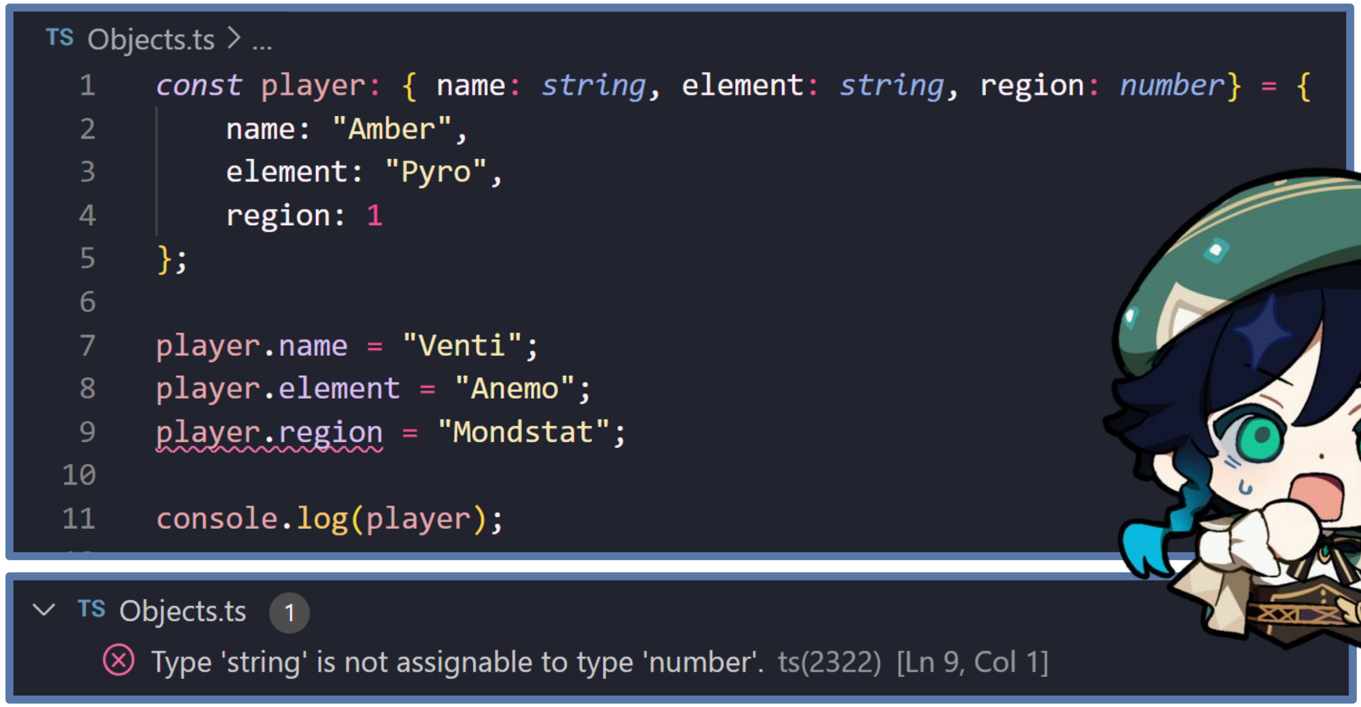
Try playing around with modifying properties and adding ones to see what happens.

5

OBJECT TYPES

TYPE INFERENCE

TypeScript can also infer the types of properties based on their values.



Objects.ts > ...

```typescript
const player: { name: string, element: string, region: number} = {
    name: "Amber",
    element: "Pyro",
    region: 1
};

player.name = "Venti";
player.element = "Anemo";
player.region = "Mondstat";

console.log(player);
```

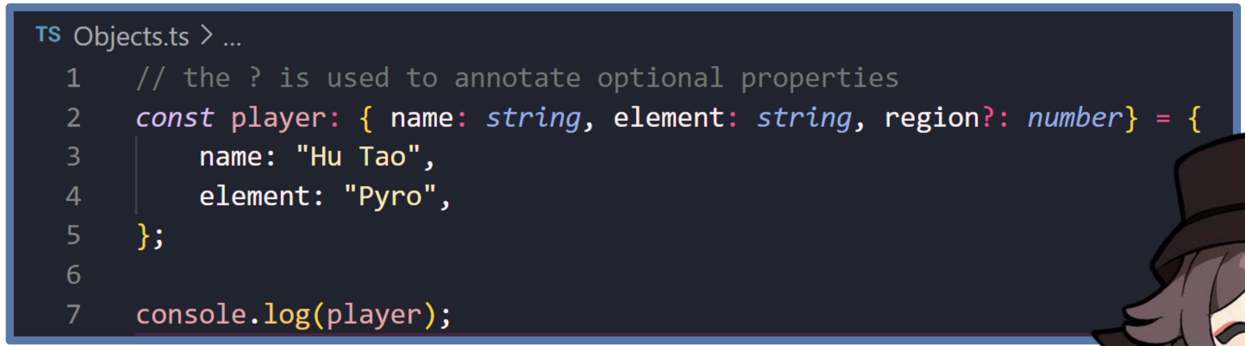
Objects.ts 1

Type 'string' is not assignable to type 'number'. ts(2322) [Ln 9, Col 1]

5 OBJECT TYPES

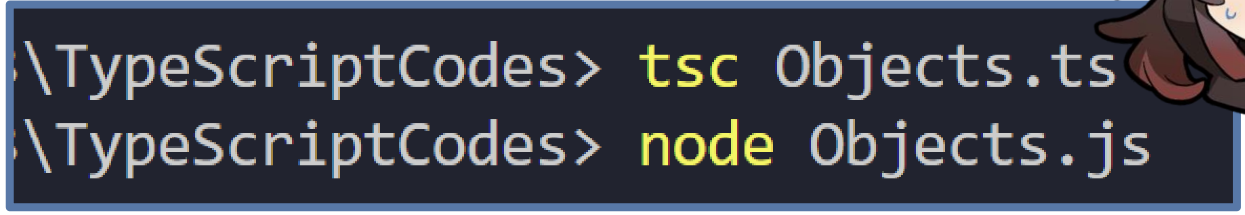
OPTIONAL PROPERTY

Optional properties are properties that don’t have to be defined in the object definition.

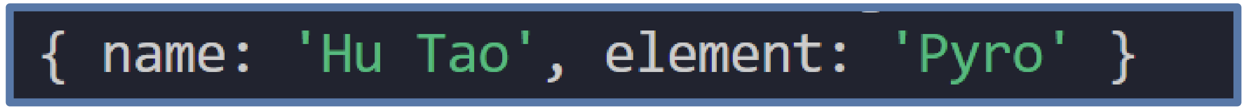


Objects.ts > ...

```typescript
// the ? is used to annotate optional properties
const player: { name: string, element: string, region?: number} = {
    name: "Hu Tao",
    element: "Pyro",
};

console.log(player);
```



```
\TypeScriptCodes> tsc Objects.ts
\TypeScriptCodes> node Objects.js
```



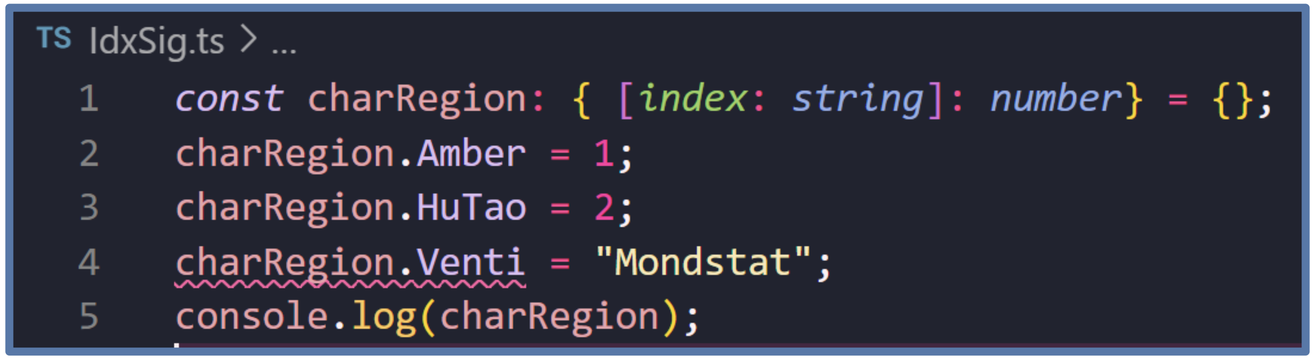
```
{ name: 'Hu Tao', element: 'Pyro' }
```

5

OBJECT TYPES

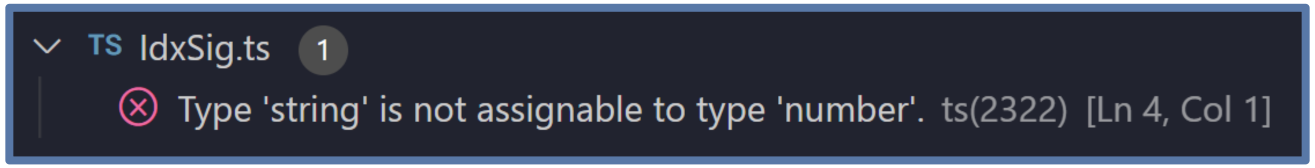
INDEX SIGNATURE

Index signatures can be used for objects without a defined list of properties.



IdxSig.ts > ...

```typescript
const charRegion: { [index: string]: number} = {};
charRegion.Amber = 1;
charRegion.HuTao = 2;
charRegion.Venti = "Mondstat";
console.log(charRegion);
```



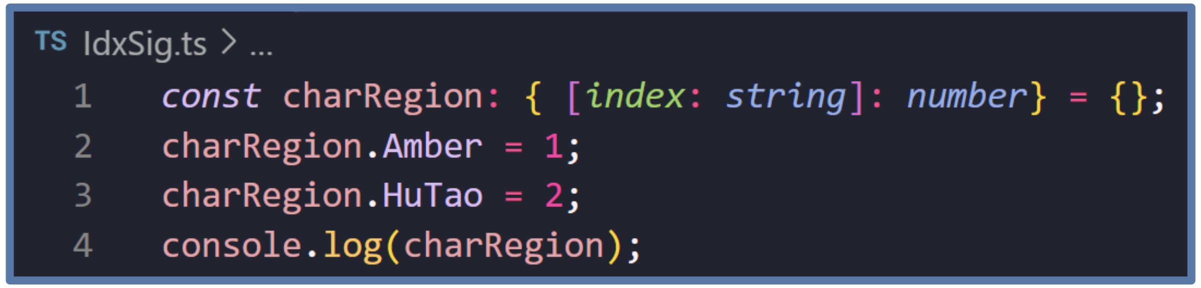
```
IdxSig.ts 1
    Type 'string' is not assignable to type 'number'. ts(2322) [Ln 4, Col 1]
```

5

OBJECT TYPES

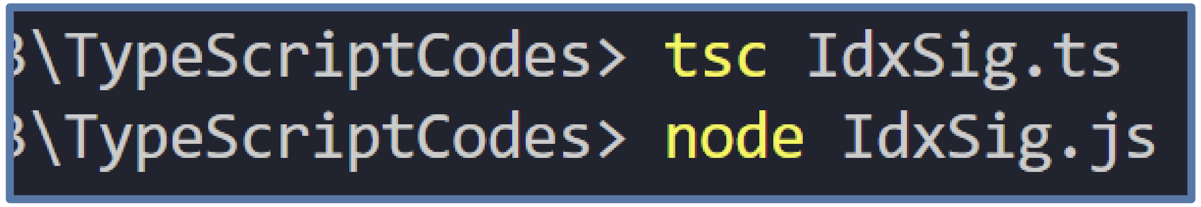
INDEX SIGNATURE

Index signatures can be used for objects without a defined list of properties.

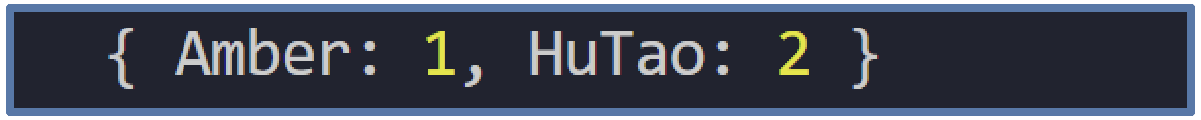


TS IdxSig.ts > ...

```typescript
const charRegion: { [index: string]: number} = {};
charRegion.Amber = 1;
charRegion.HuTao = 2;
console.log(charRegion);
```



```
3\TypeScriptCodes> tsc IdxSig.ts
3\TypeScriptCodes> node IdxSig.js
```



```
{ Amber: 1, HuTao: 2 }
```

Thank You!