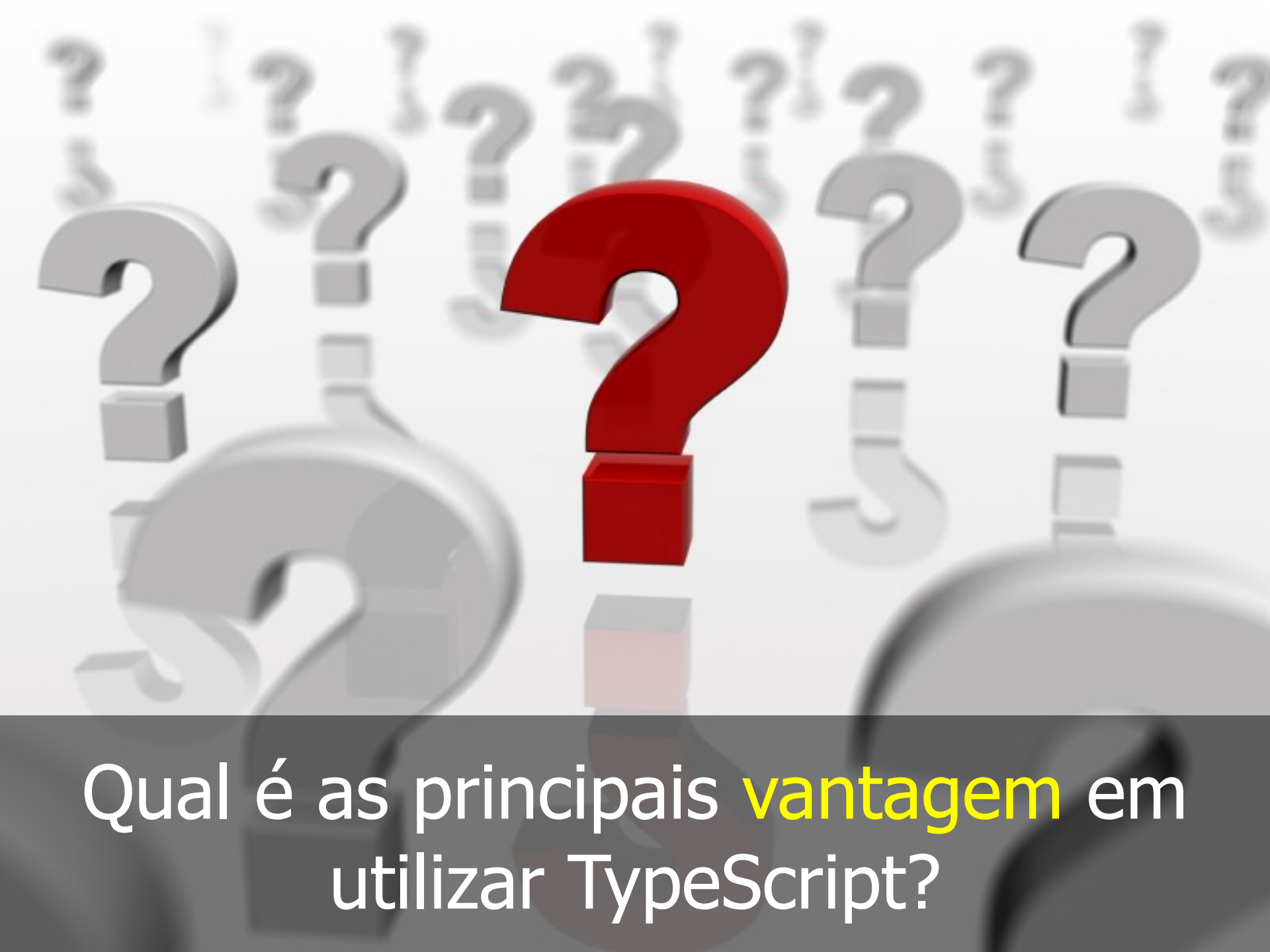
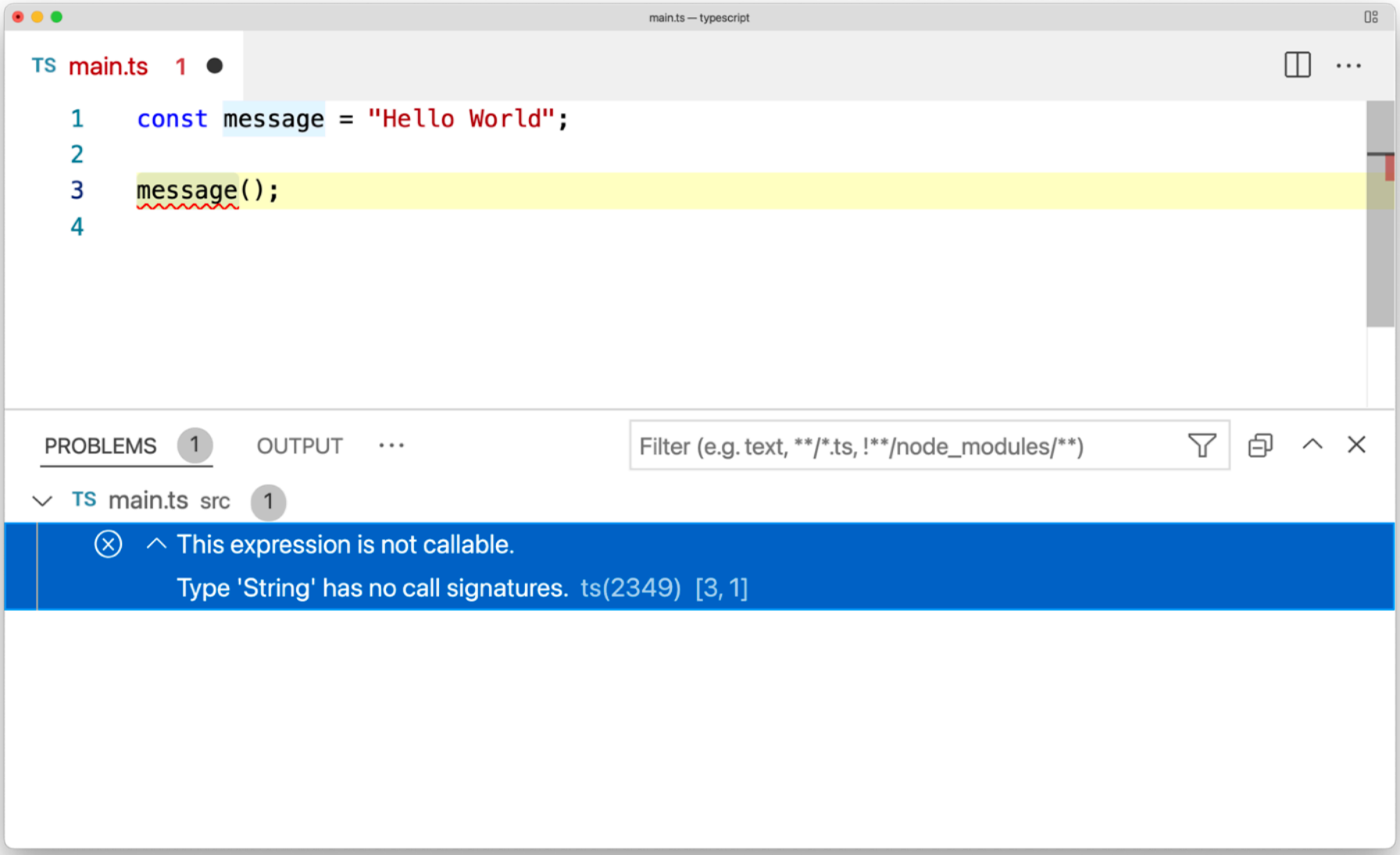
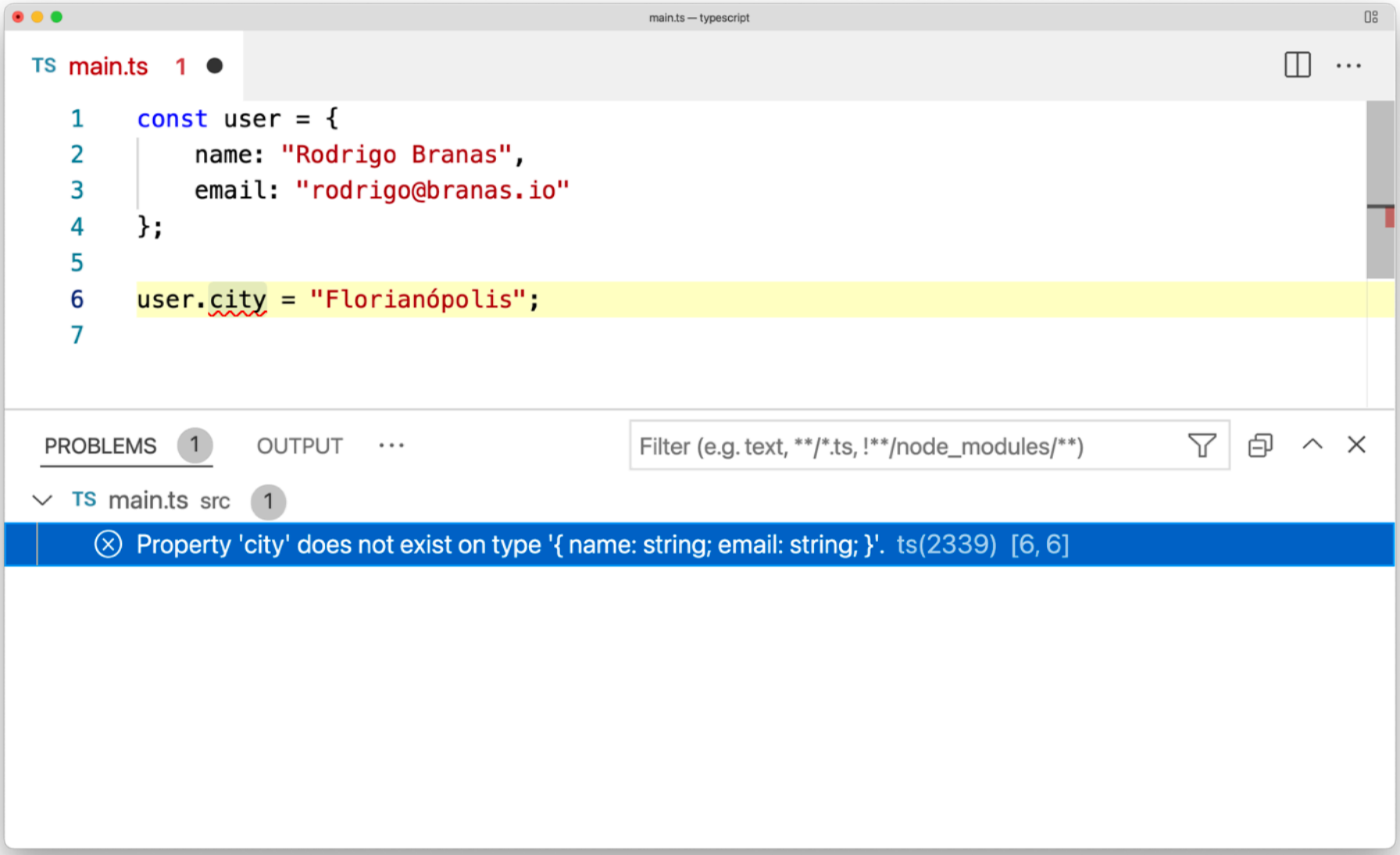

TS
TypeScript

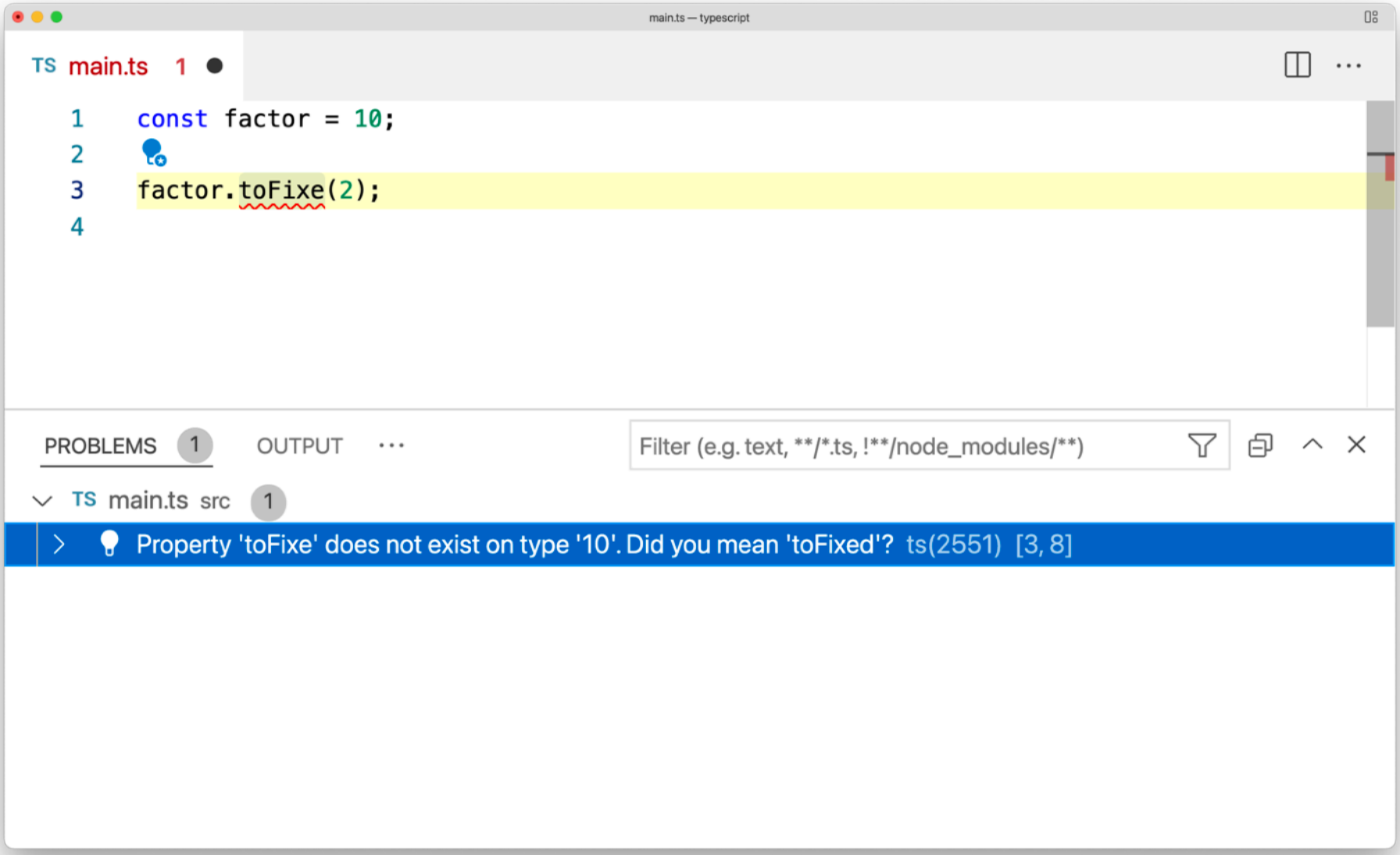


Qual é as principais vantagem em utilizar TypeScript?

Pense nos erros que existiem na linguagem JavaScript, em tempo de interpretação. Boa parte deles seria facilmente evitada se uma verificação da tipagem fosse realizada enquanto programamos


main.ts — typescript
TS main.ts 1
1 const message = "Hello World";
2
3 message();
4
PROBLEMS 1
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 1
This expression is not callable.
Type 'String' has no call signatures. ts(2349) [3, 1]


main.ts — typescript
TS main.ts 1
1 const user = {
2 name: "Rodrigo Branas",
3 email: "rodrigo@branas.io"
4 };
5
6 user.city = "Florianópolis";
7
PROBLEMS 1
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 1
Property 'city' does not exist on type '{ name: string; email: string; }'. ts(2339) [6, 6]

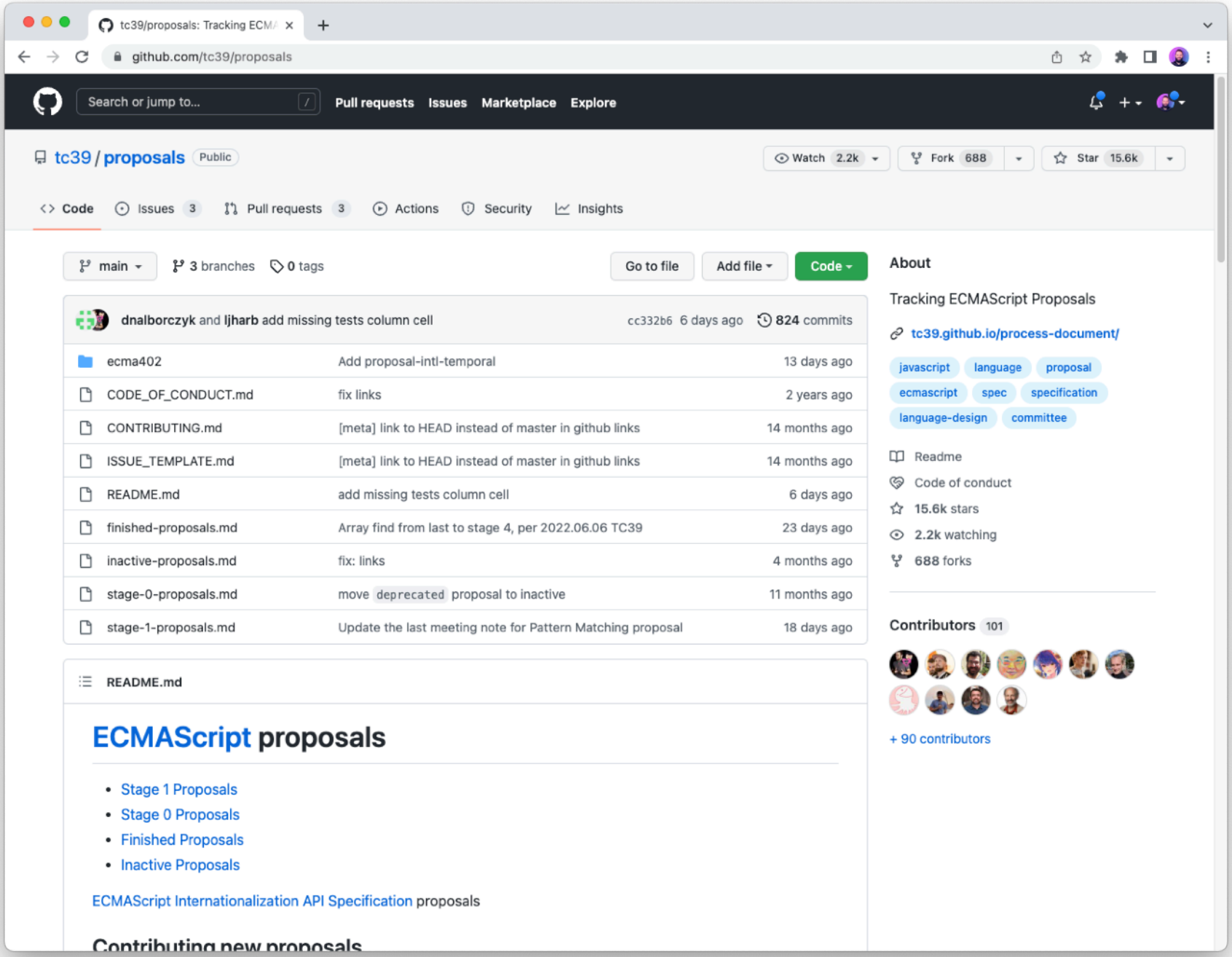

main.ts — typescript
TS main.ts 1
1 const factor = 10;
2
3 factor.toFixe(2);
4
PROBLEMS 1
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 1
Property 'toFixe' does not exist on type '10'. Did you mean 'toFixed'? ts(2551) [3, 8]

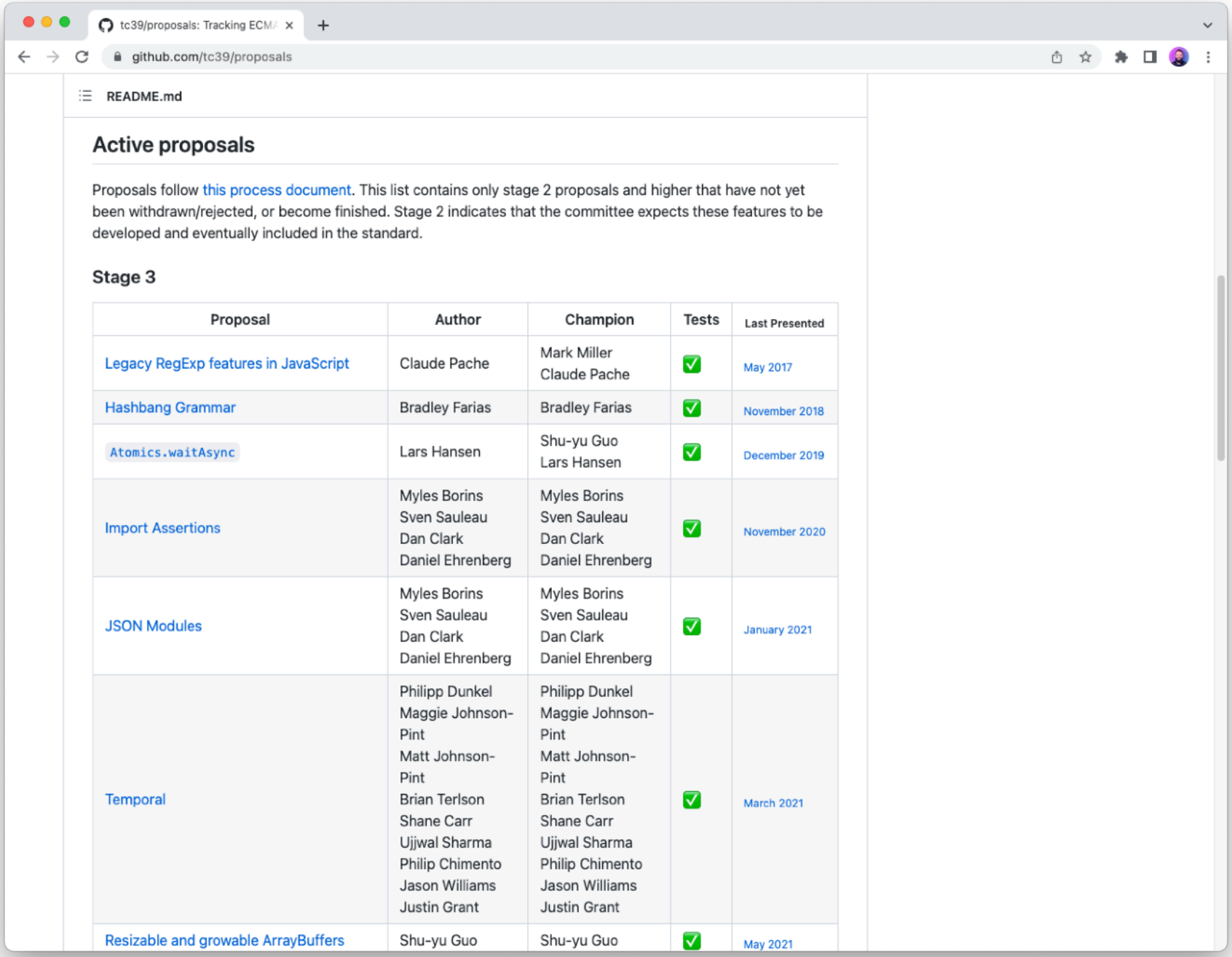
Além disso, temos mais suporte a OO, permitindo uma implementação muito mais alinhada com os princípios do SOLID e especialmente em projetos com regras de negócio mais complexas

Não que não seja possível utilizar abordagens orientadas ao domínio como o DDD com JavaScript mas tipos, interfaces, classes abstratas e modificadores de visibilidade fazem muita diferença na implementação

Existem ainda funcionalidades como os decorators, que ajudam muito na configuração de frameworks e bibliotecas

Por fim, antecipar as novas funcionalidades que ainda estão em stage 3 e 4 no TC39 e que em breve serão lançadas na linguagem JavaScript


tc39/proposals: Tracking ECMA
github.com/tc39/proposals
Search or jump to...
Pull requests
Issues
Marketplace
Explore
tc39 / proposals
Public
Watch 2.2k
Fork 688
Star 15.6k
Code
Issues 3
Pull requests 3
Actions
Security
Insights
main
3 branches
0 tags
Go to file
Add file
Code
dnalborczyk and ljharb add missing tests column cell
cc332b6 6 days ago
824 commits
ecma402 Add proposal-intl-temporal 13 days ago
CODE_OF_CONDUCT.md fix links 2 years ago
CONTRIBUTING.md [meta] link to HEAD instead of master in github links 14 months ago
ISSUE_TEMPLATE.md [meta] link to HEAD instead of master in github links 14 months ago
README.md add missing tests column cell 6 days ago
finished-proposals.md Array find from last to stage 4, per 2022.06.06 TC39 23 days ago
inactive-proposals.md fix: links 4 months ago
stage-0-proposals.md move deprecated proposal to inactive 11 months ago
stage-1-proposals.md Update the last meeting note for Pattern Matching proposal 18 days ago
README.md
ECMAScript proposals
Stage 1 Proposals
Stage 0 Proposals
Finished Proposals
Inactive Proposals
ECMAScript Internationalization API Specification proposals
Contributing new proposals
About
Tracking ECMAScript Proposals
tc39.github.io/process-document/
javascript
language
proposal
ecmascript
spec
specification
language-design
committee
Readme
Code of conduct
15.6k stars
2.2k watching
688 forks
Contributors 101
+ 90 contributors



Active proposals

Proposals follow this process document. This list contains only stage 2 proposals and higher that have not yet been withdrawn/rejected, or become finished. Stage 2 indicates that the committee expects these features to be developed and eventually included in the standard.

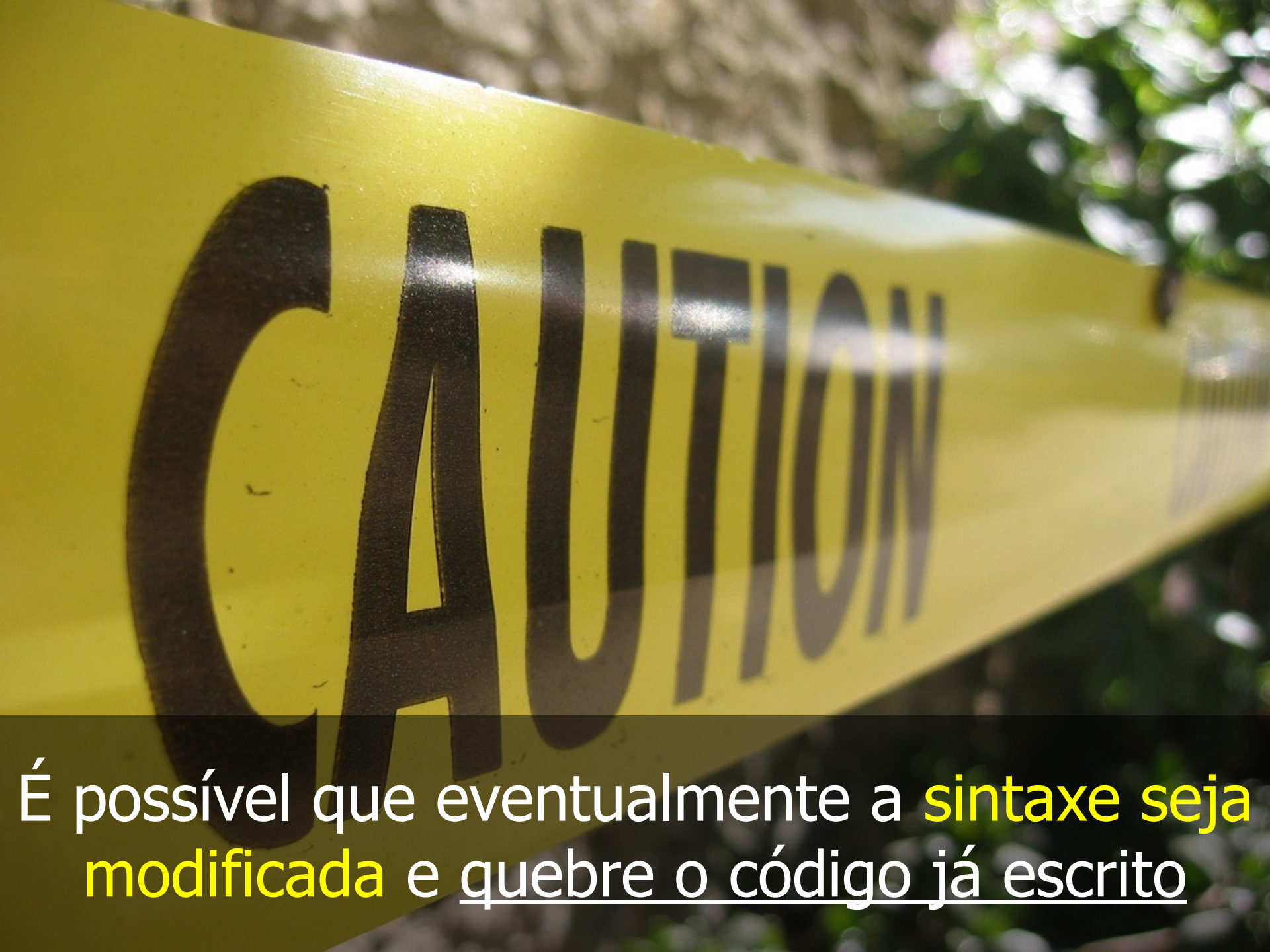
Stage 3

Proposal	Author	Champion	Tests	Last Presented
Legacy RegExp features in JavaScript	Claude Pache	Mark Miller Claude Pache	✅	May 2017
Hashbang Grammar	Bradley Farias	Bradley Farias	✅	November 2018
`Atomics.waitAsync`	Lars Hansen	Shu-yu Guo Lars Hansen	✅	December 2019
Import Assertions	Myles Borins Sven Sauleau Dan Clark Daniel Ehrenberg	Myles Borins Sven Sauleau Dan Clark Daniel Ehrenberg	✅	November 2020
JSON Modules	Myles Borins Sven Sauleau Dan Clark Daniel Ehrenberg	Myles Borins Sven Sauleau Dan Clark Daniel Ehrenberg	✅	January 2021
Temporal	Philipp Dunkel Maggie Johnson-Pint Matt Johnson-Pint Brian Terlson Shane Carr Ujjwal Sharma Philip Chimento Jason Williams Justin Grant	Philipp Dunkel Maggie Johnson-Pint Matt Johnson-Pint Brian Terlson Shane Carr Ujjwal Sharma Philip Chimento Jason Williams Justin Grant	✅	March 2021
Resizable and growable ArrayBuffers	Shu-yu Guo	Shu-yu Guo	✅	May 2021

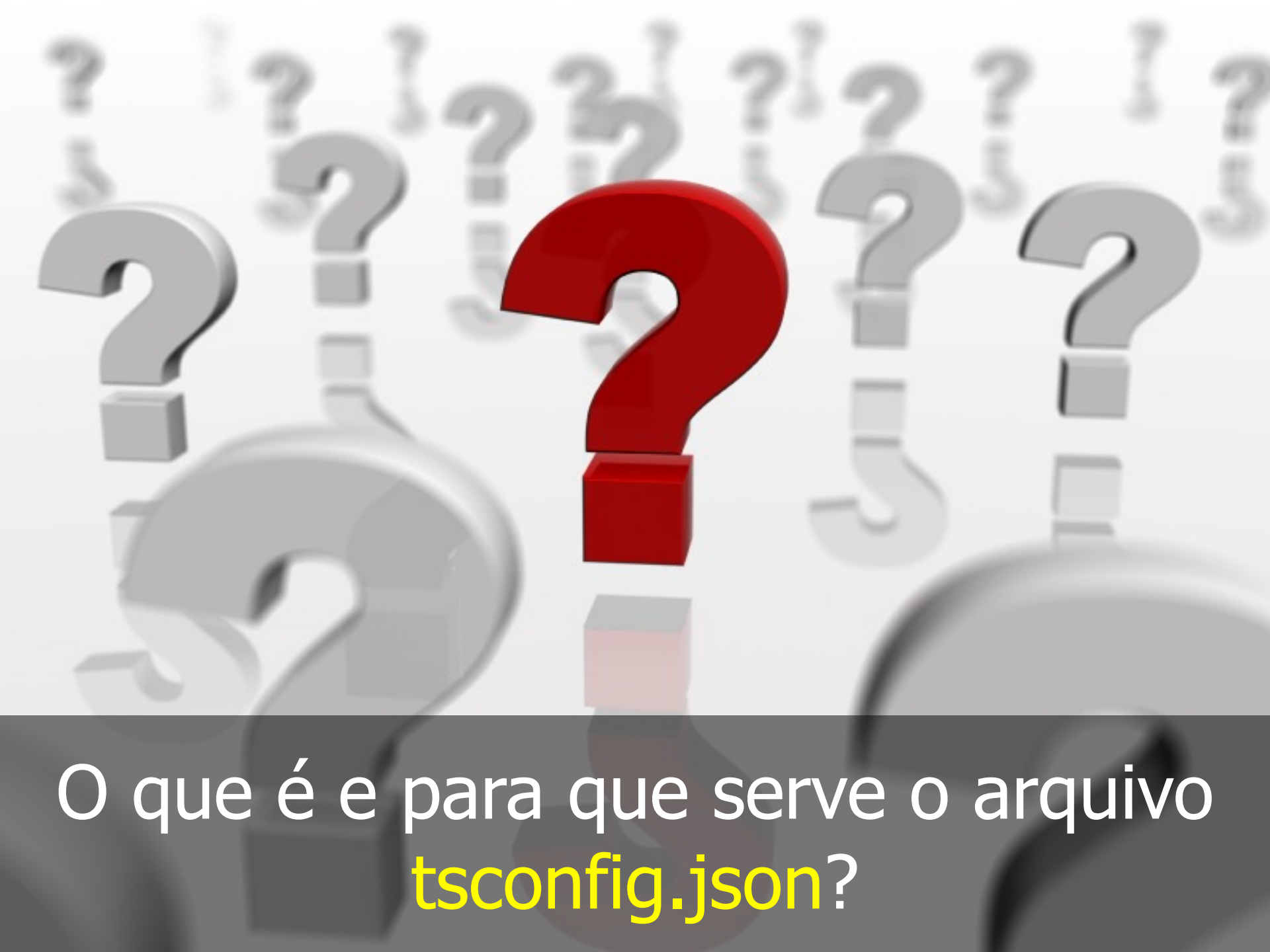
README.md

Stage 2

Proposal	Author	Champion	Last Presented
`function.sent` metaproperty	Allen Wirfs-Brock	HE Shi-Jun	July 2019
`throw` expressions	Ron Buckton	Ron Buckton	January 2018
Function implementation hiding	Domenic Denicola Michael Ficarra	Michael Ficarra	June 2020
New Set methods	Michał Wadas Sathya Gunasekaran	Sathya Gunasekaran	January 2019
Sequence properties in Unicode property escapes	Mathias Bynens	Mathias Bynens	October 2019
collection normalization	Bradley Farias	Bradley Farias	January 2019
Array.isTemplateObject	Mike Samuel, Krzysztof Kotowicz	Krzysztof Kotowicz	December 2019
Iterator helpers	Gus Caplan	Michael Ficarra Jonathan Keslin	July 2020
Explicit Resource Management	Ron Buckton	Ron Buckton	February 2020
`Map.prototype.emplace`	Bradley Farias	Erica Pramer	July 2020
Dynamic Import Host Adjustment	Mike Samuel, Krzysztof Kotowicz	Krzysztof Kotowicz	December 2019
WeakRefs `cleanupSome`	Yulia Startsev Daniel Ehrenberg	Yulia Startsev Daniel Ehrenberg	July 2020
Record & Tuple	Robin Ricard Richard Button	Robin Ricard Richard Button	December 2021
JSON.parse source text access	Richard Gibson	Richard Gibson	July 2020
Module Blocks	Surma Daniel Ehrenberg	Surma	January 2021
Pipeline Operator	J. S. Choi James DiGioia	J. S. Choi Ron Buckton	August 2021

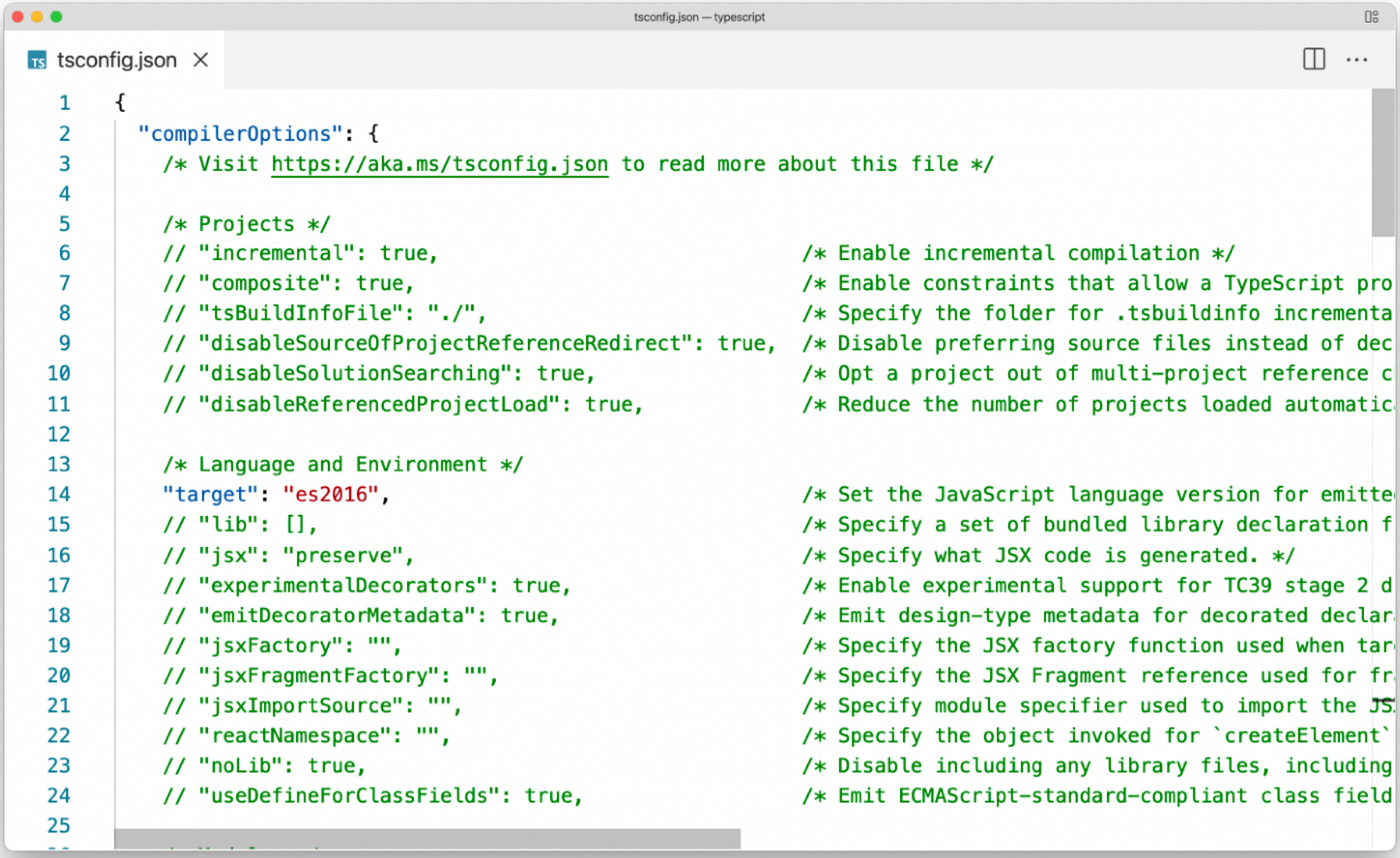


É possível que eventualmente a sintaxe seja modificada e quebre o código já escrito



O que é e para que serve o arquivo tsconfig.json?

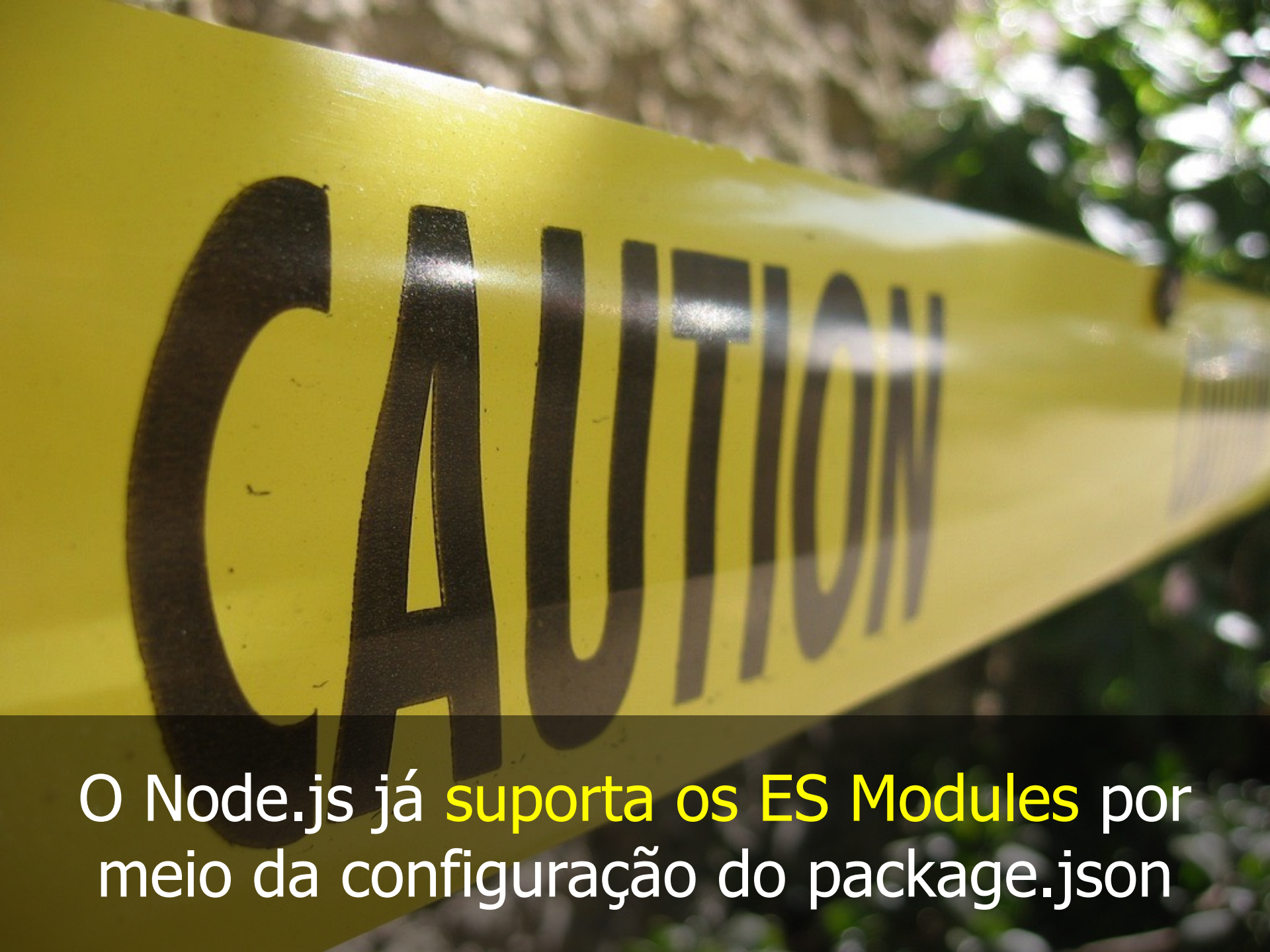
O arquivo tsconfig.json centraliza todas as regras de análise estática e de transpilação e é importante configurá-lo de forma adequada, conforme as necessidades e políticas do projeto


tsconfig.json — typescript
tsconfig.json
{
"compilerOptions": {
/* Visit https://aka.ms/tsconfig.json to read more about this file */
/* Projects */
// "incremental": true, /* Enable incremental compilation */
// "composite": true, /* Enable constraints that allow a TypeScript pro
// "tsBuildInfoFile": "./", /* Specify the folder for .tsbuildinfo incrementa
// "disableSourceOfProjectReferenceRedirect": true, /* Disable preferring source files instead of dec
// "disableSolutionSearching": true, /* Opt a project out of multi-project reference c
// "disableReferencedProjectLoad": true, /* Reduce the number of projects loaded automatic
/* Language and Environment */
"target": "es2016", /* Set the JavaScript language version for emitte
// "lib": [], /* Specify a set of bundled library declaration f
// "jsx": "preserve", /* Specify what JSX code is generated. */
// "experimentalDecorators": true, /* Enable experimental support for TC39 stage 2 d
// "emitDecoratorMetadata": true, /* Emit design-type metadata for decorated declar
// "jsxFactory": "", /* Specify the JSX factory function used when tar
// "jsxFragmentFactory": "", /* Specify the JSX Fragment reference used for fr
// "jsxImportSource": "", /* Specify module specifier used to import the JS
// "reactNamespace": "", /* Specify the object invoked for `createElement`
// "noLib": true, /* Disable including any library files, including
// "useDefineForClassFields": true, /* Emit ECMAScript-standard-compliant class field

A opção incremental serve para habilitar a trnaspilação incremental, guardando uma cache que agiliza futuras transpilações especialmente em projetos muito grandes

A opção target define qual é a versão da linguagem JavaScript que será utilizada como destino da transpilação

A opção module é importante para definir o tipo de sistema de módulo que será utilizado, sendo que os mais importantes são CommonJS ou os ES Modules, lançado no ES6



O Node.js já suporta os ES Modules por meio da configuração do package.json

A opção include define quais são os diretórios que devem ser transpilados pelo TypeScript enquanto o exclude ignora um conjunto de diretórios

Emit

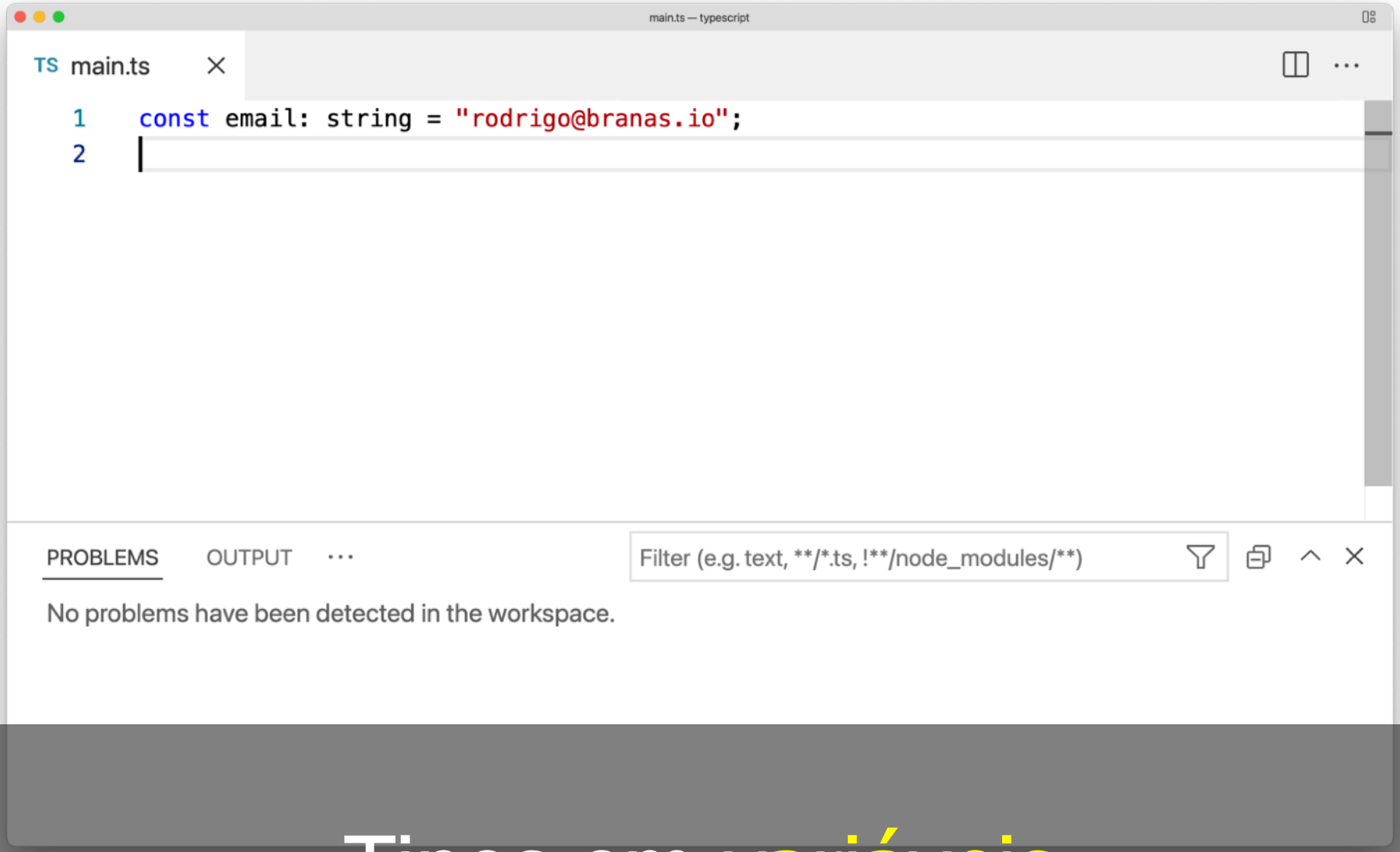
O grupo de configurações emit serve para definir parâmetros relacionados com a transpilação

A opção outFile é muito utilizada para projetos que rodam no browser definindo um arquivo que vai concatenar todos os arquivos que foram transpilados

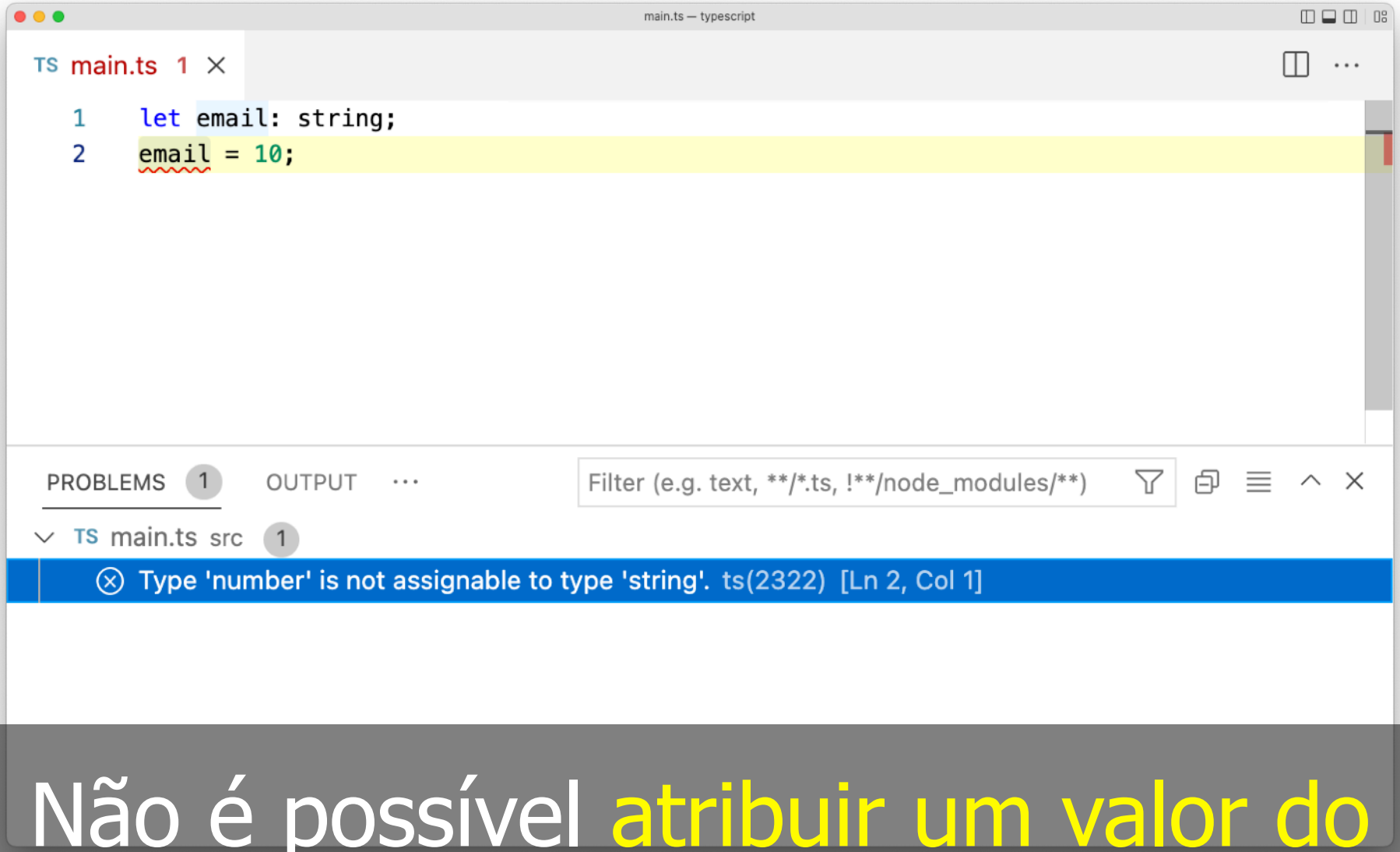
A opção outDir é essencial já que define o diretório de armazenamento dos arquivos transpilados

Definição de tipos

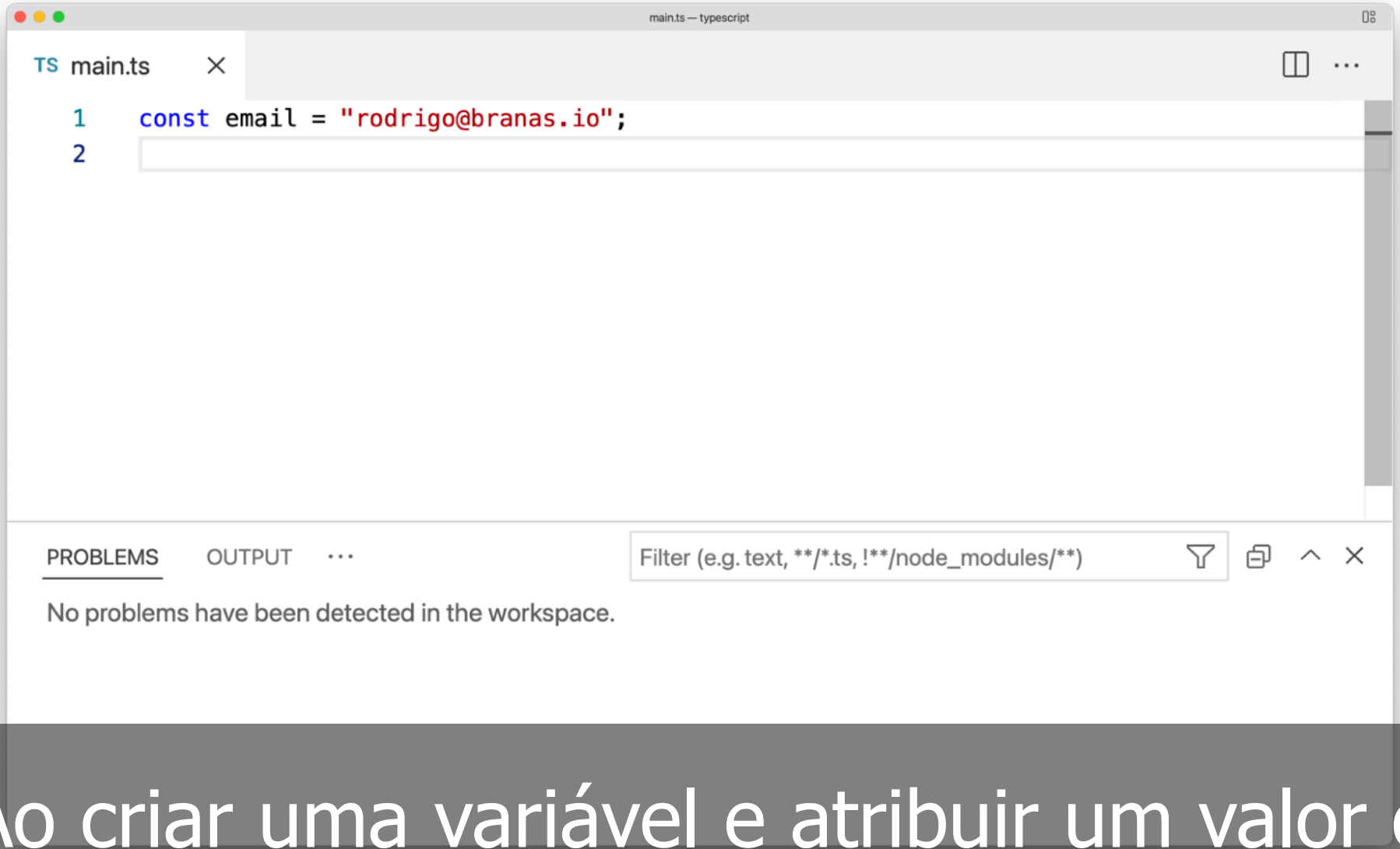
Definição dos tipos em variáveis, parâmetros e retornos de funções e propriedades de objetos



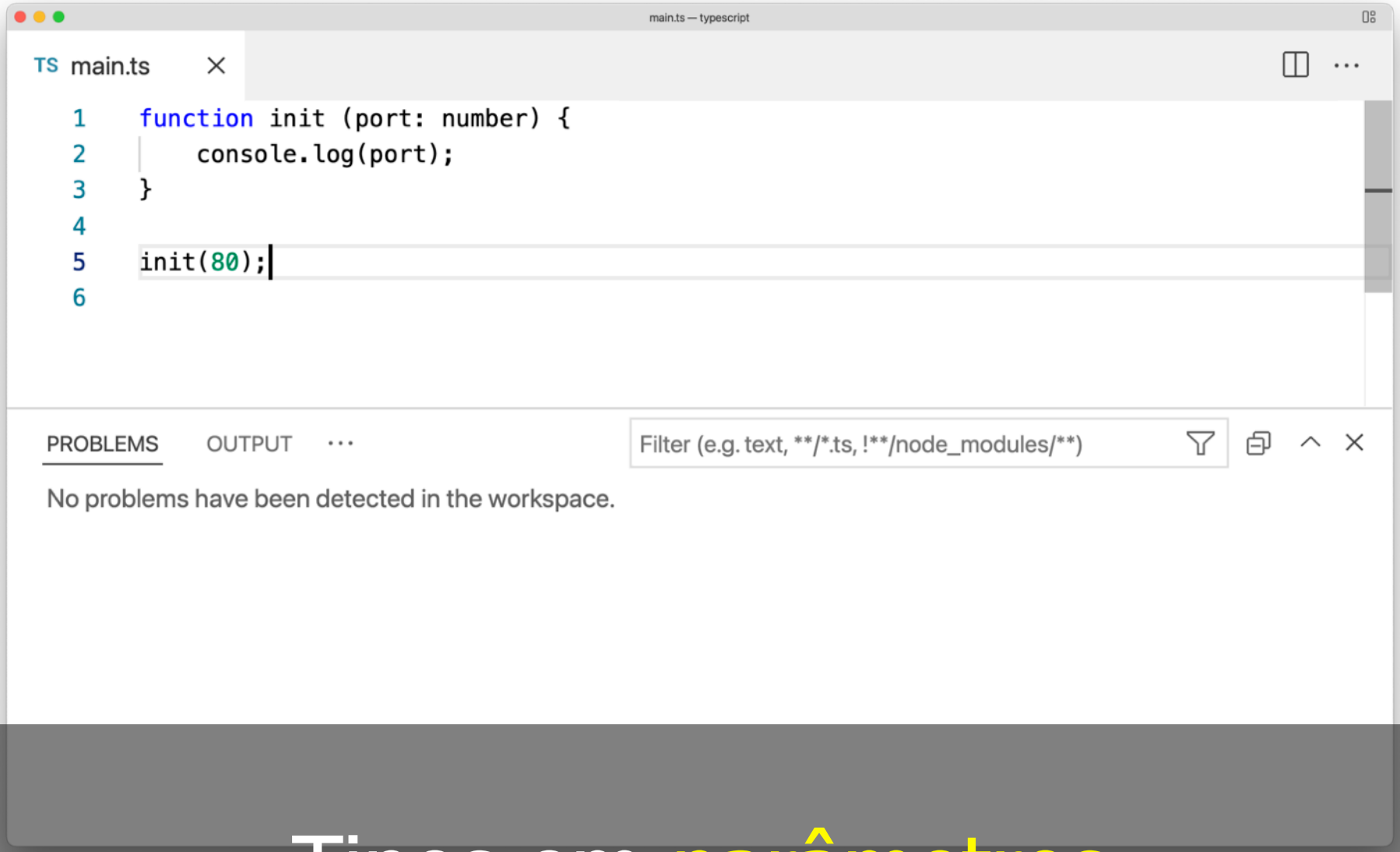
Tipos em variáveis



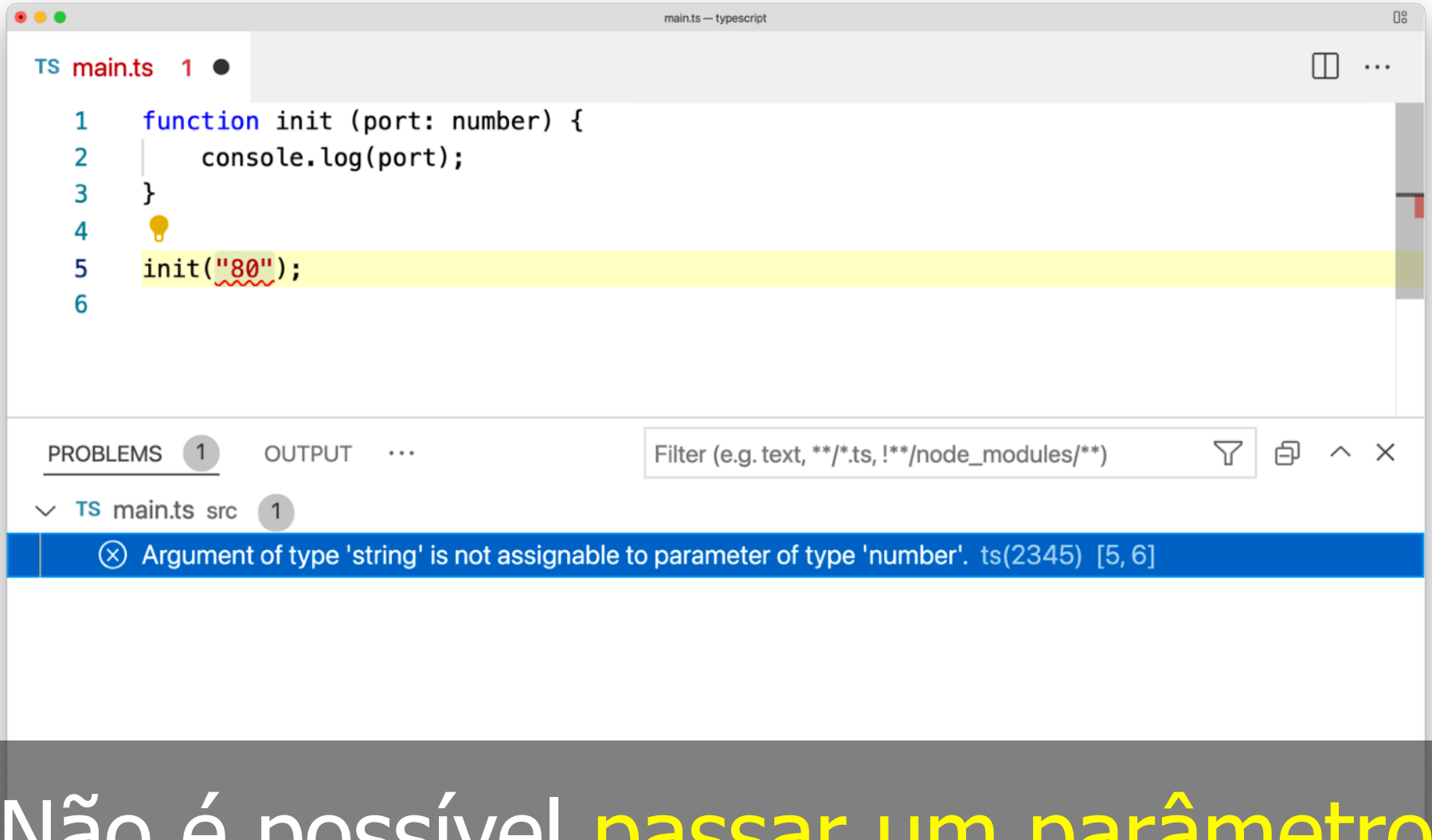
Não é possível atribuir um valor do tipo errado na variável



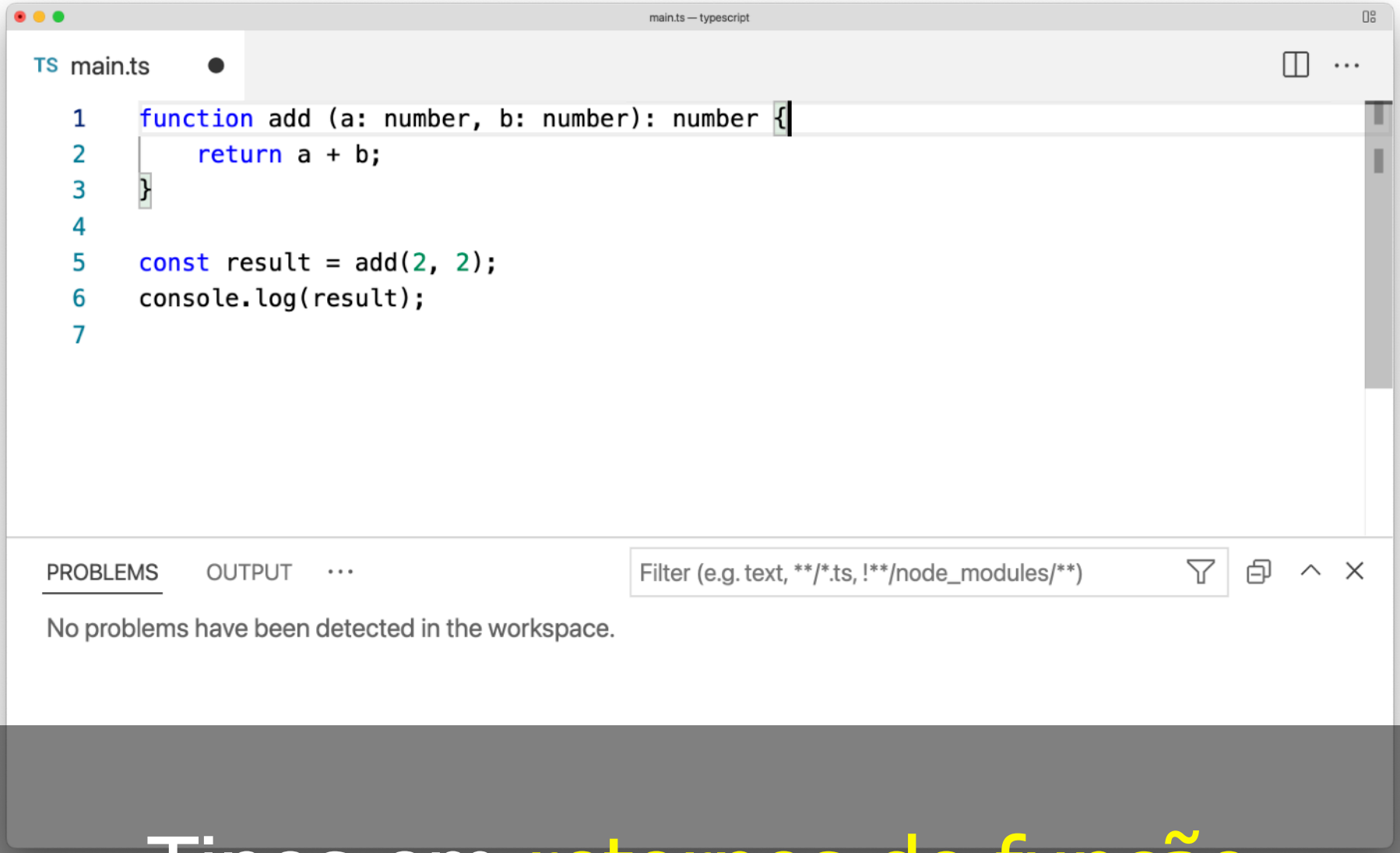
Ao criar uma variável e atribuir um valor o TS infere o tipo automaticamente



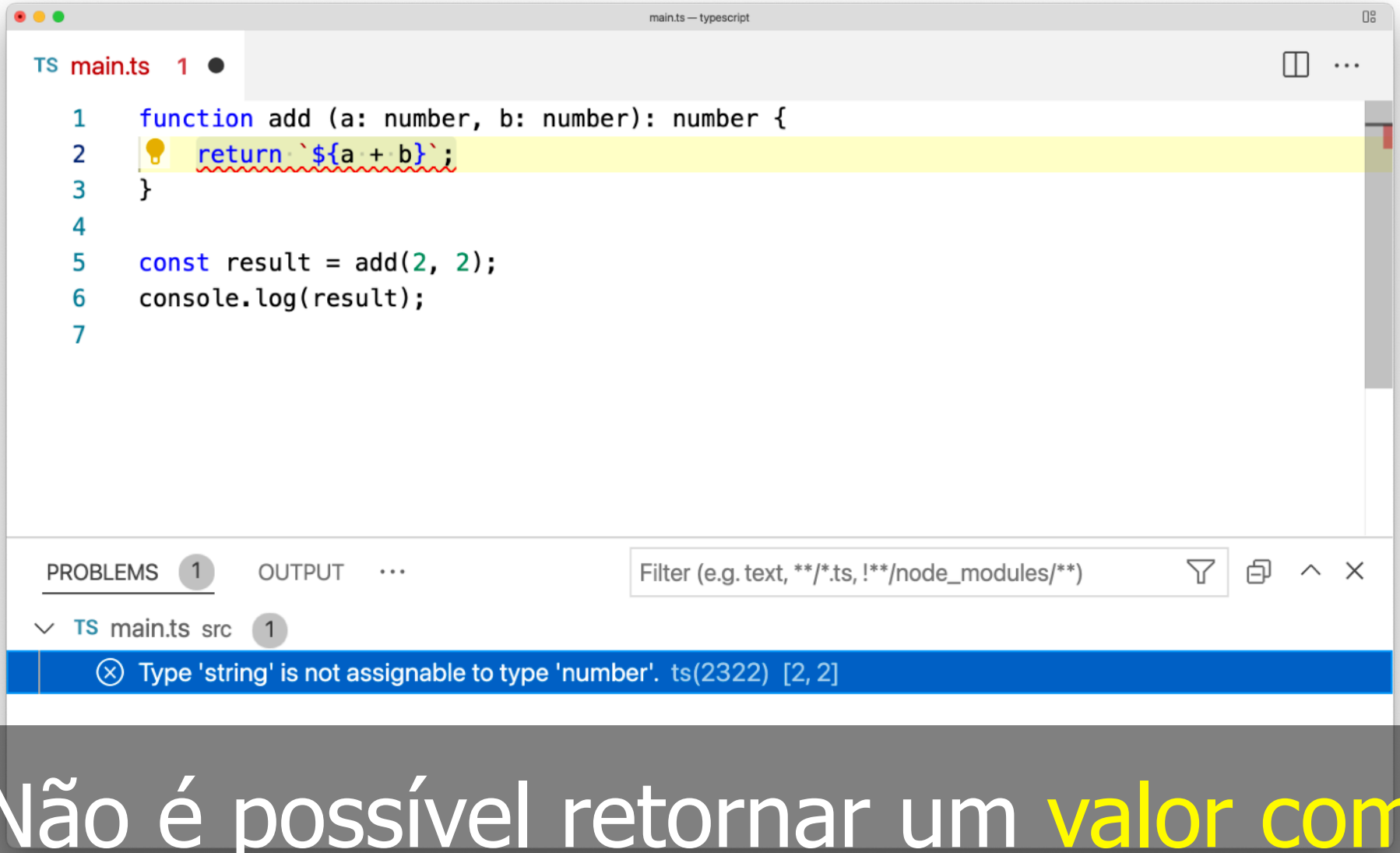
Tipos em parâmetros



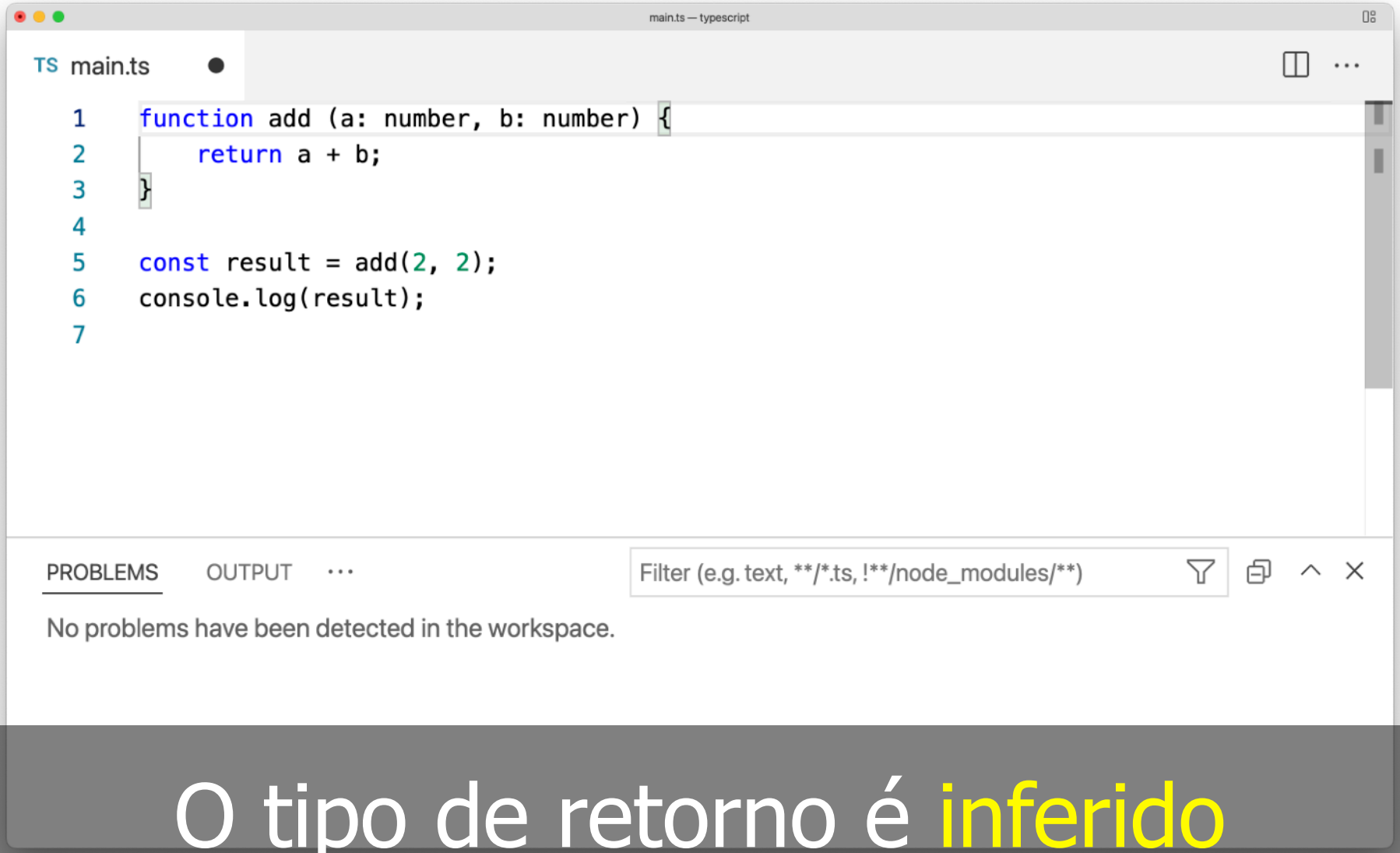
Não é possível passar um parâmetro com o tipo errado ao invocar a função



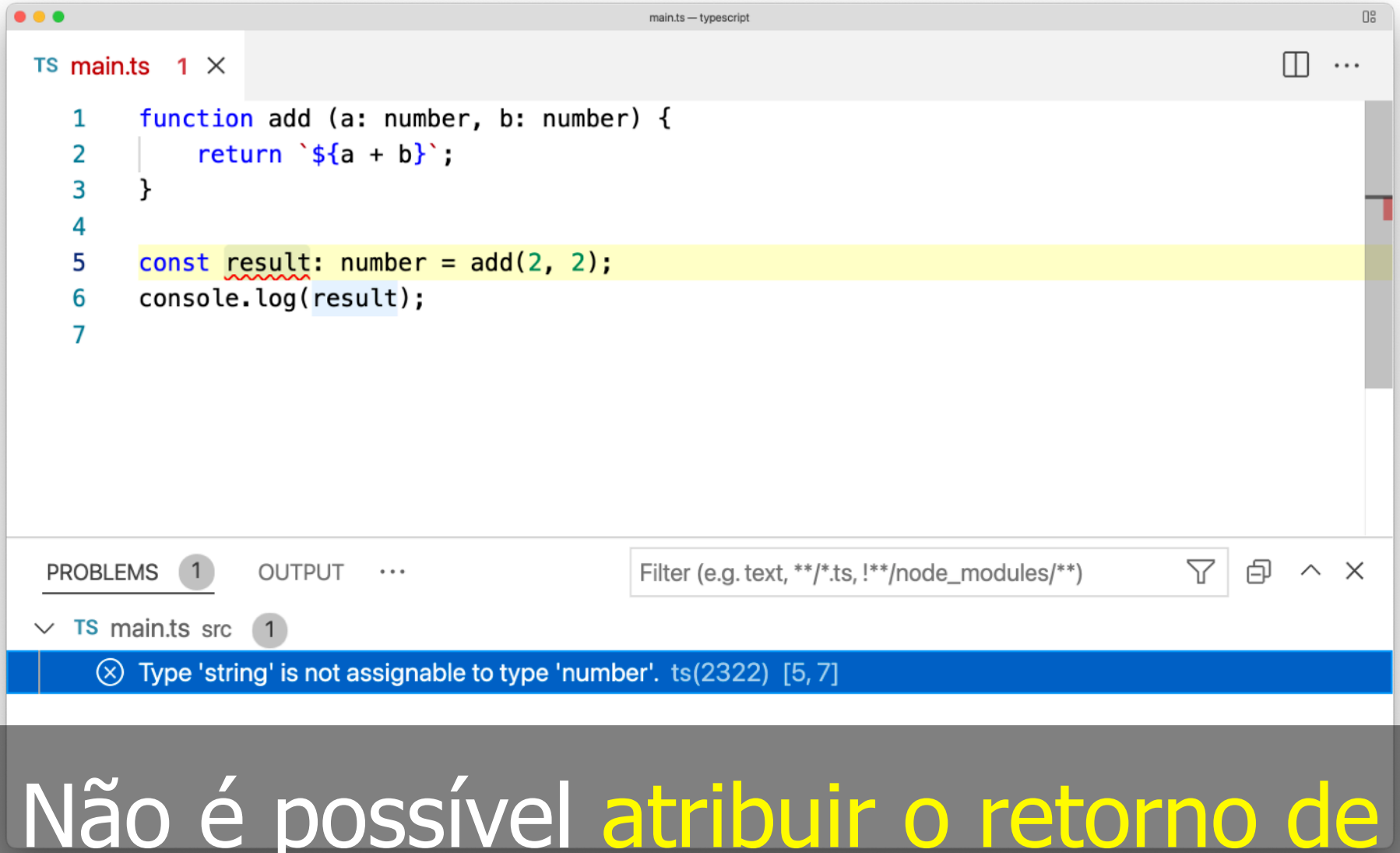
Tipos em retornos de função



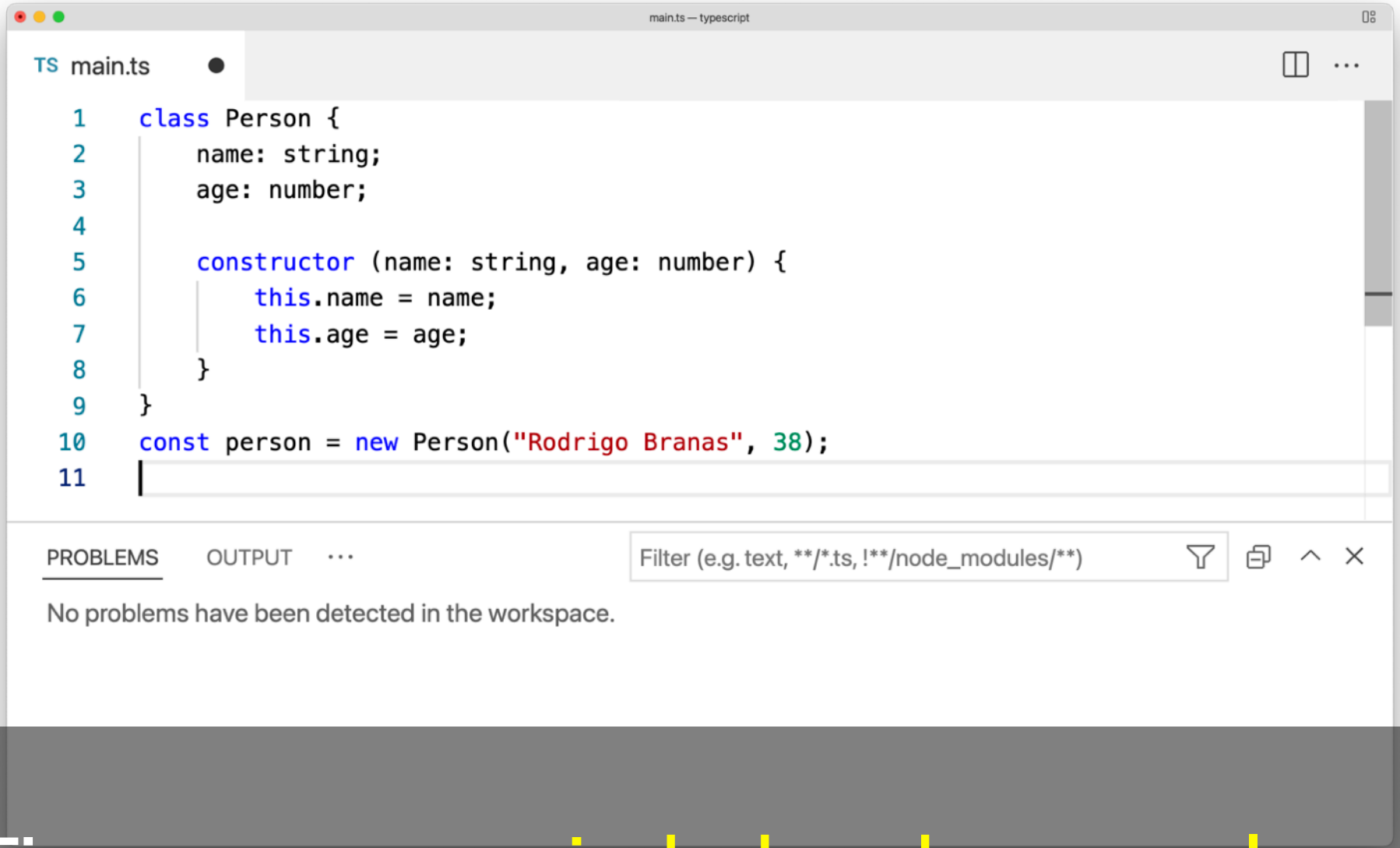
Não é possível retornar um valor com o tipo errado



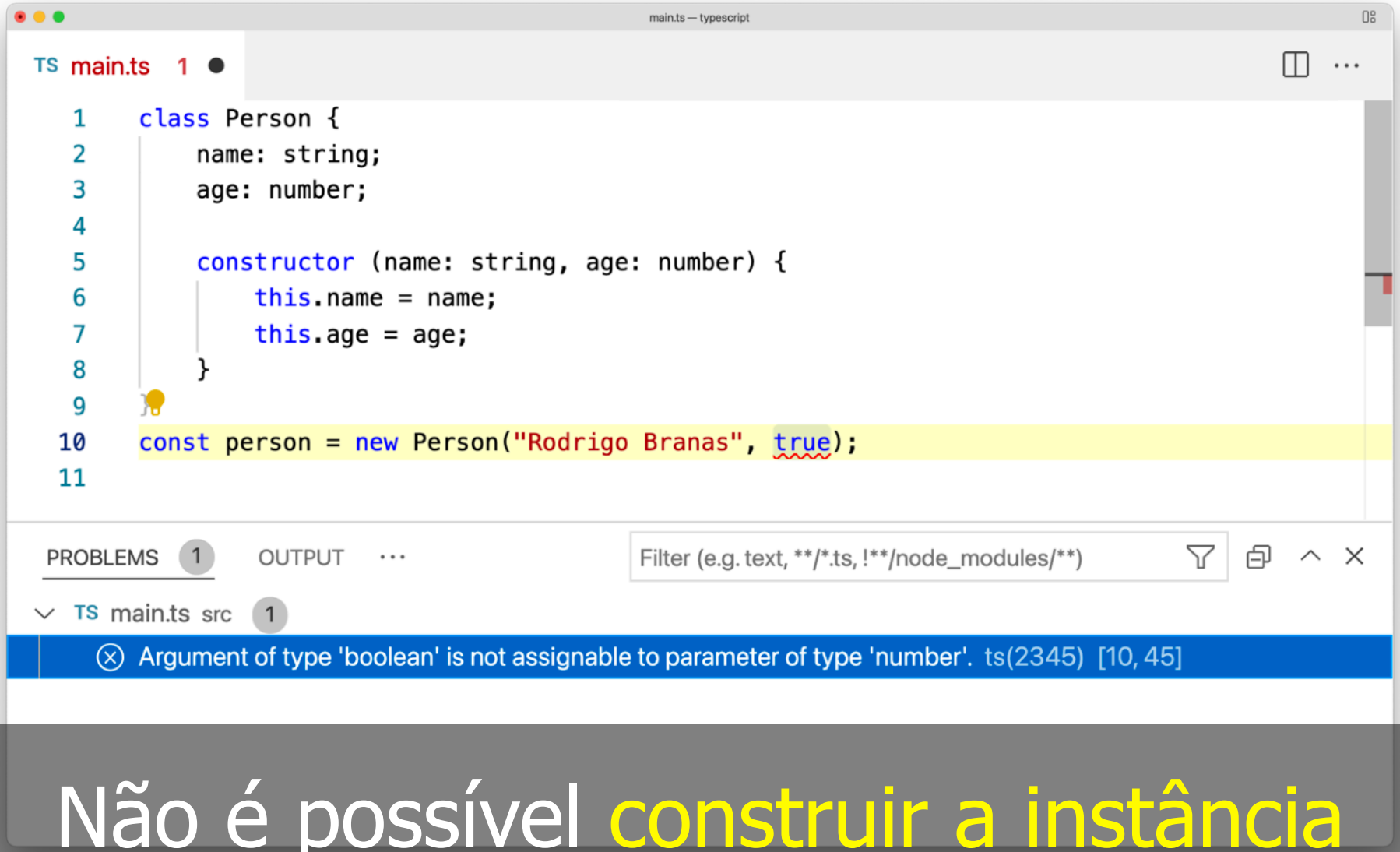
O tipo de retorno é inferido automaticamente



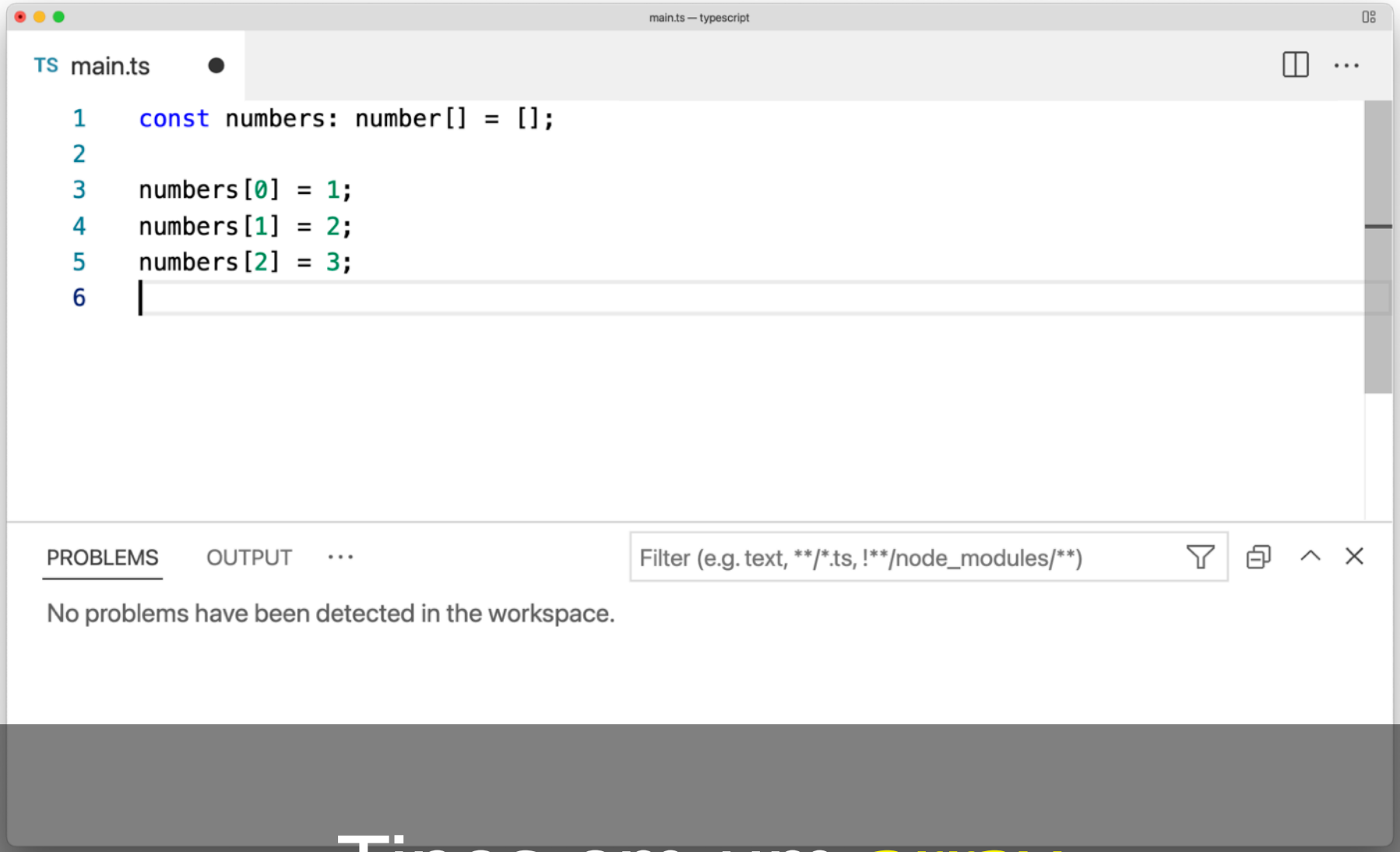
Não é possível atribuir o retorno de uma função ao tipo errado



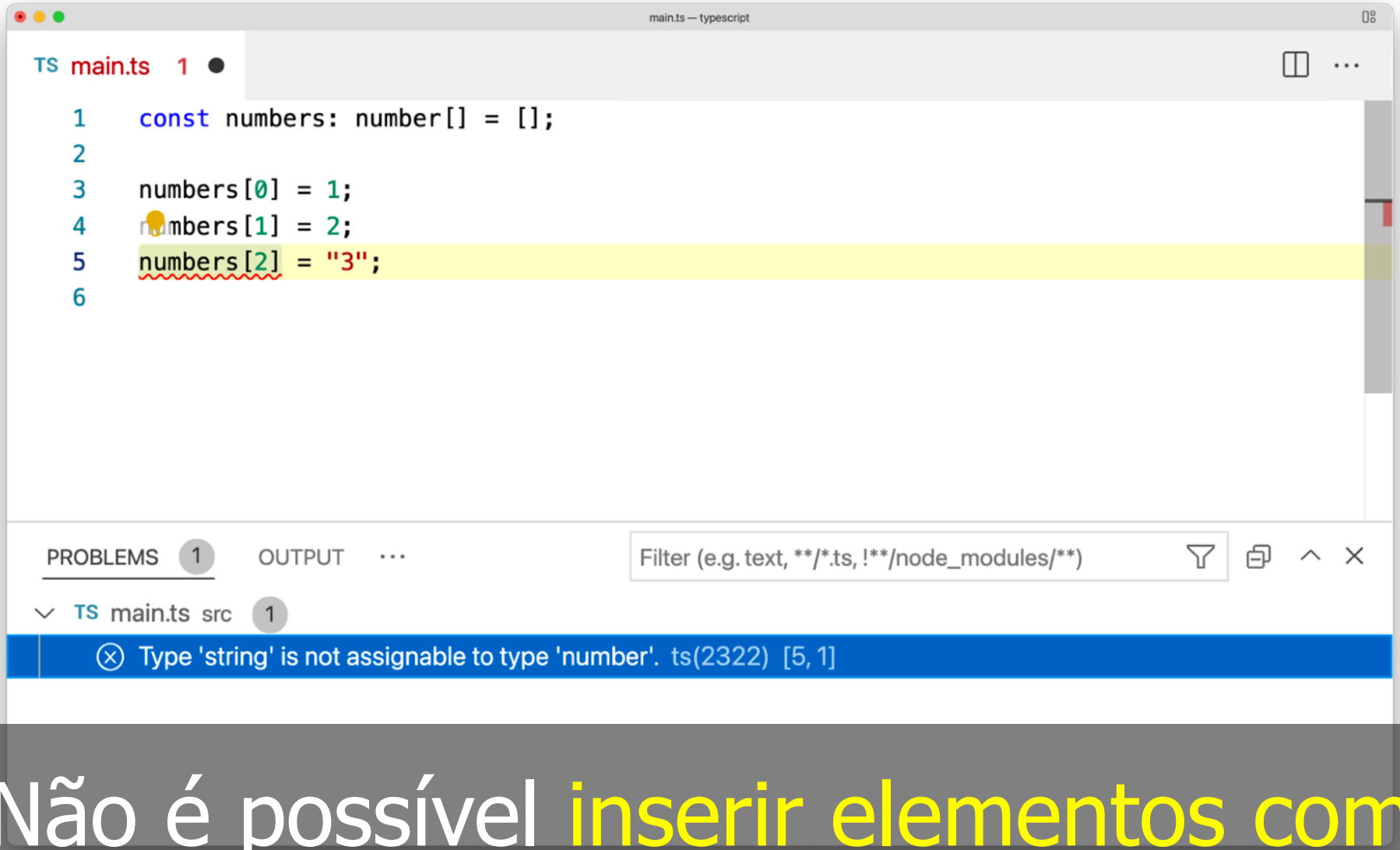
Tipos nas propriedades de uma classe



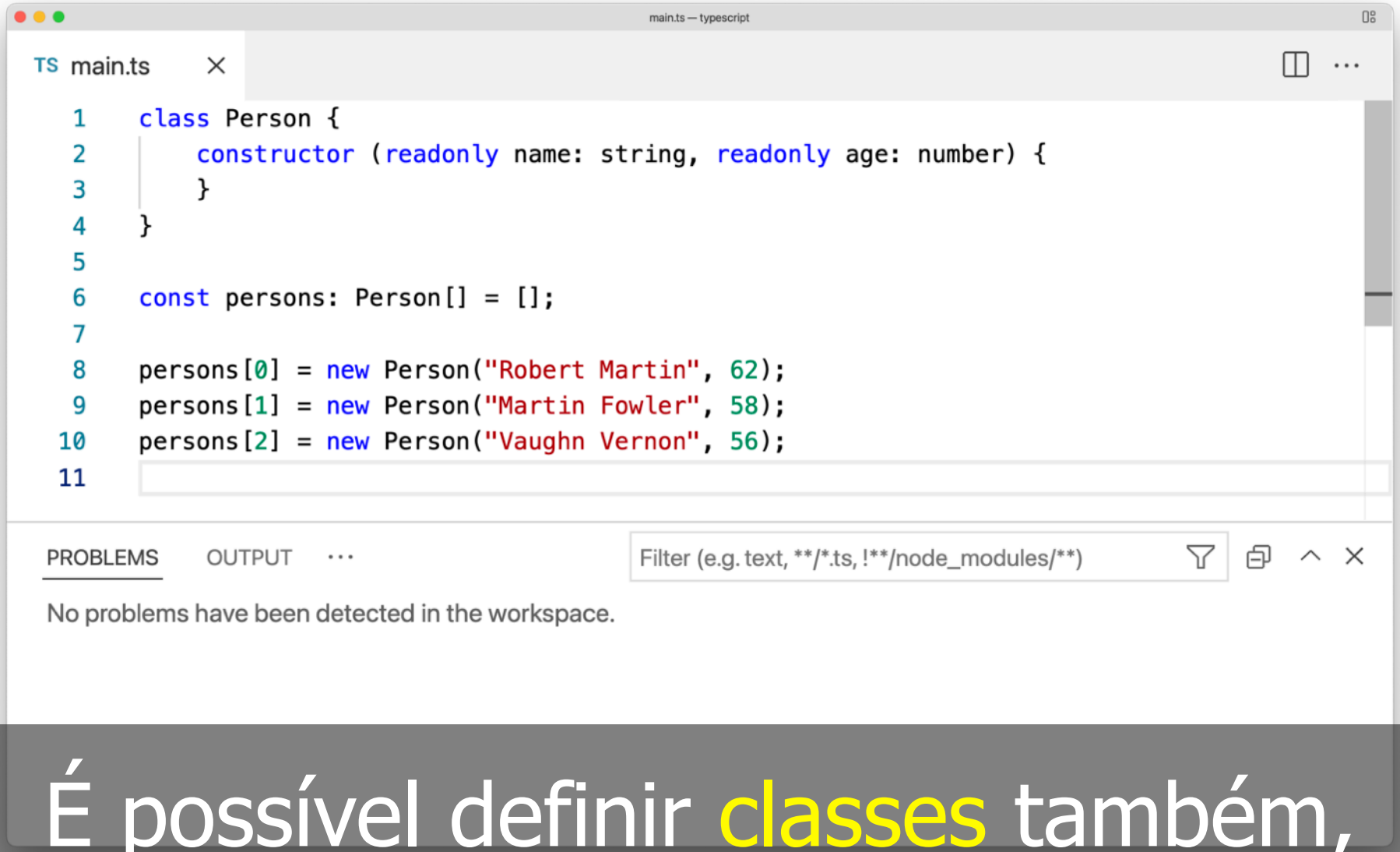
Não é possível construir a instância passando um parâmetro do tipo errado



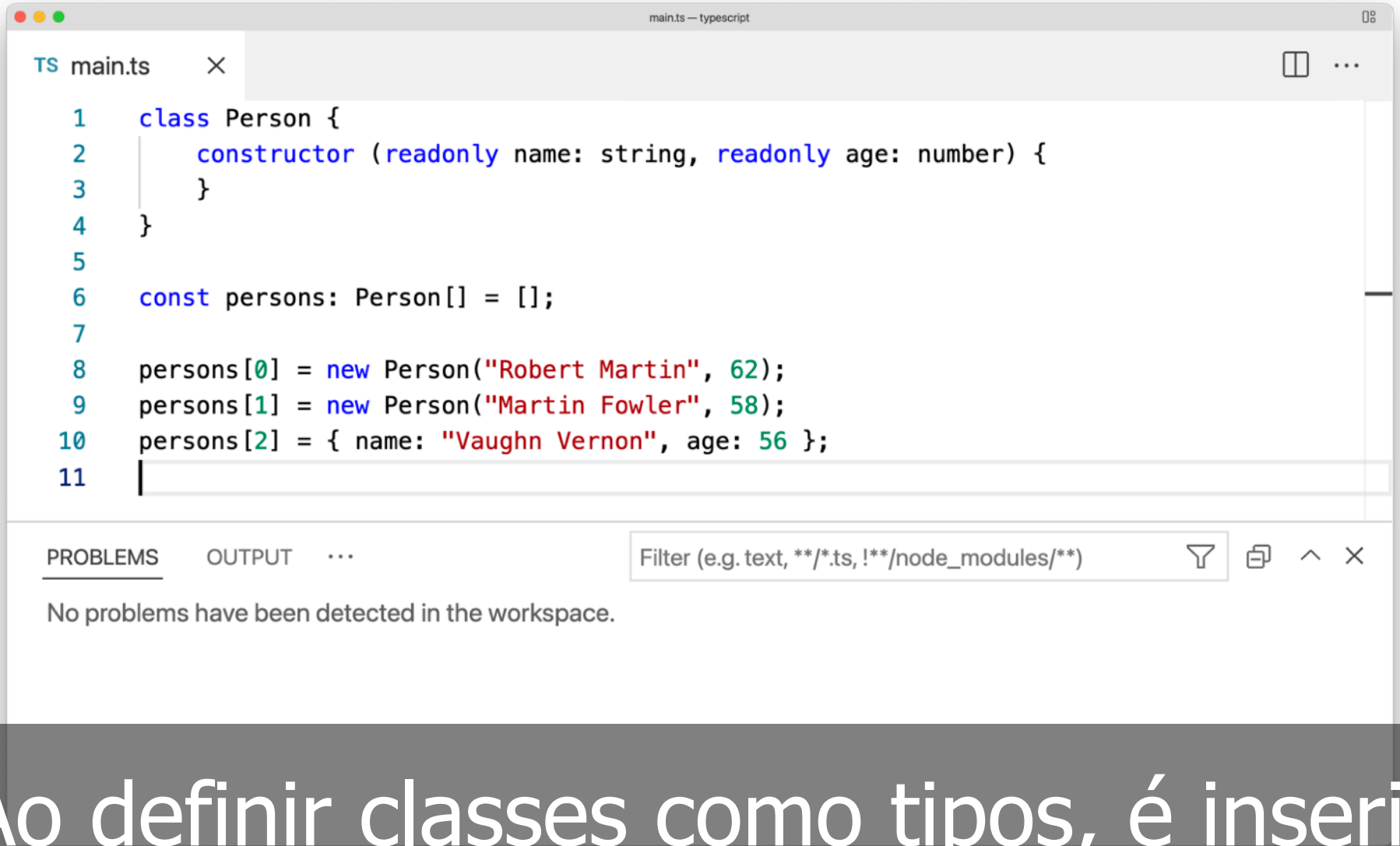
Tipos em um array



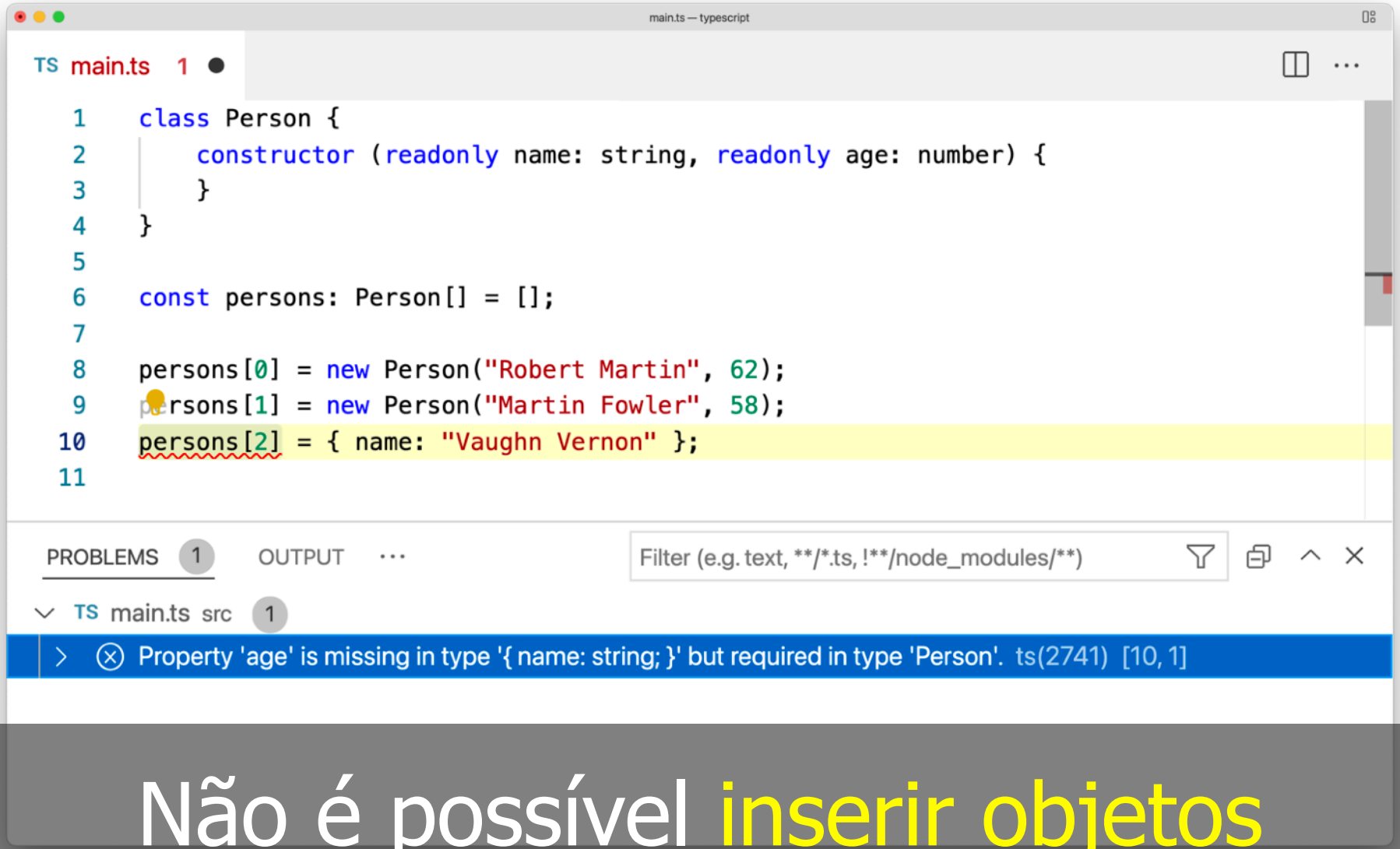
Não é possível inserir elementos com tipos errados



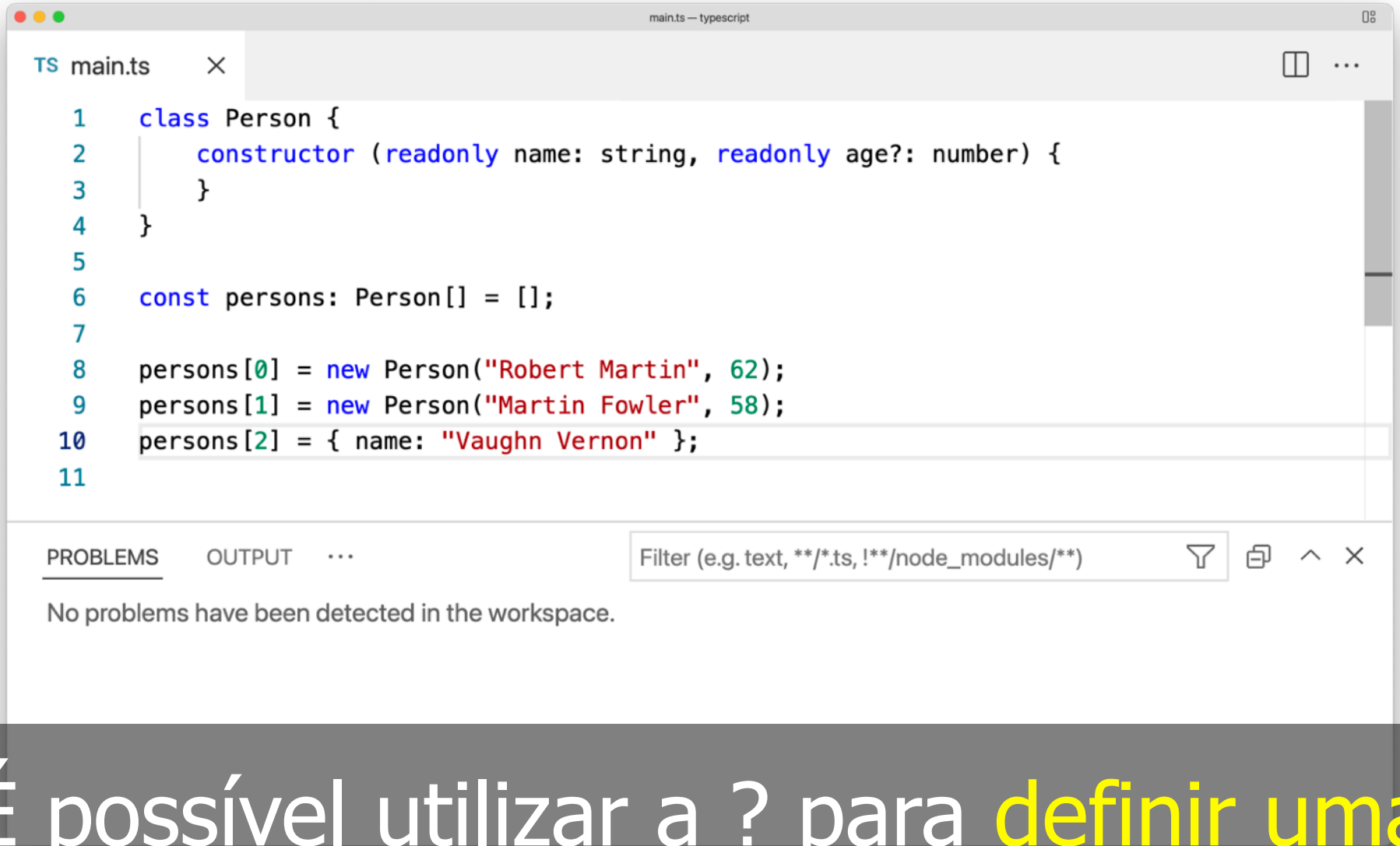
É possível definir classes também, como tipos de um array



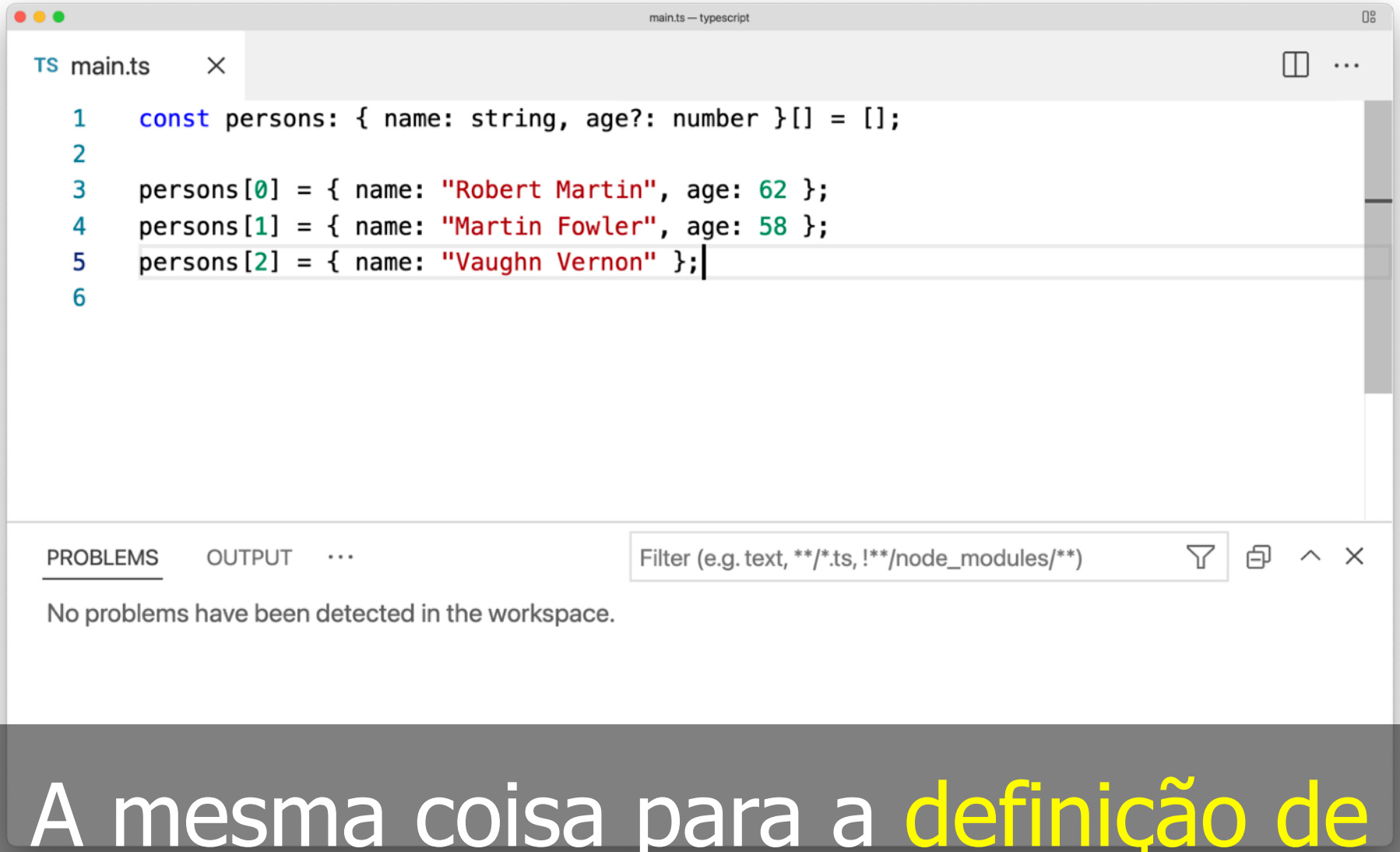
Ao definir classes como tipos, é inserir objetos compatíveis no array



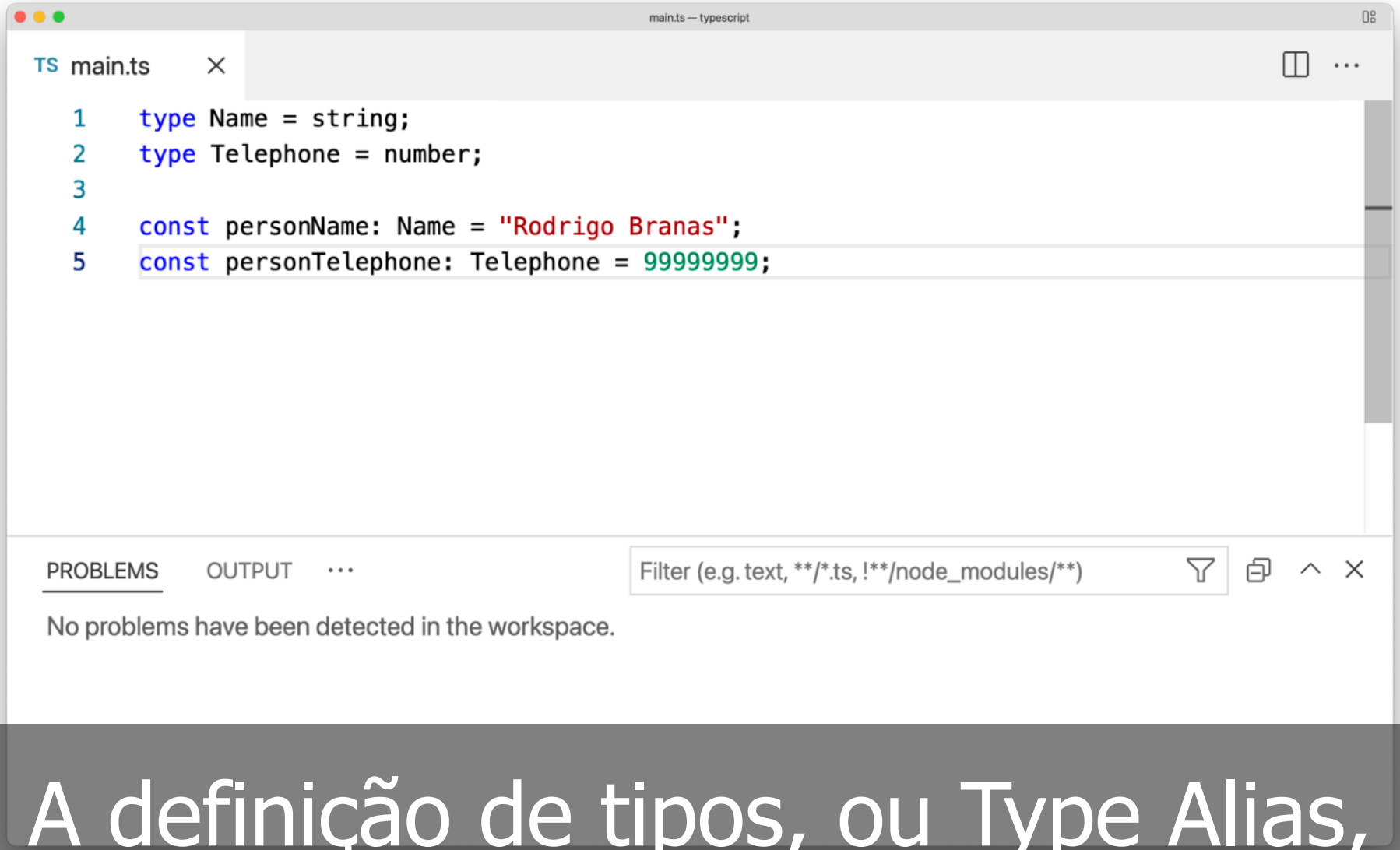
Não é possível inserir objetos incompatíveis com a classe



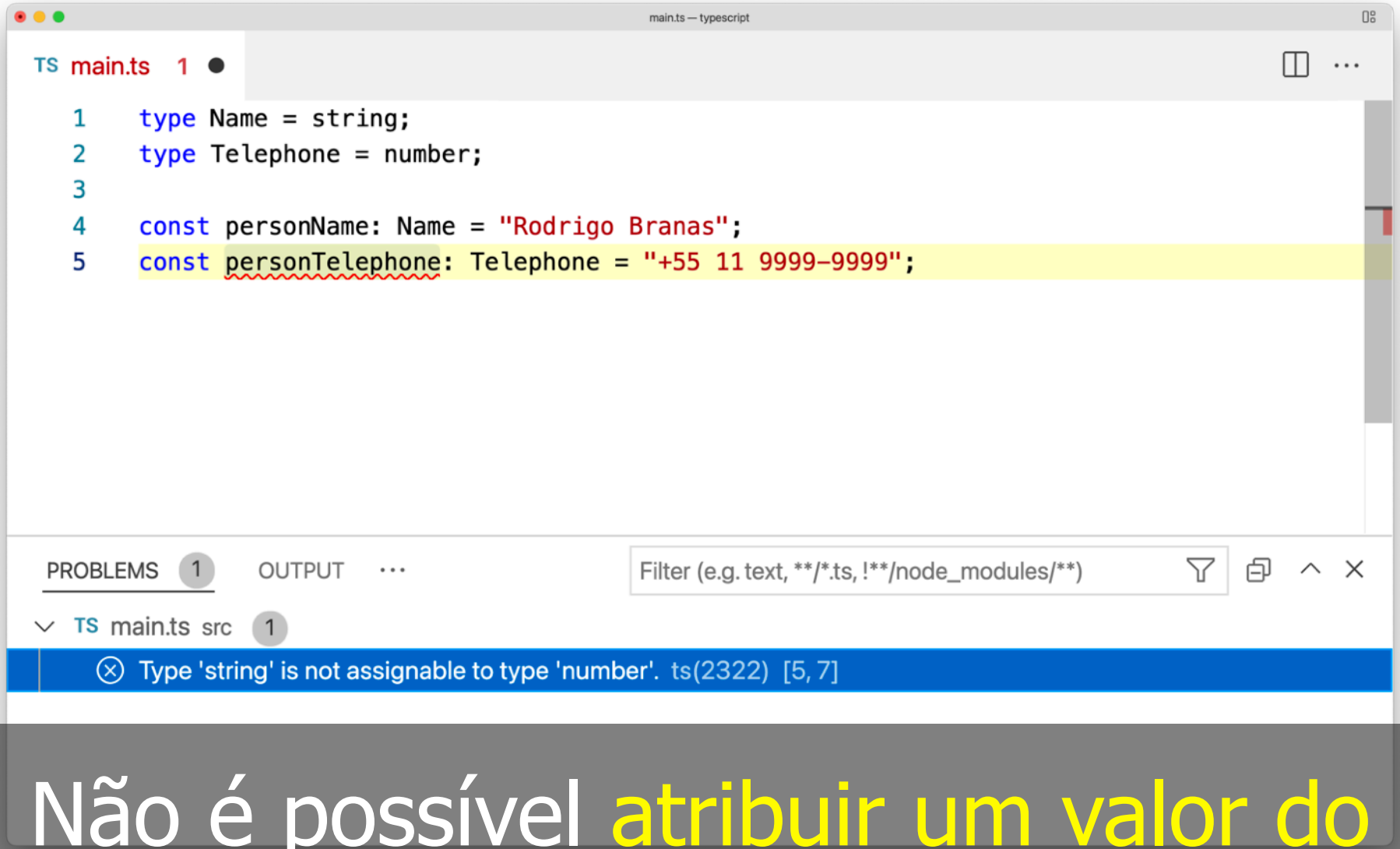
É possível utilizar a ? para definir uma propriedade como opcional



A mesma coisa para a definição de tipos

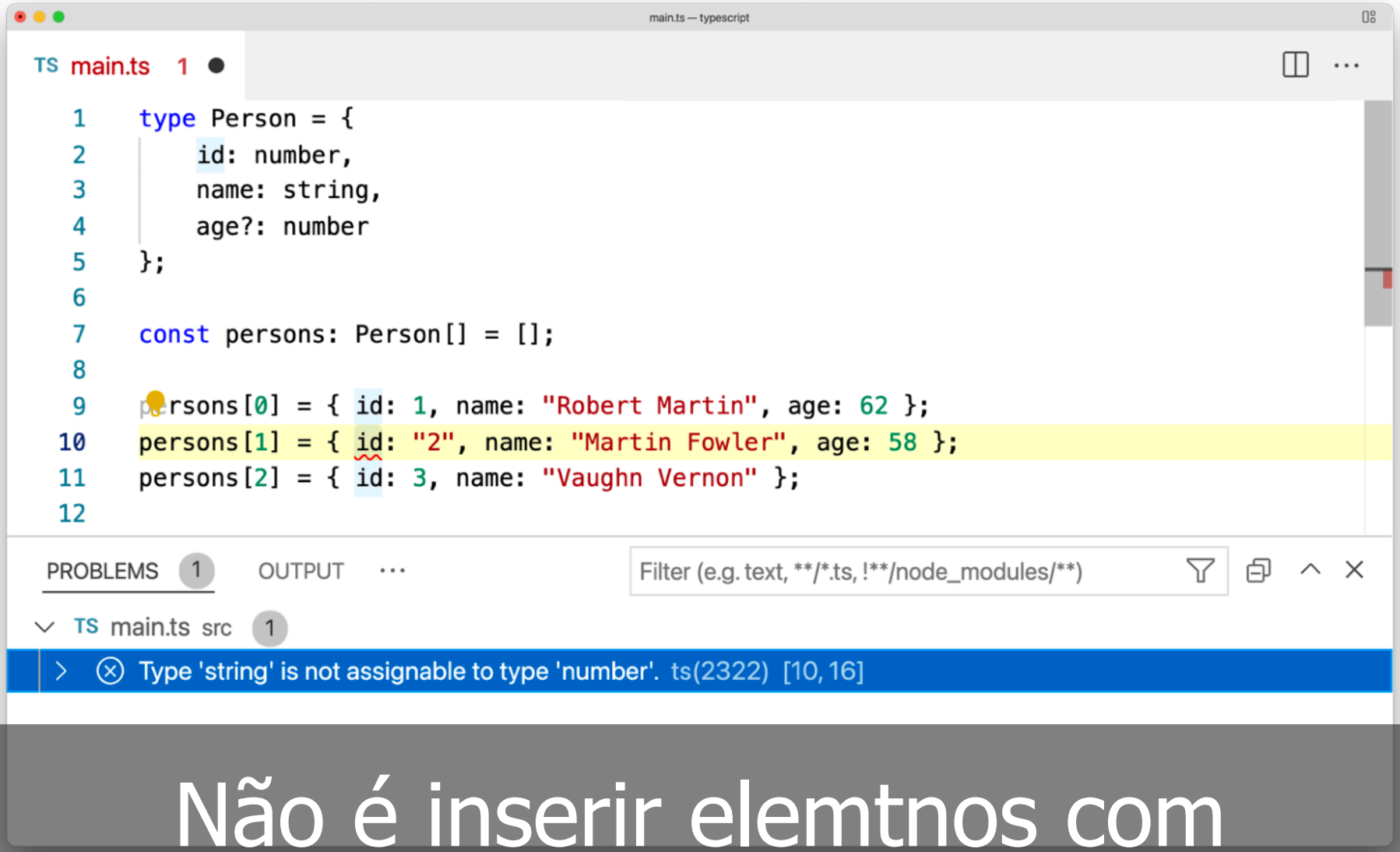


A definição de tipos, ou Type Alias, permite reusá-los

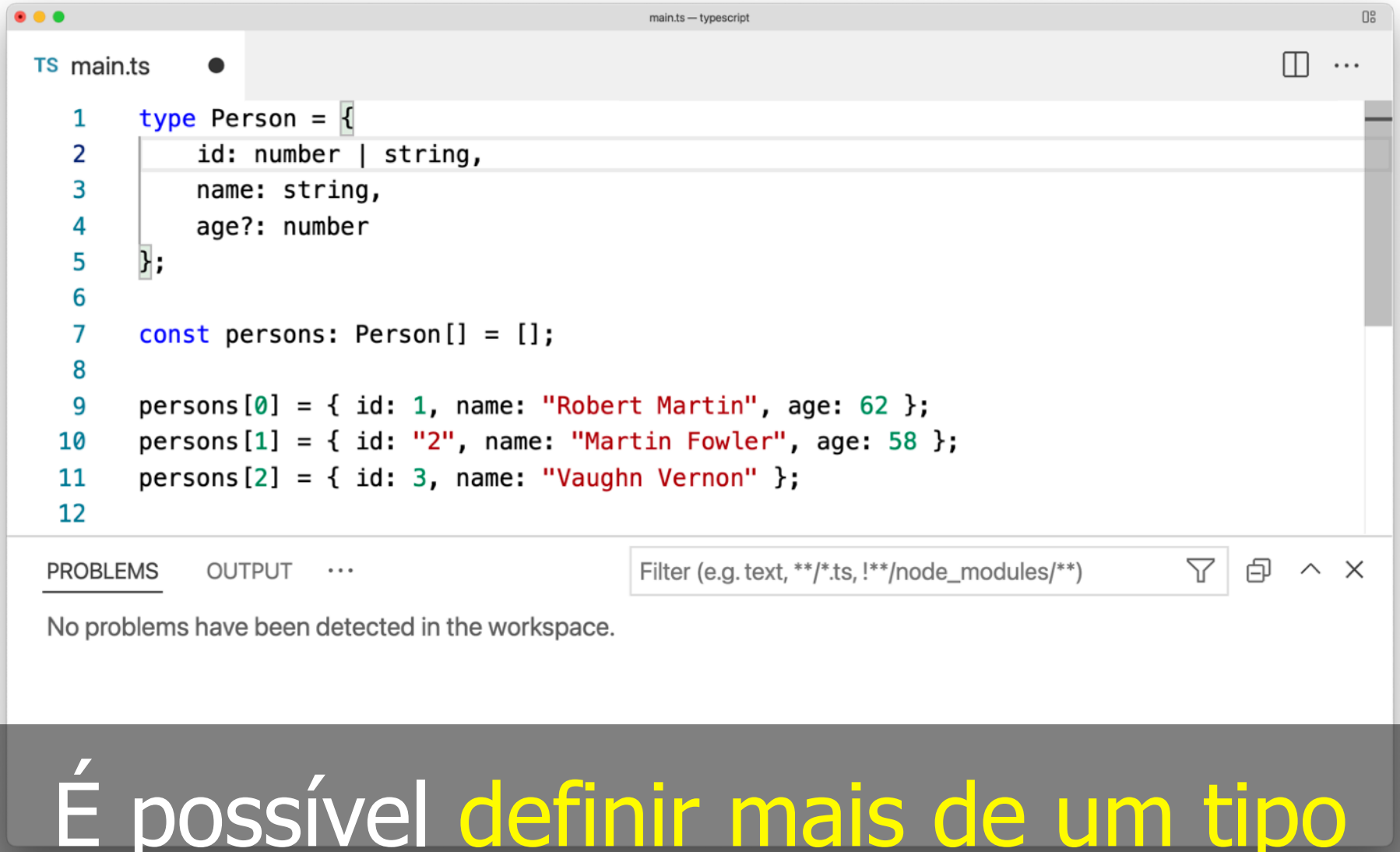


Não é possível atribuir um valor do tipo errado a uma variável tipada

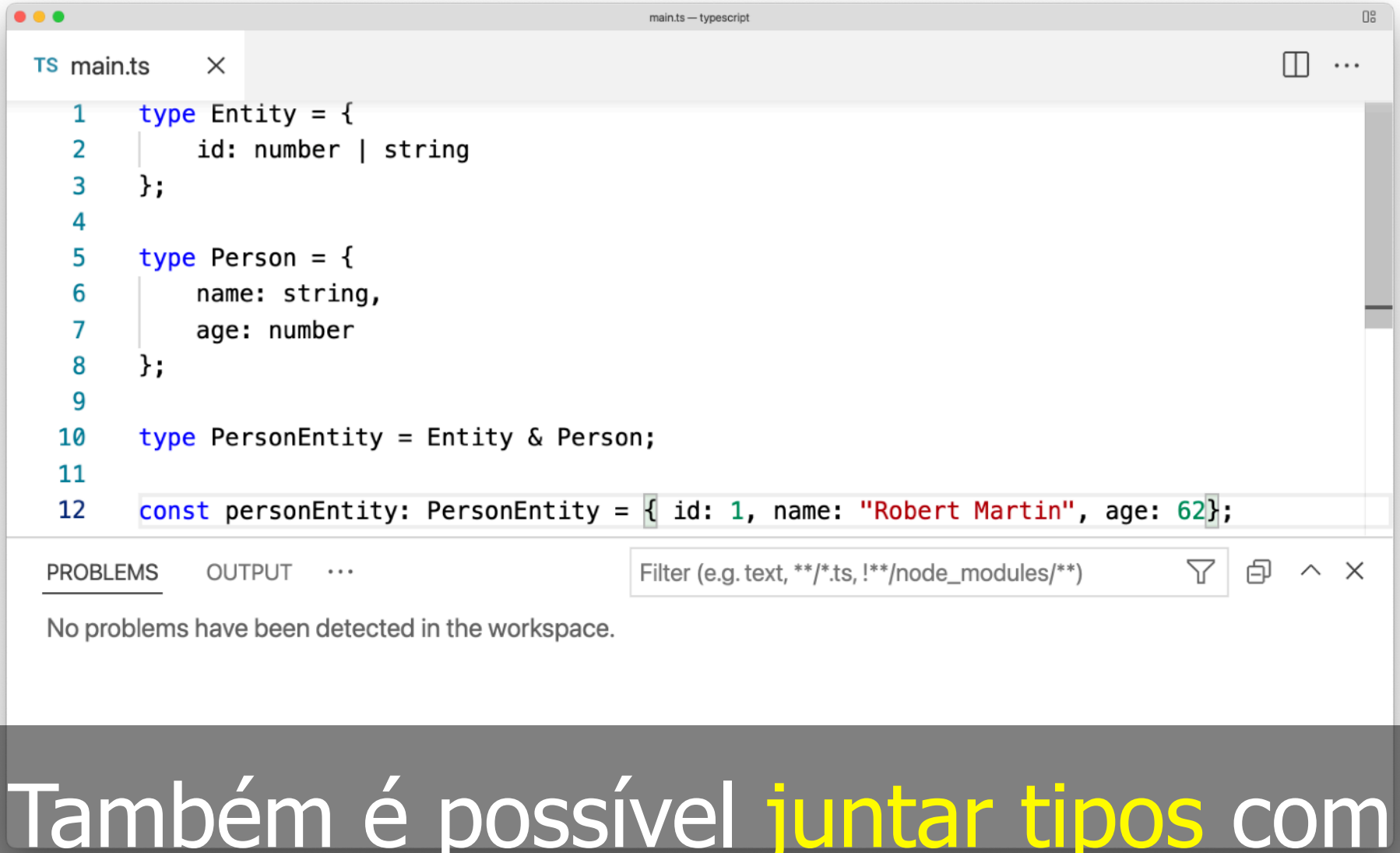
É possível definir propriedades opcionais nos tipos



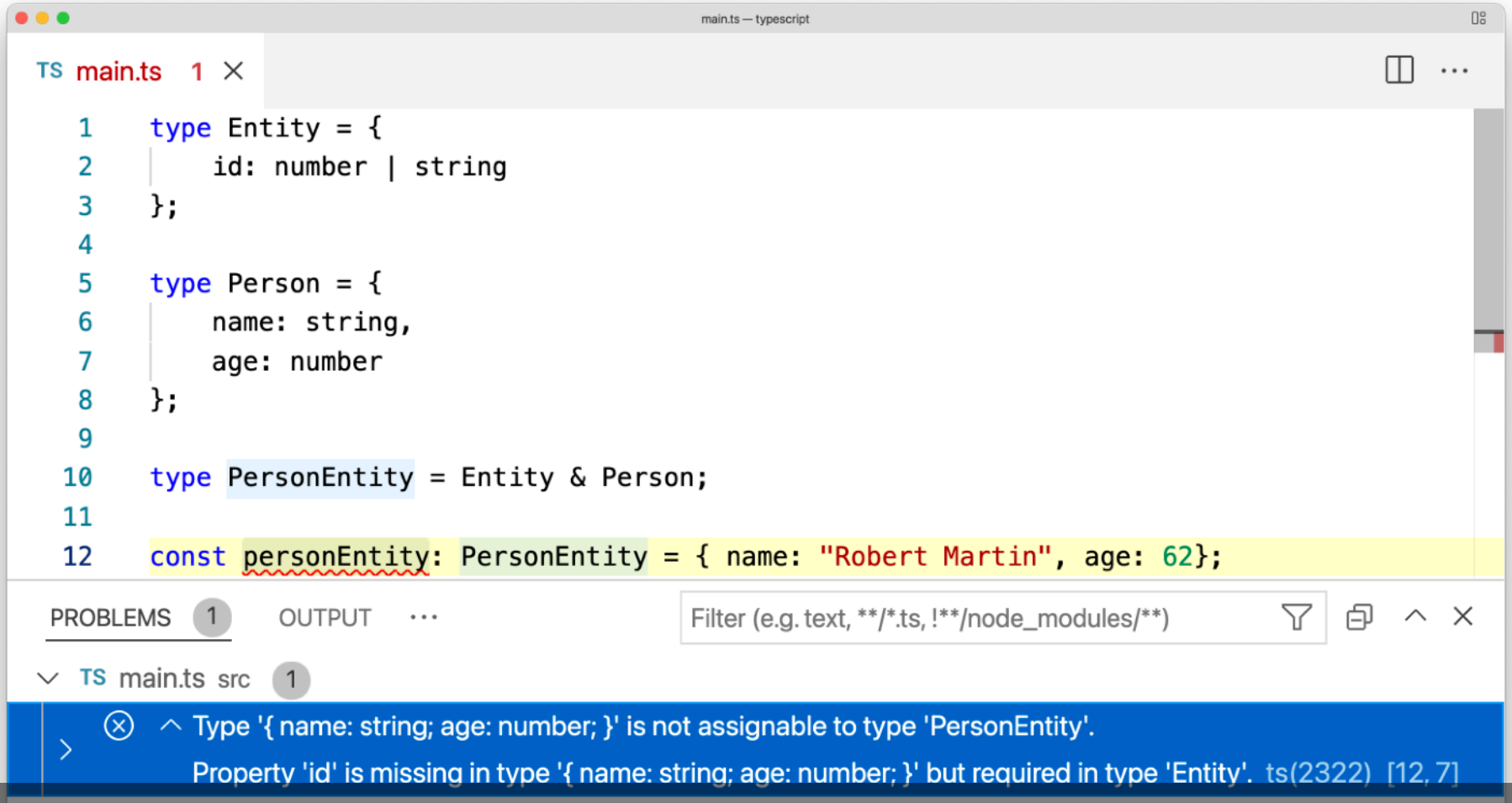
Não é inserir elemtnos com propriedades do tipo errado em arrays



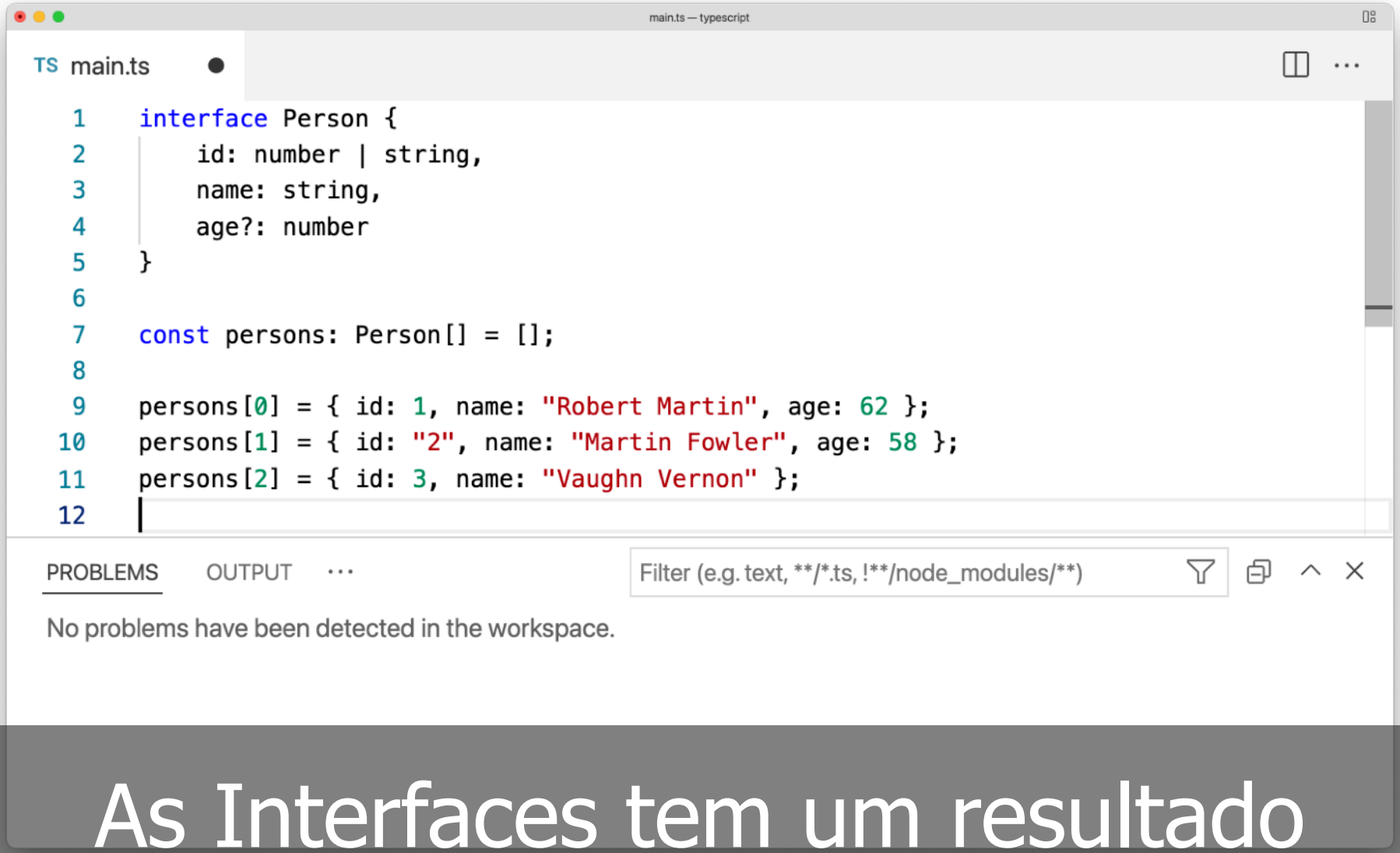
É possível definir mais de um tipo utilizando Union Types



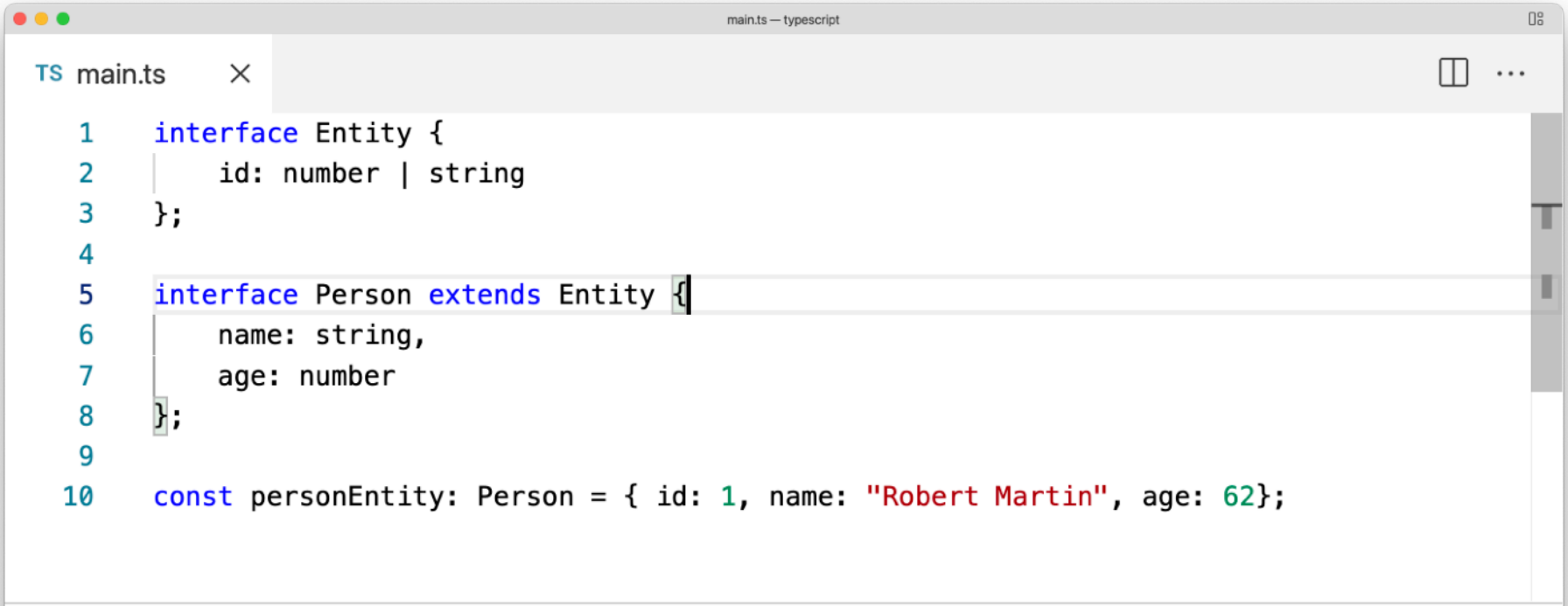
Também é possível juntar tipos com um Merge Type



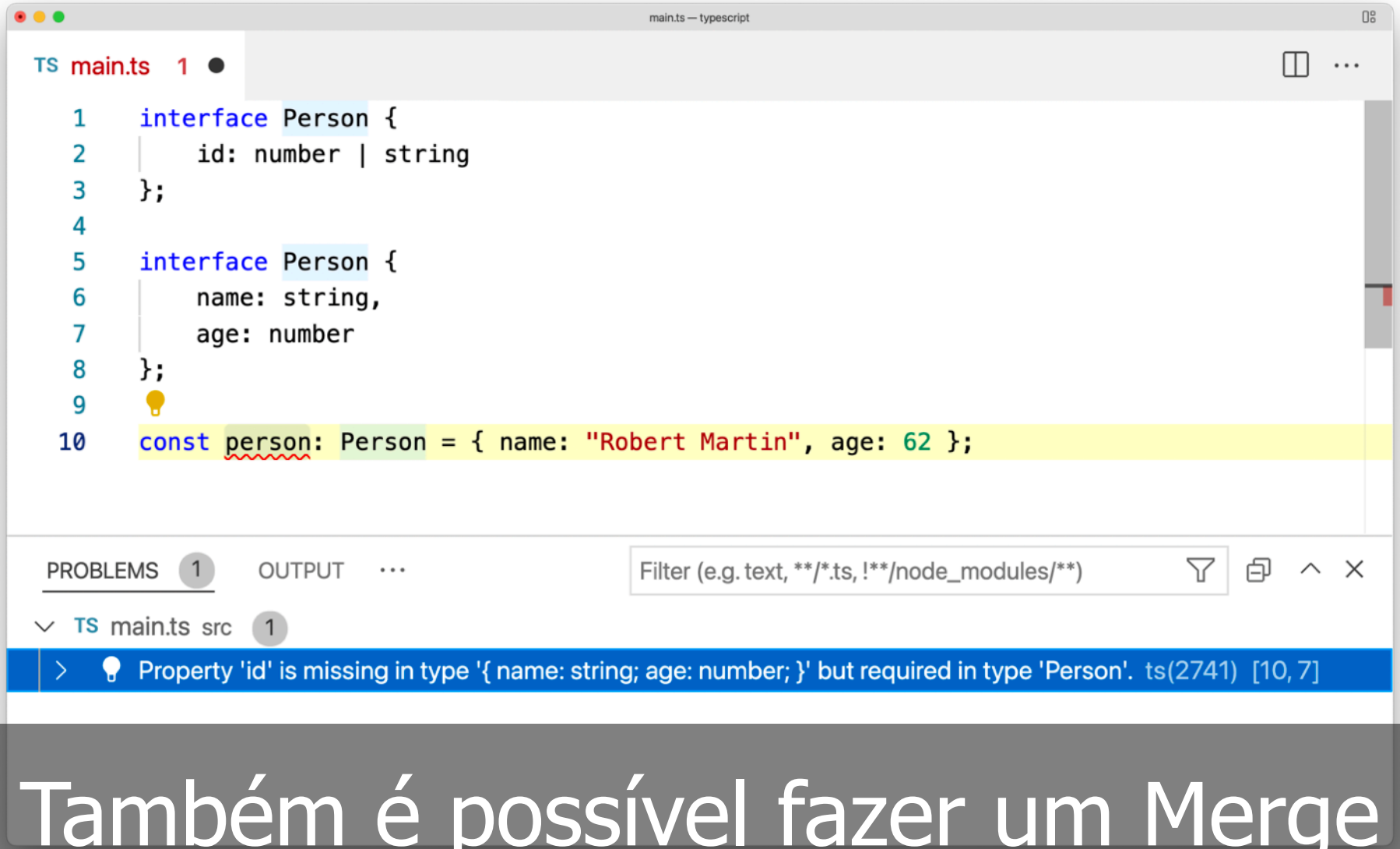
Os Merge Types são verificados e não é possível atribuir um valor errado



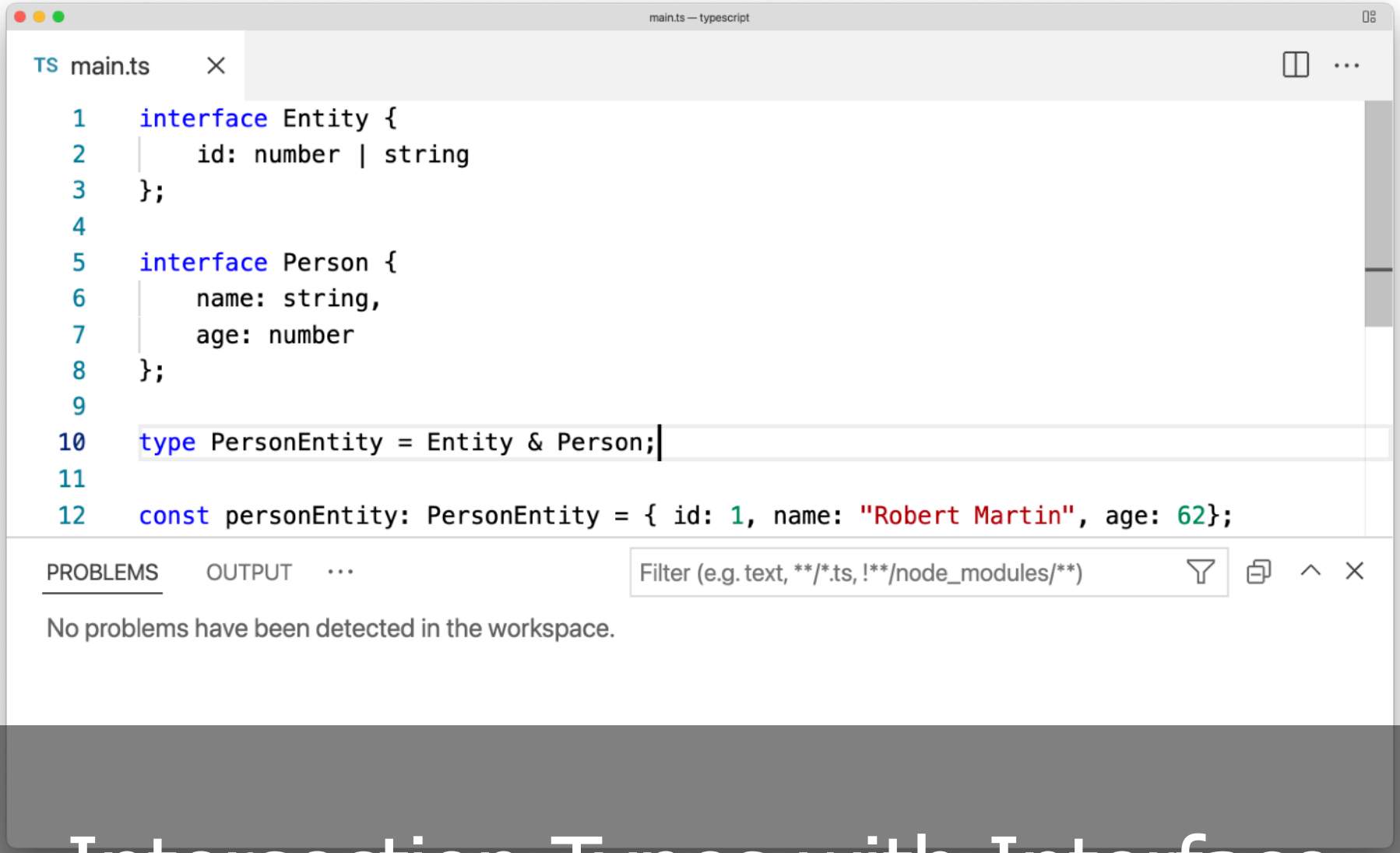
As Interfaces tem um resultado similar aos Types



Uma das diferenças é que é possível extender as Interfaces



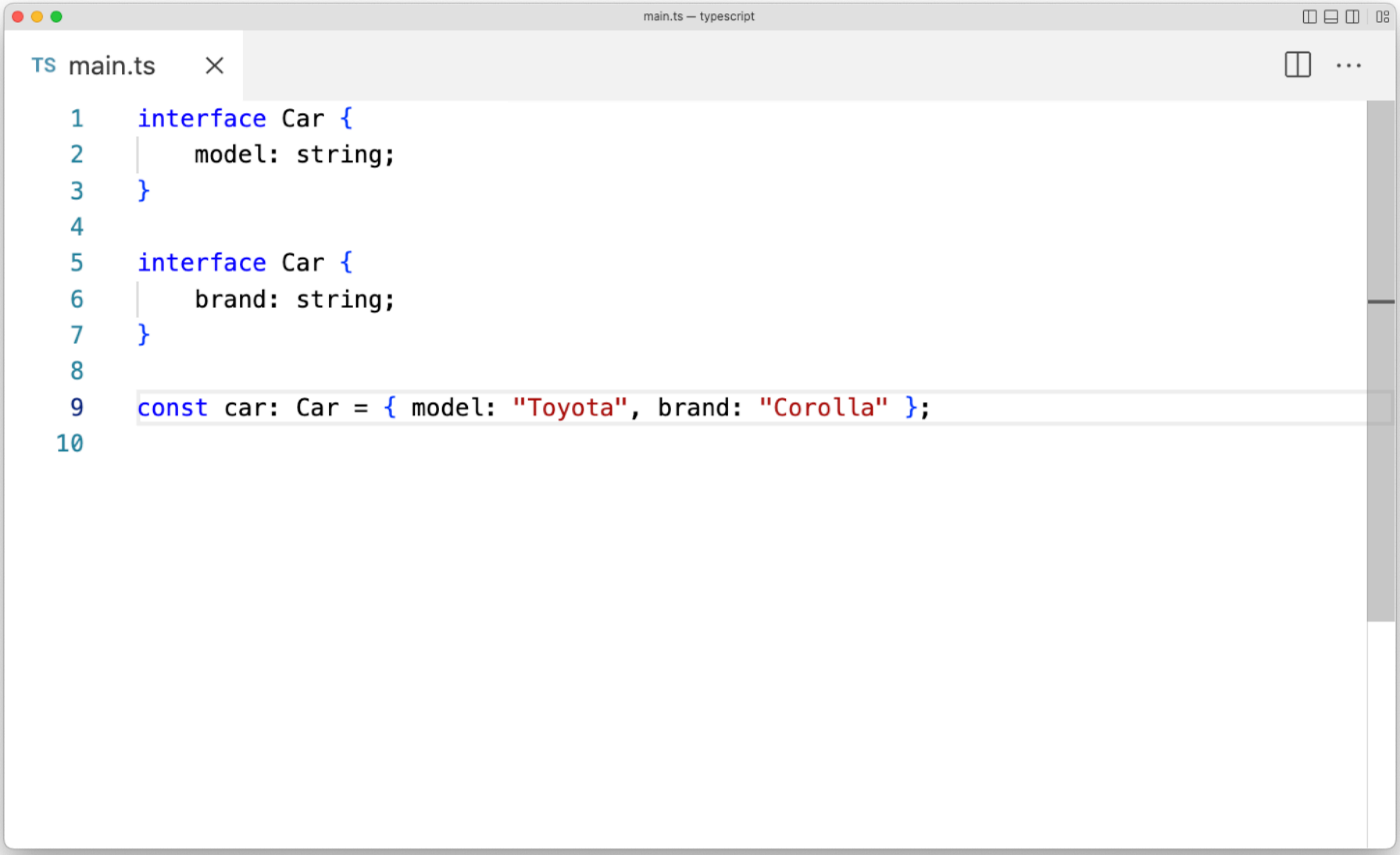
Também é possível fazer um Merge das interfaces

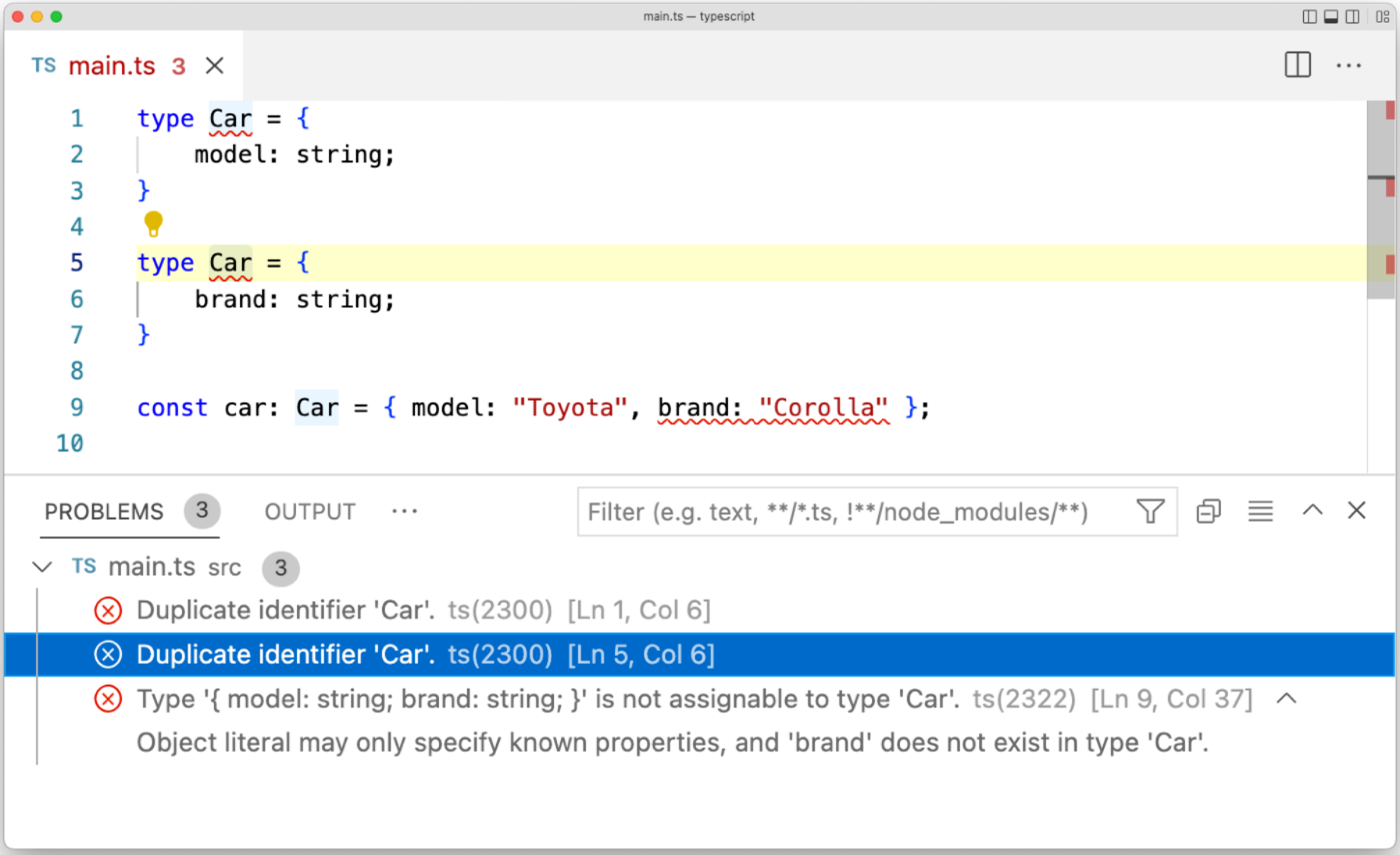


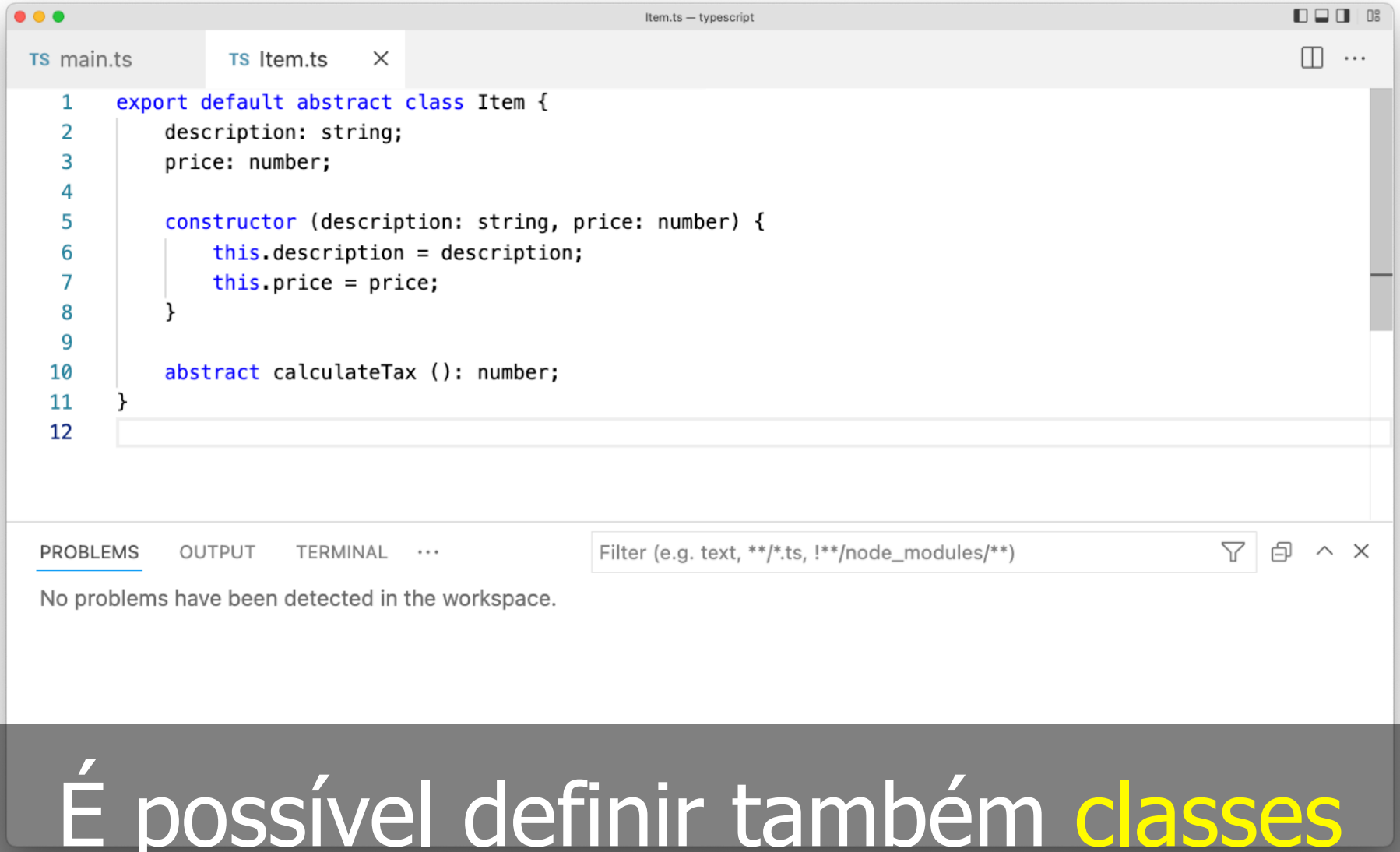
Intersection Types with Interface


Qual é a diferença entre Type Alias e Interface?

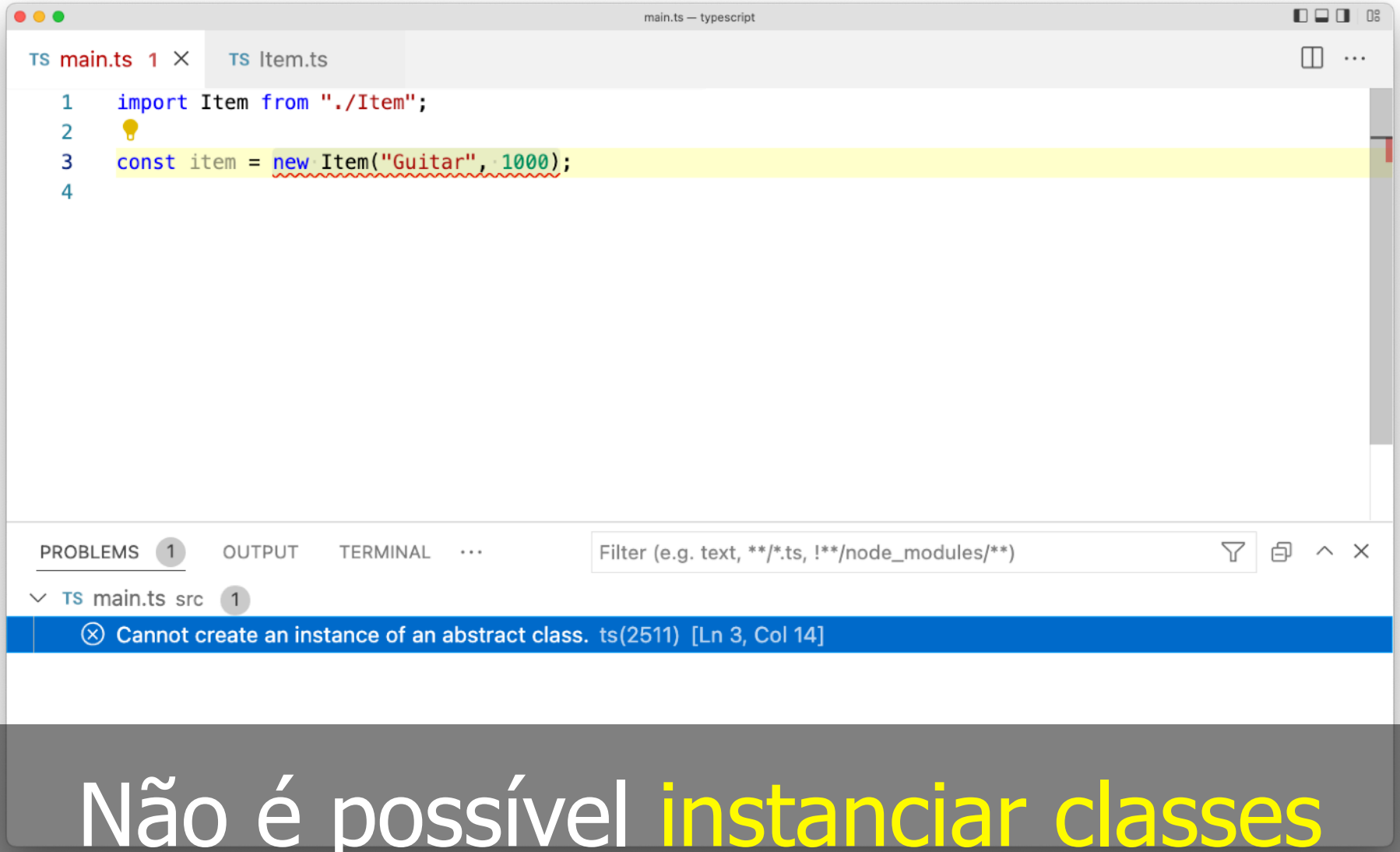
Type aliases and interfaces are very similar, and in many cases you can choose between them freely. Almost all features of an interface are available in type, the key distinction is that a type cannot be re-opened to add new properties vs an interface which is always extendable


main.ts — typescript
TS main.ts
interface Car {
model: string;
}
interface Car {
brand: string;
}
const car: Car = { model: "Toyota", brand: "Corolla" };

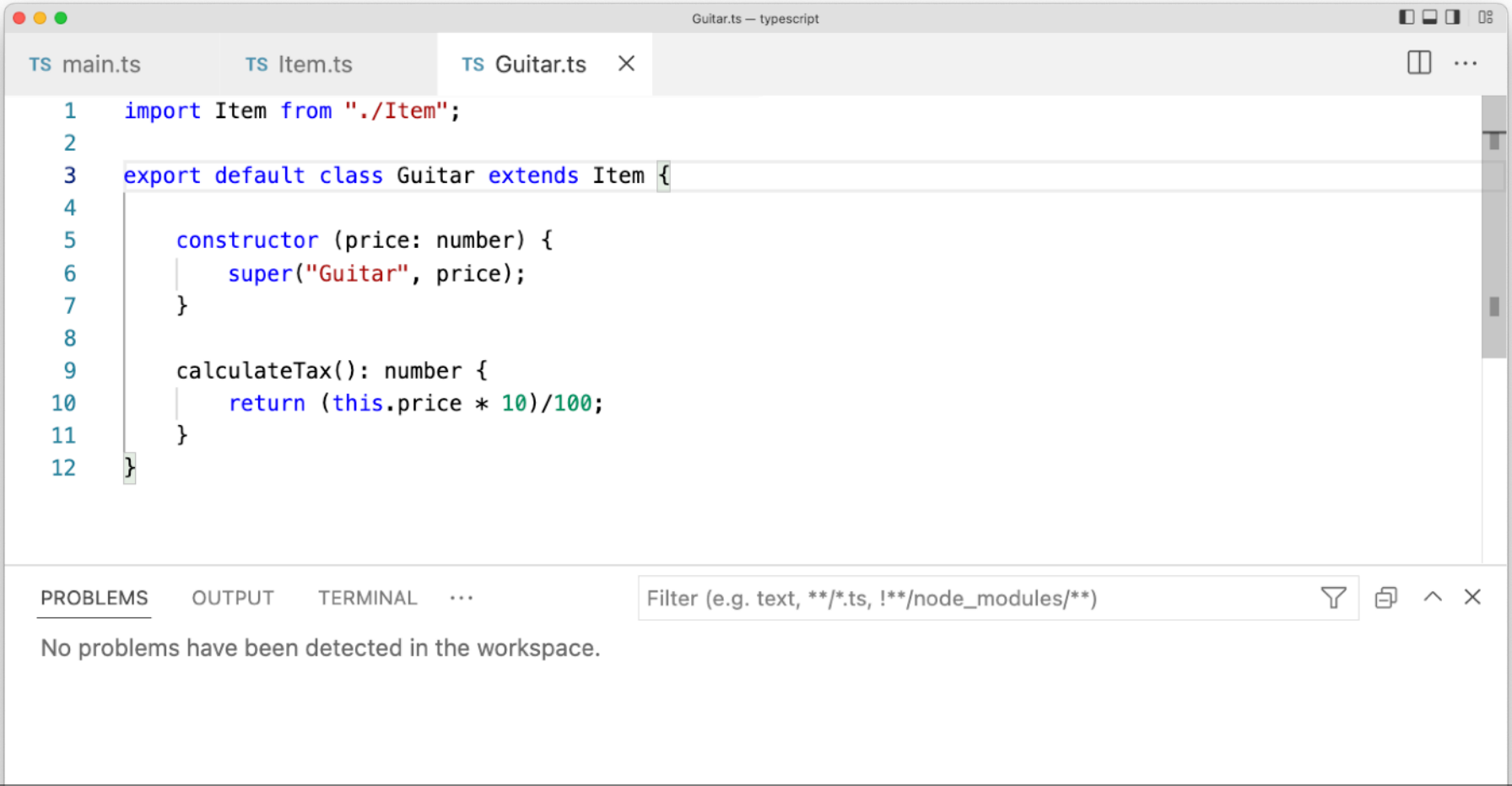

main.ts — typescript
TS main.ts 3
1 type Car = {
2 model: string;
3 }
4
5 type Car = {
6 brand: string;
7 }
8
9 const car: Car = { model: "Toyota", brand: "Corolla" };
10
PROBLEMS 3
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 3
Duplicate identifier 'Car'. ts(2300) [Ln 1, Col 6]
Duplicate identifier 'Car'. ts(2300) [Ln 5, Col 6]
Type '{ model: string; brand: string; }' is not assignable to type 'Car'. ts(2322) [Ln 9, Col 37]
Object literal may only specify known properties, and 'brand' does not exist in type 'Car'.



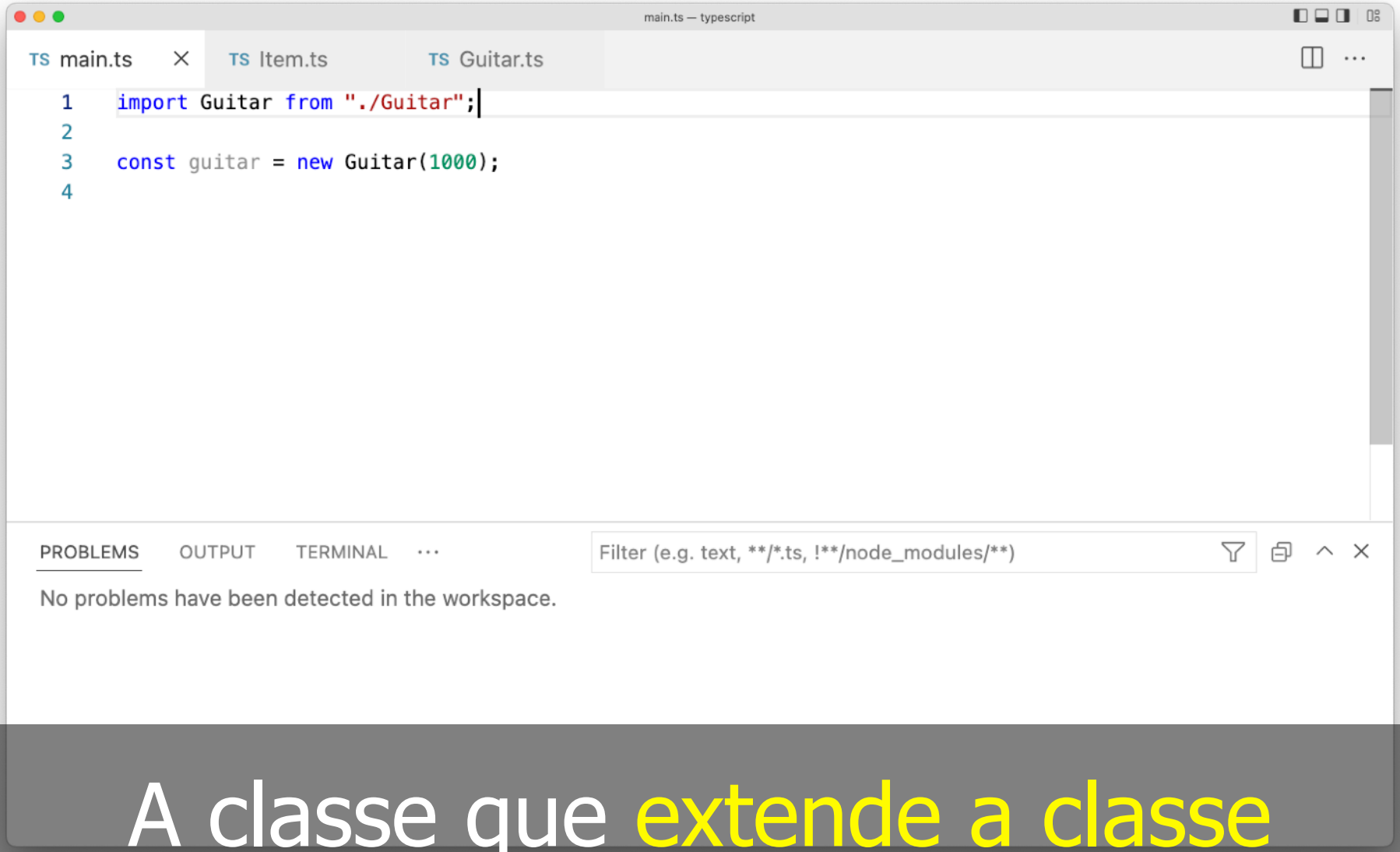
É possível definir também classes abstratas



Não é possível instanciar classes abstratas

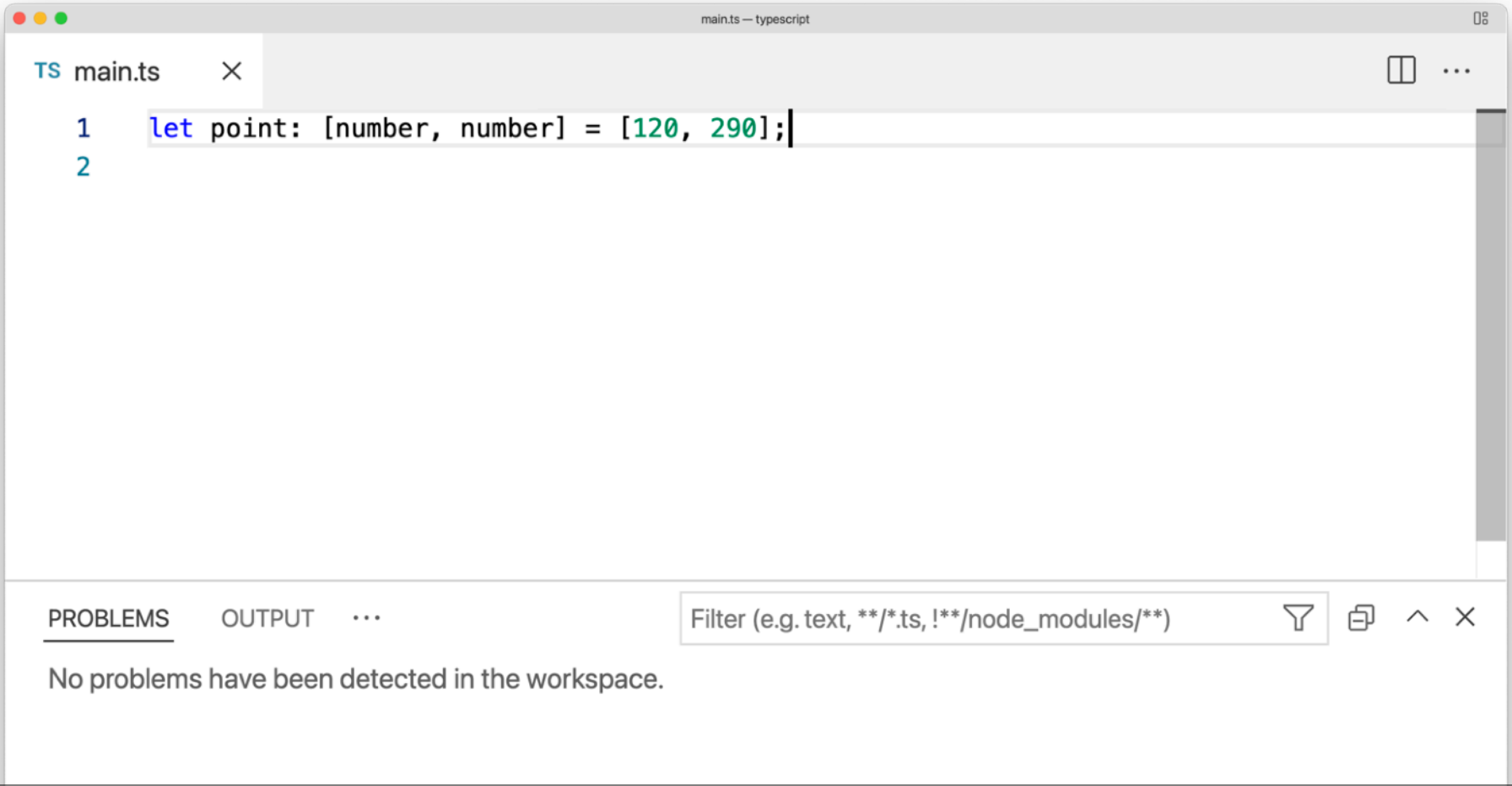


Ao extender uma classe abstrata devemos implementar os métodos abstratos

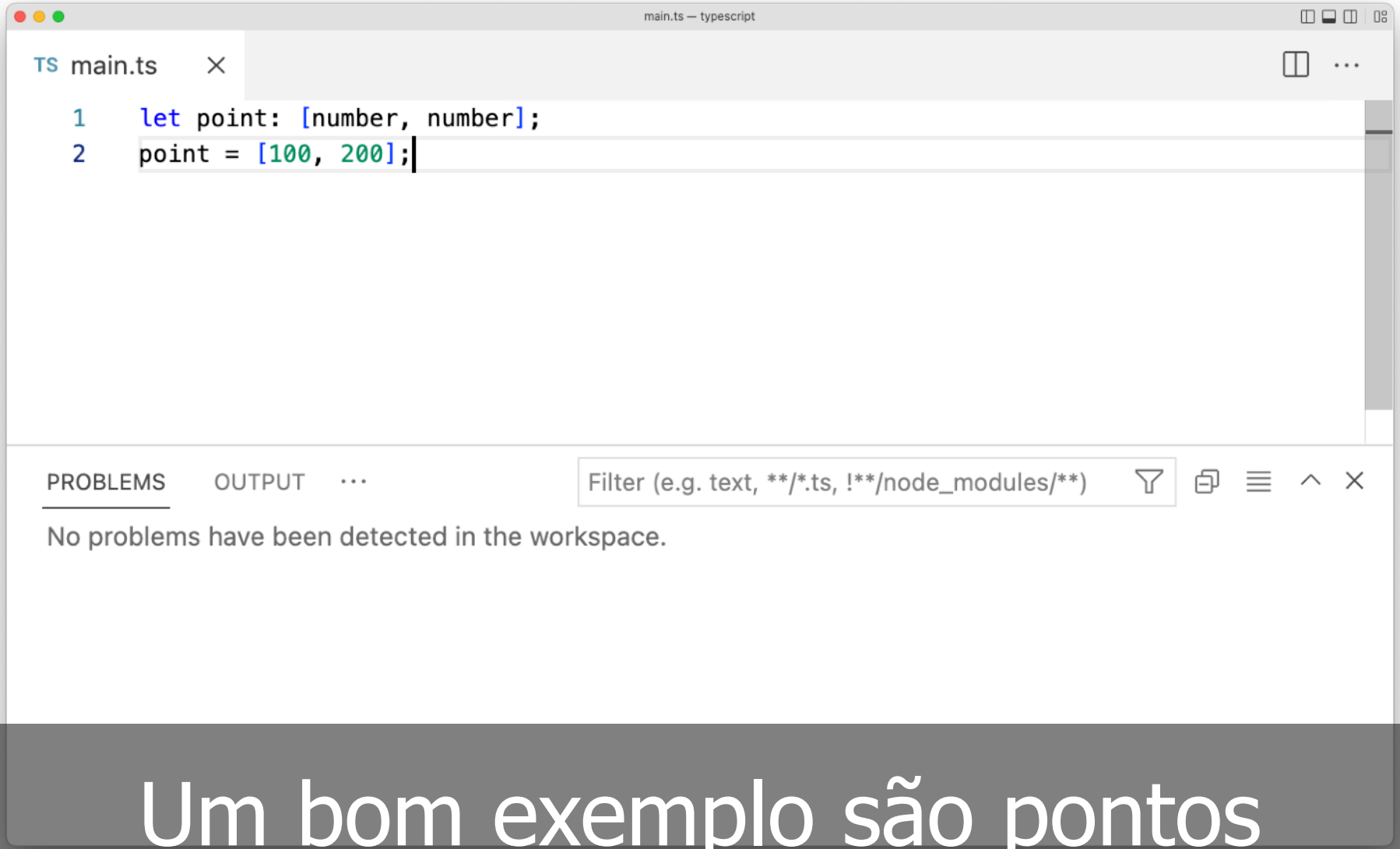


A classe que extende a classe abstrata pode ser instanciada

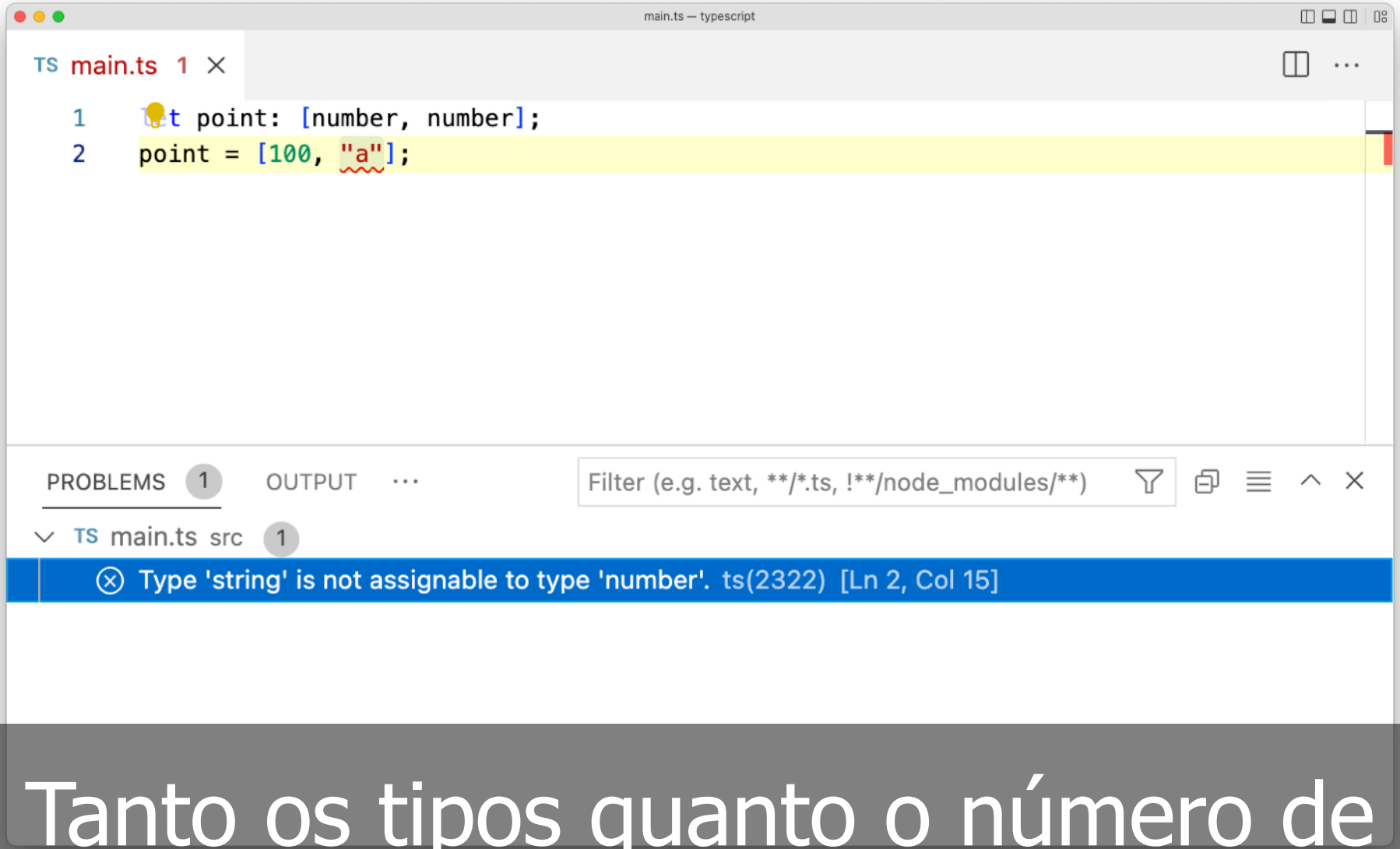
Qual é a diferença entre abstract class e interface?



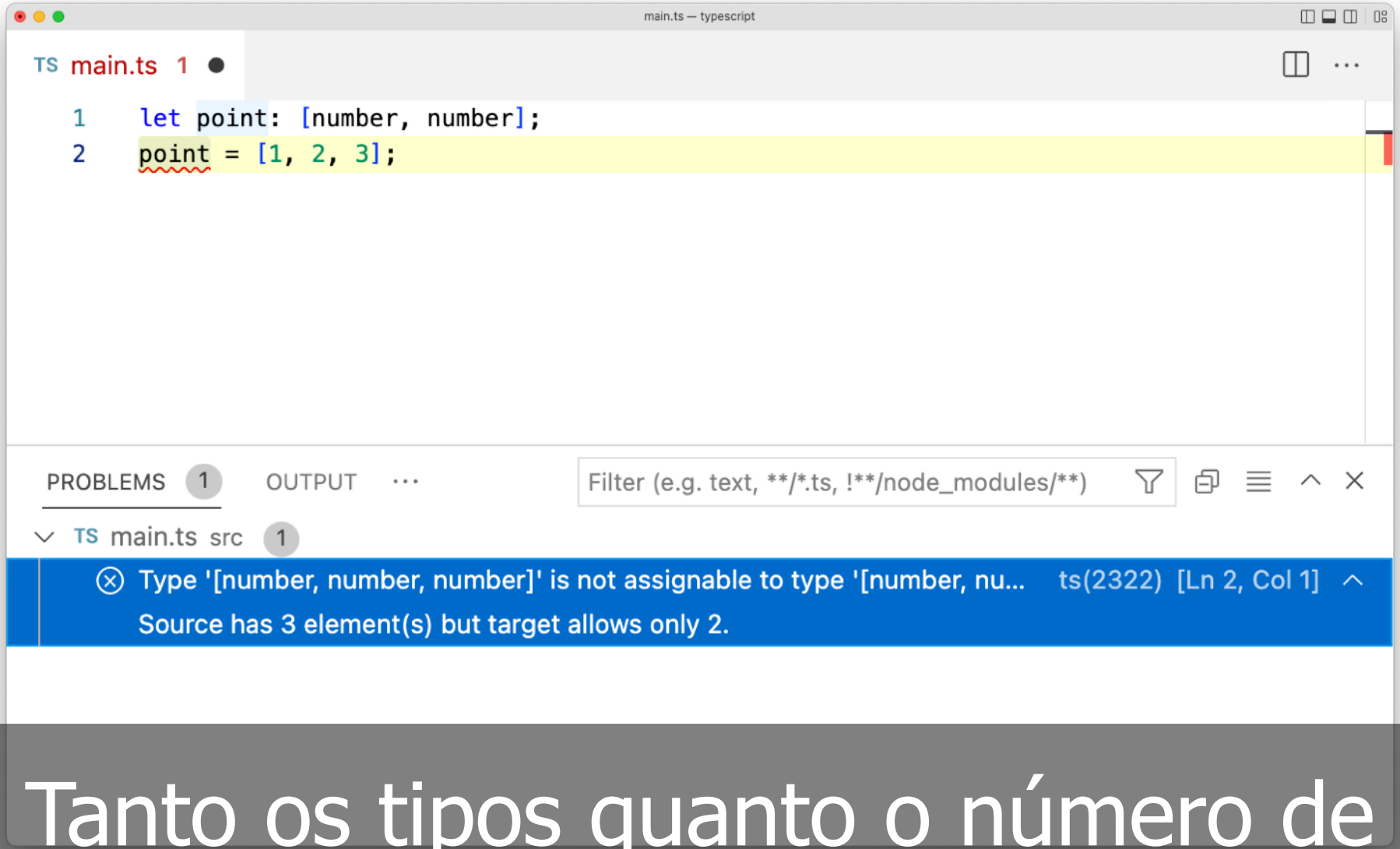
É possível também definir Tuplas que são arrays com elementos tipados



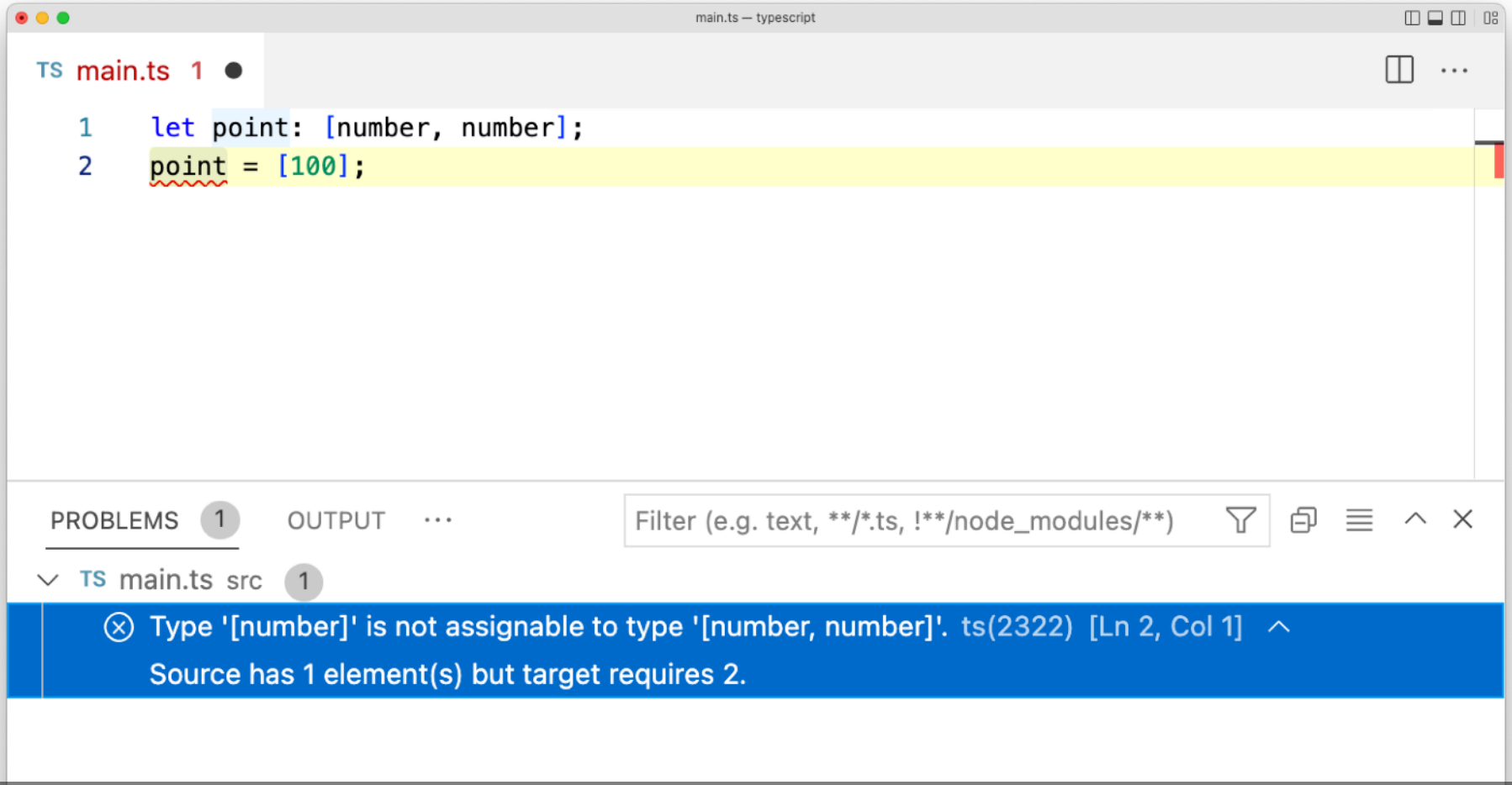
Um bom exemplo são pontos cartesianos ou latitude e longitude



Tanto os tipos quanto o número de elementos deve ser considerado



Tanto os tipos quanto o número de elementos deve ser considerado

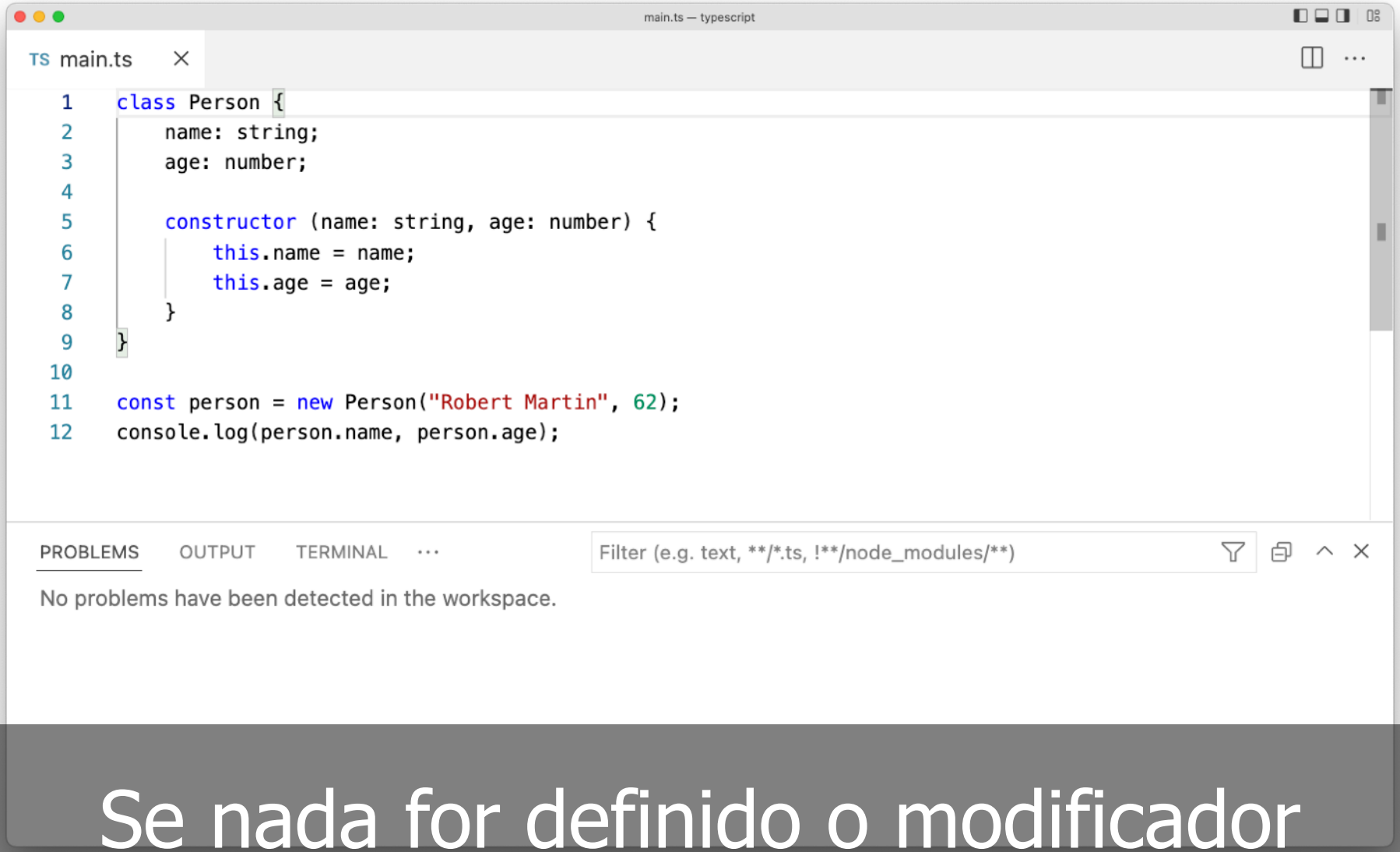


Tanto os tipos quanto o número de elementos deve ser considerado

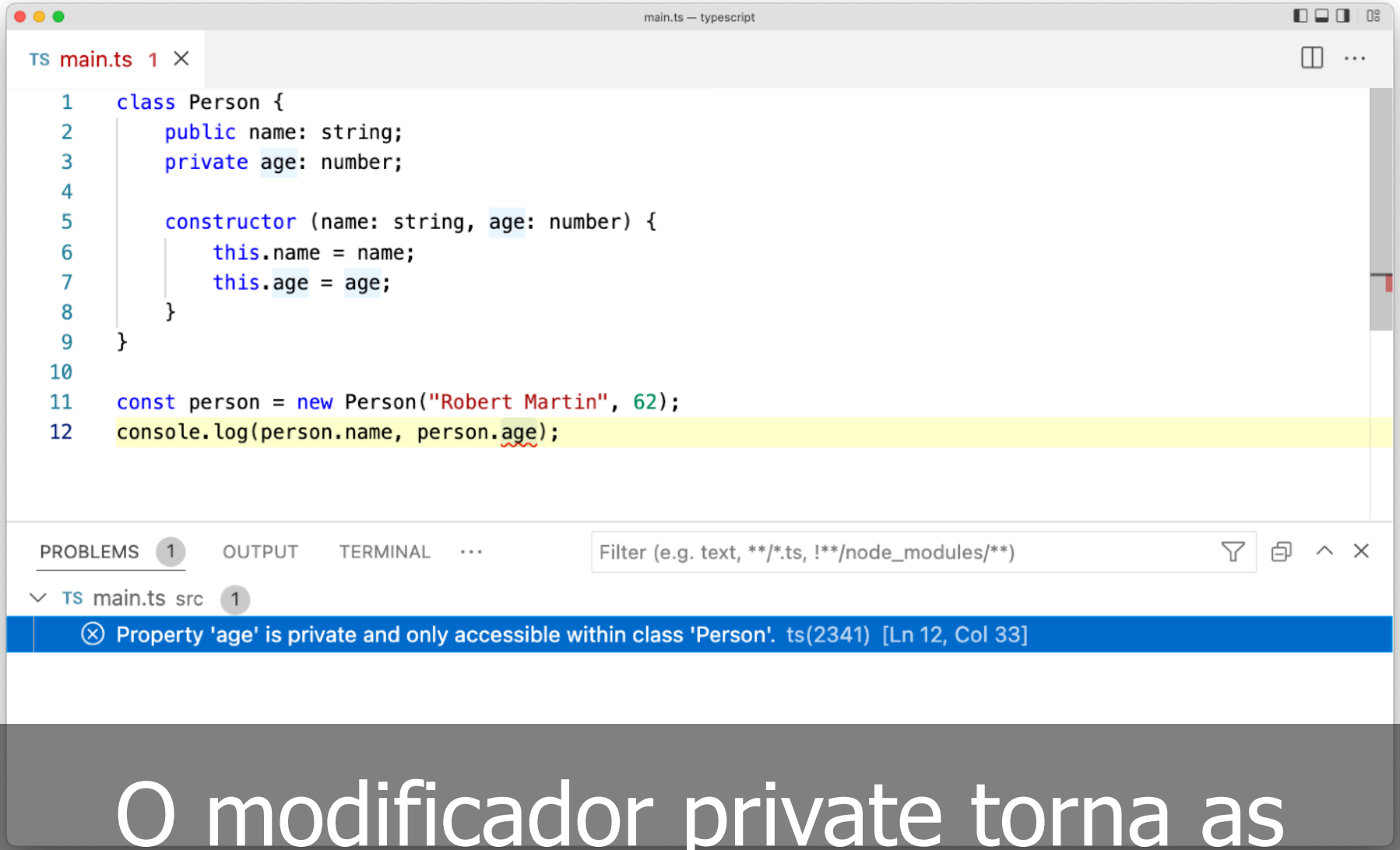
Modificadores de Visibilidade

Existem diversos modificadores no TypeScript como public, private e protected, sendo public é o padrão

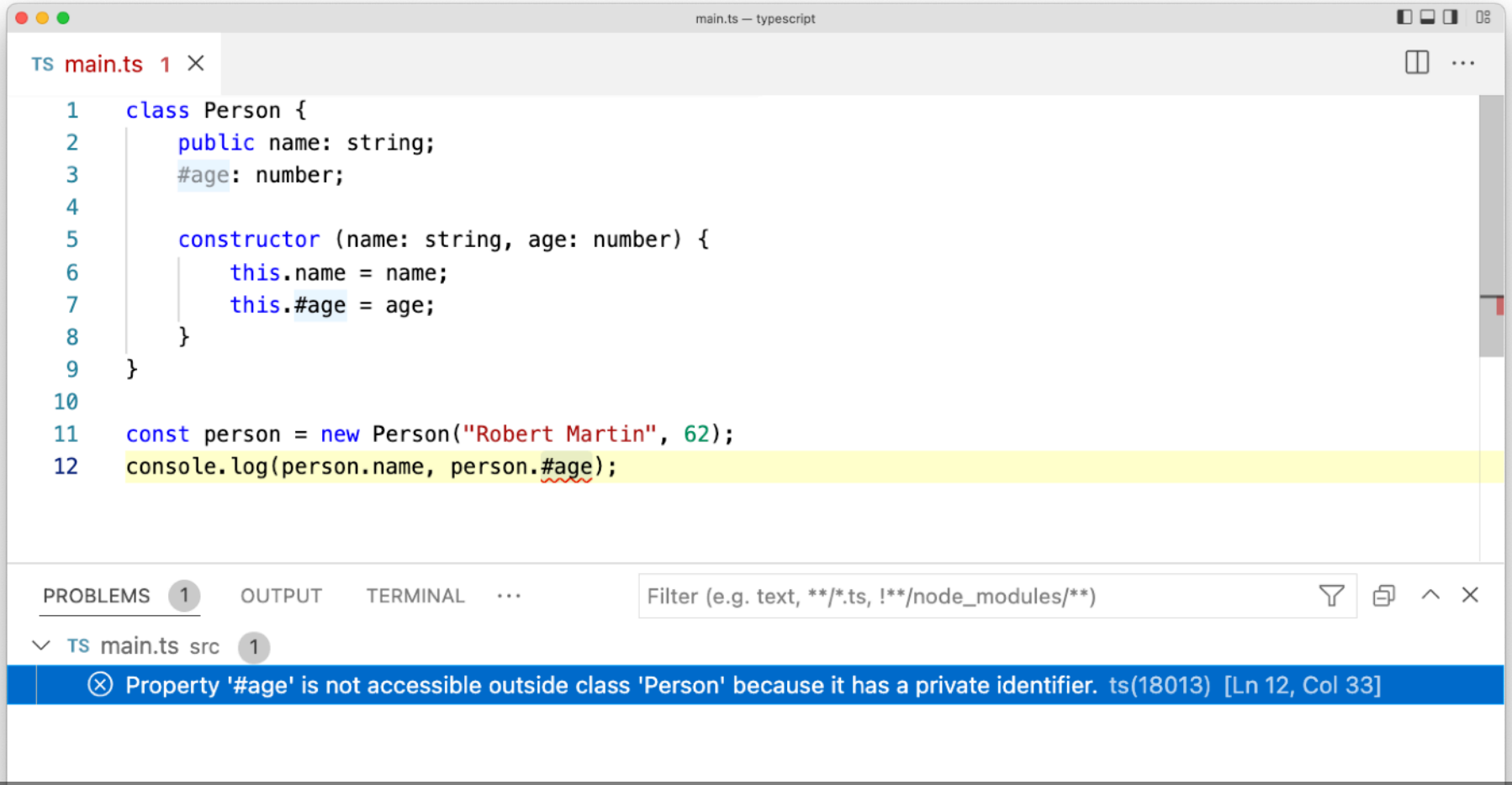
O modificador public permite que a propriedade seja acessada externamente



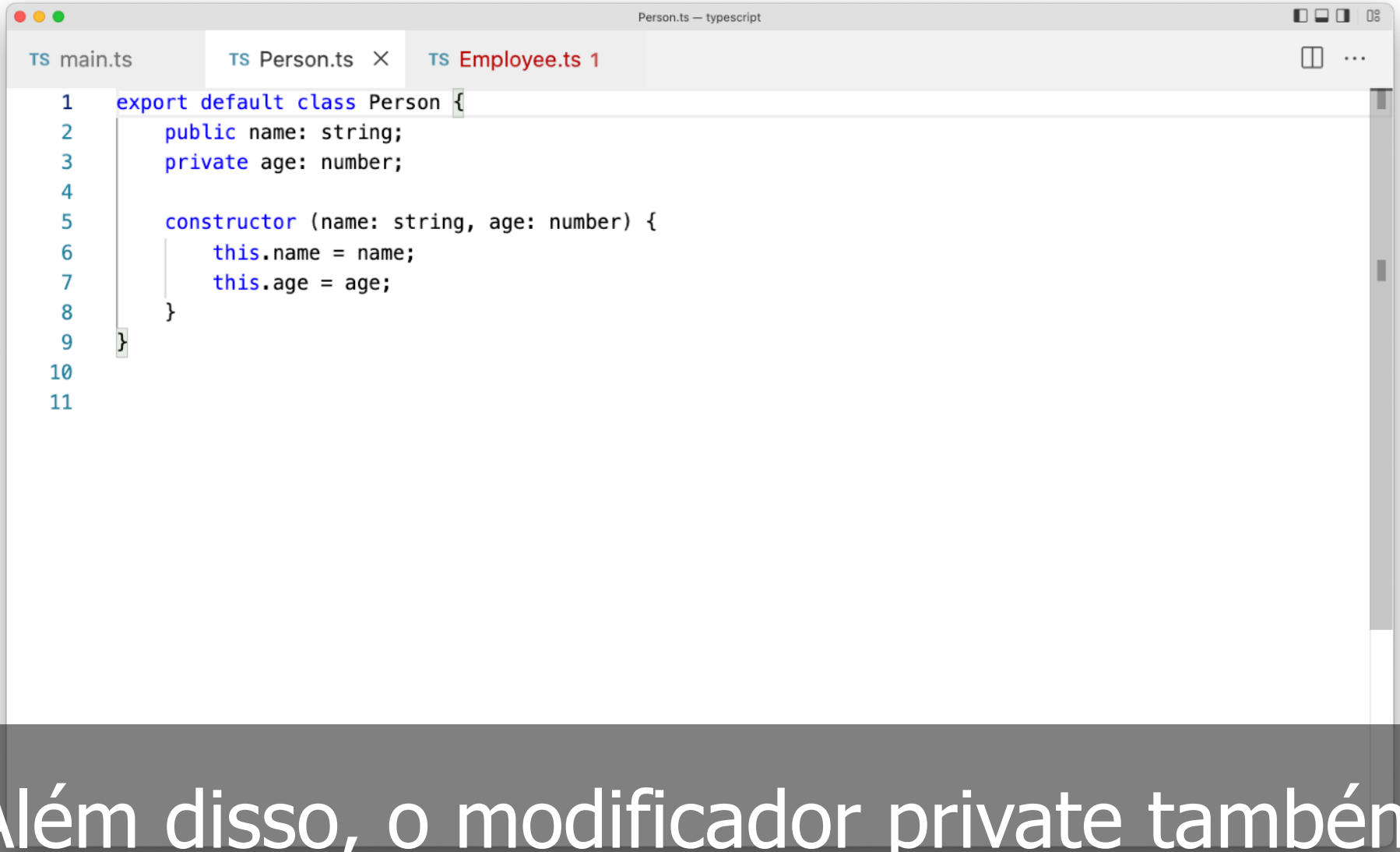
Se nada for definido o modificador utilizado é public



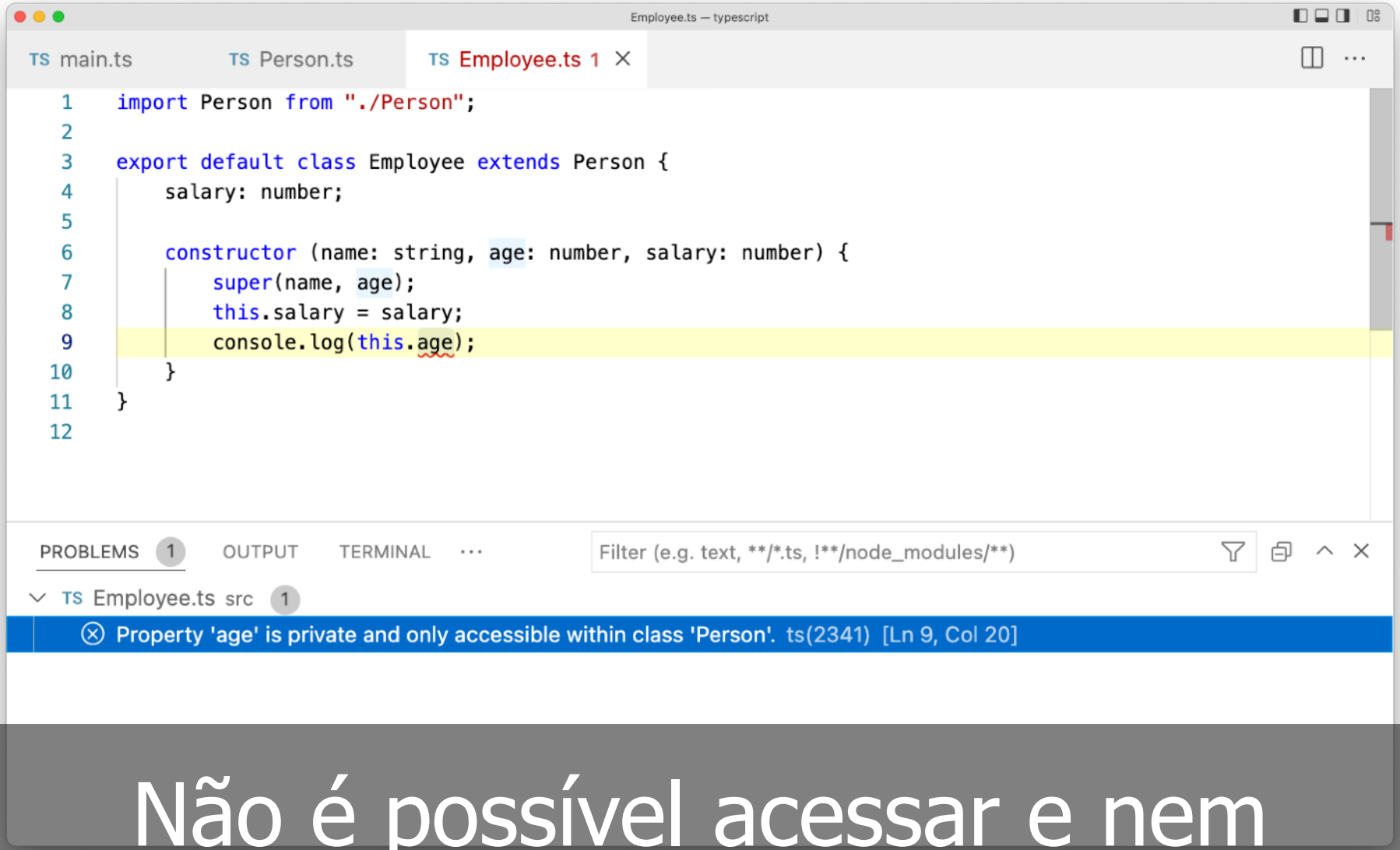
O modificador private torna as propriedades inacessíveis



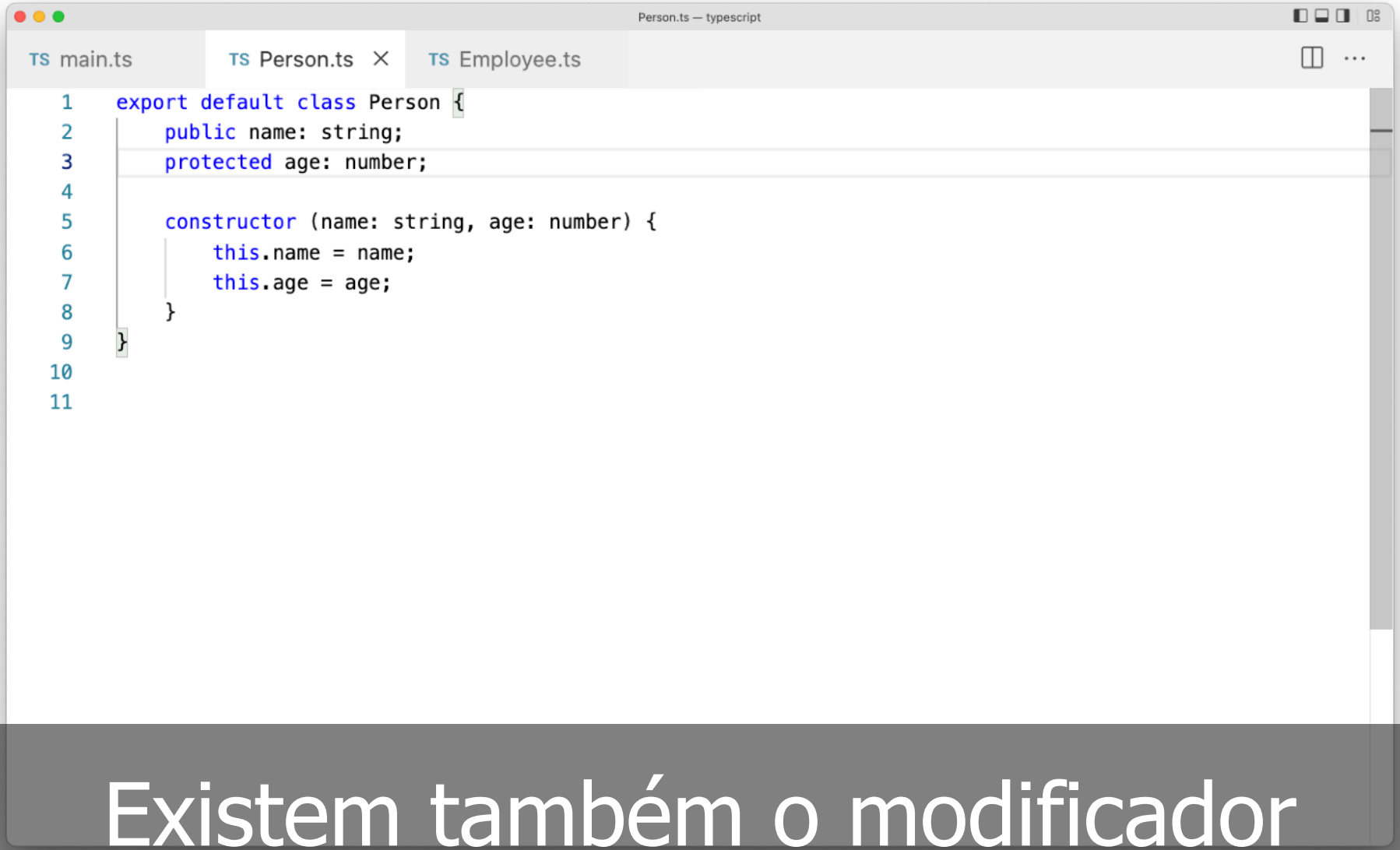
Também é possível utilizar o # como modificador privado



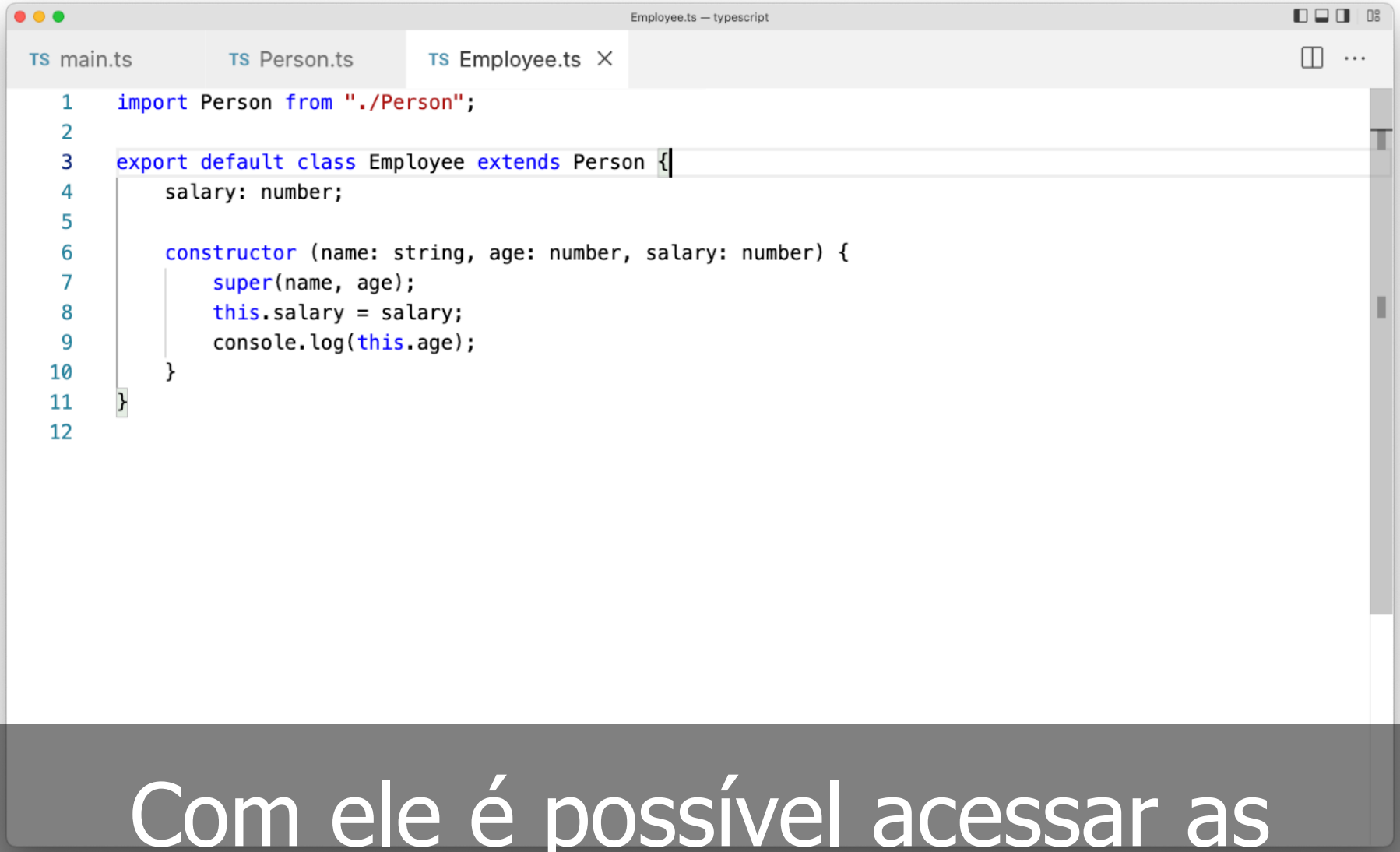
Além disso, o modificador private também restringe as propriedades em subclasses



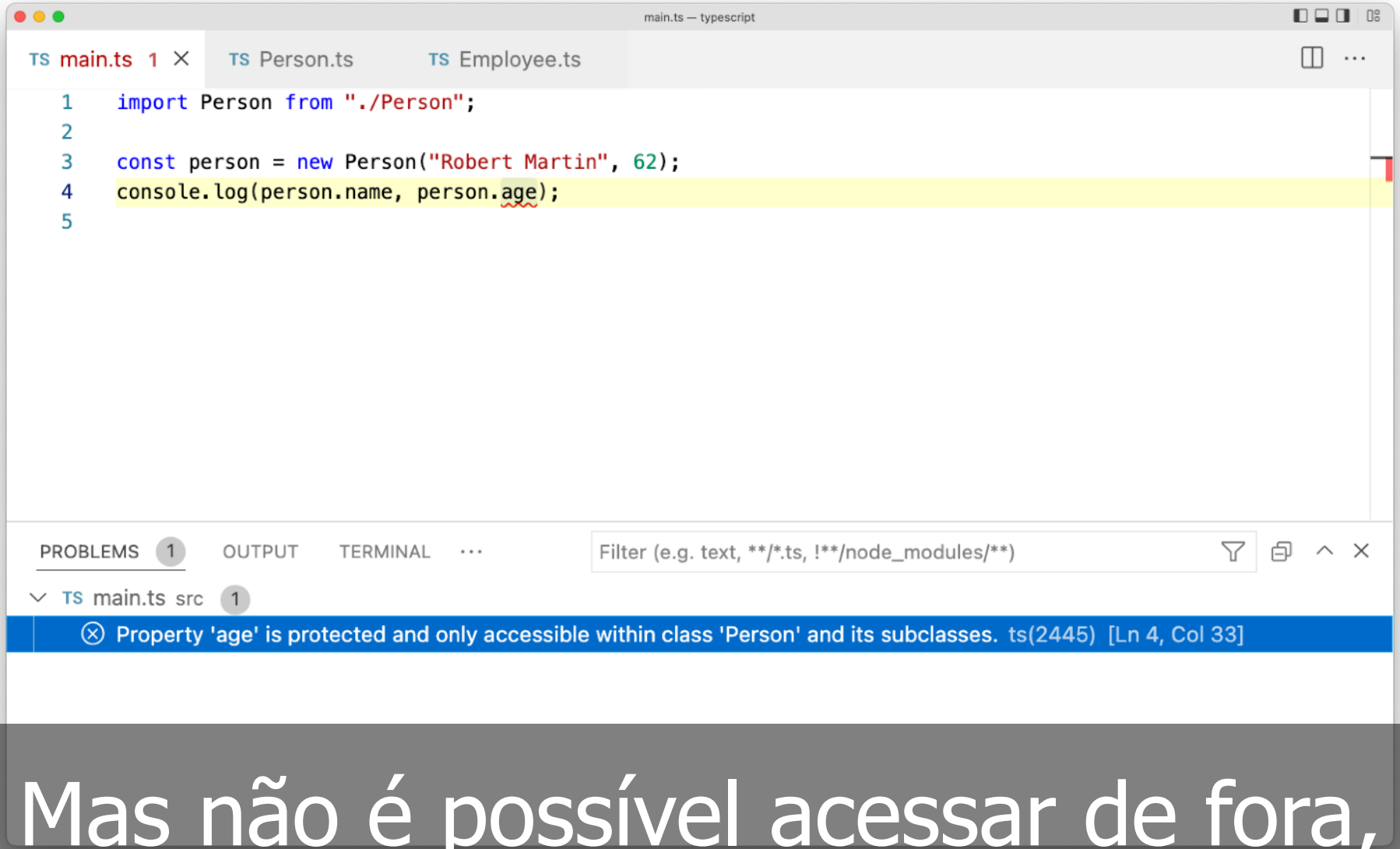
Não é possível acessar e nem modificar



Existem também o modificador protected



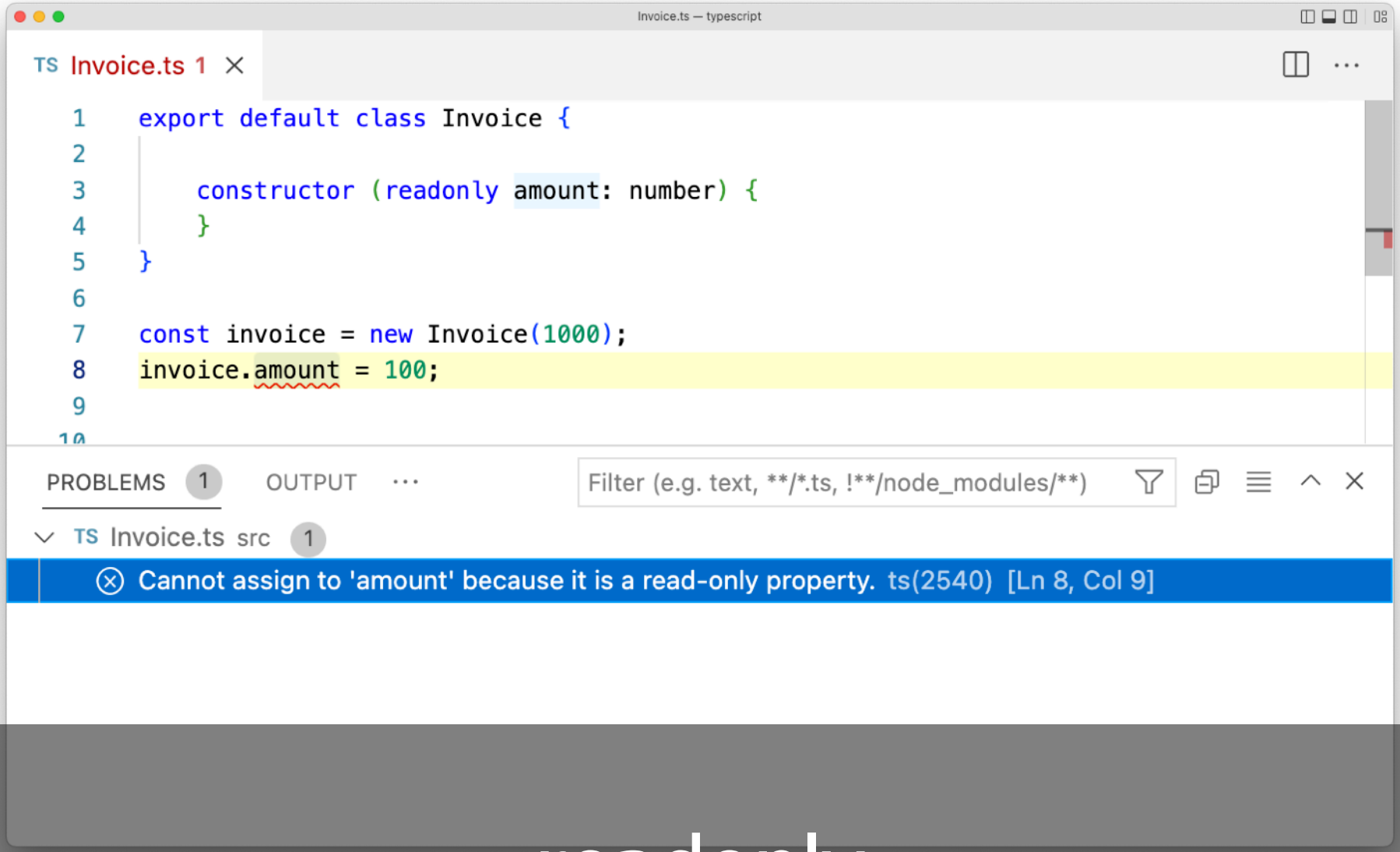
Com ele é possível acessar as propriedades nas subclsses



Mas não é possível acessar de fora, por meio da instância

Readonly

O modificador readonly impede que uma propriedade seja modificada após ser atribuída

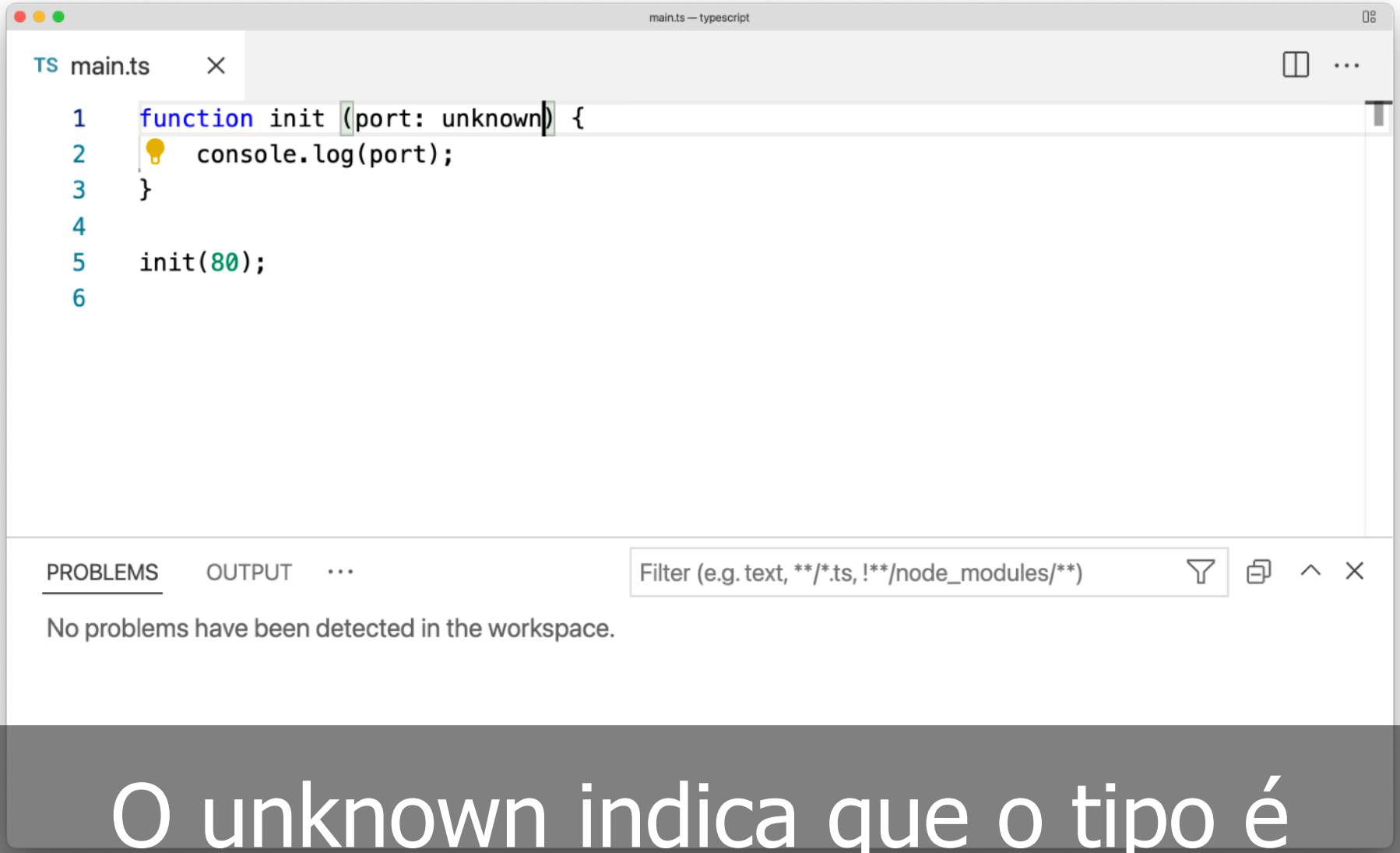

Invoice.ts — typescript
TS Invoice.ts 1
1 export default class Invoice {
2
3 constructor (readonly amount: number) {
4 }
5 }
6
7 const invoice = new Invoice(1000);
8 invoice.amount = 100;
9
PROBLEMS 1
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS Invoice.ts src 1
Cannot assign to 'amount' because it is a read-only property. ts(2540) [Ln 8, Col 9]

readonly

any e unknown

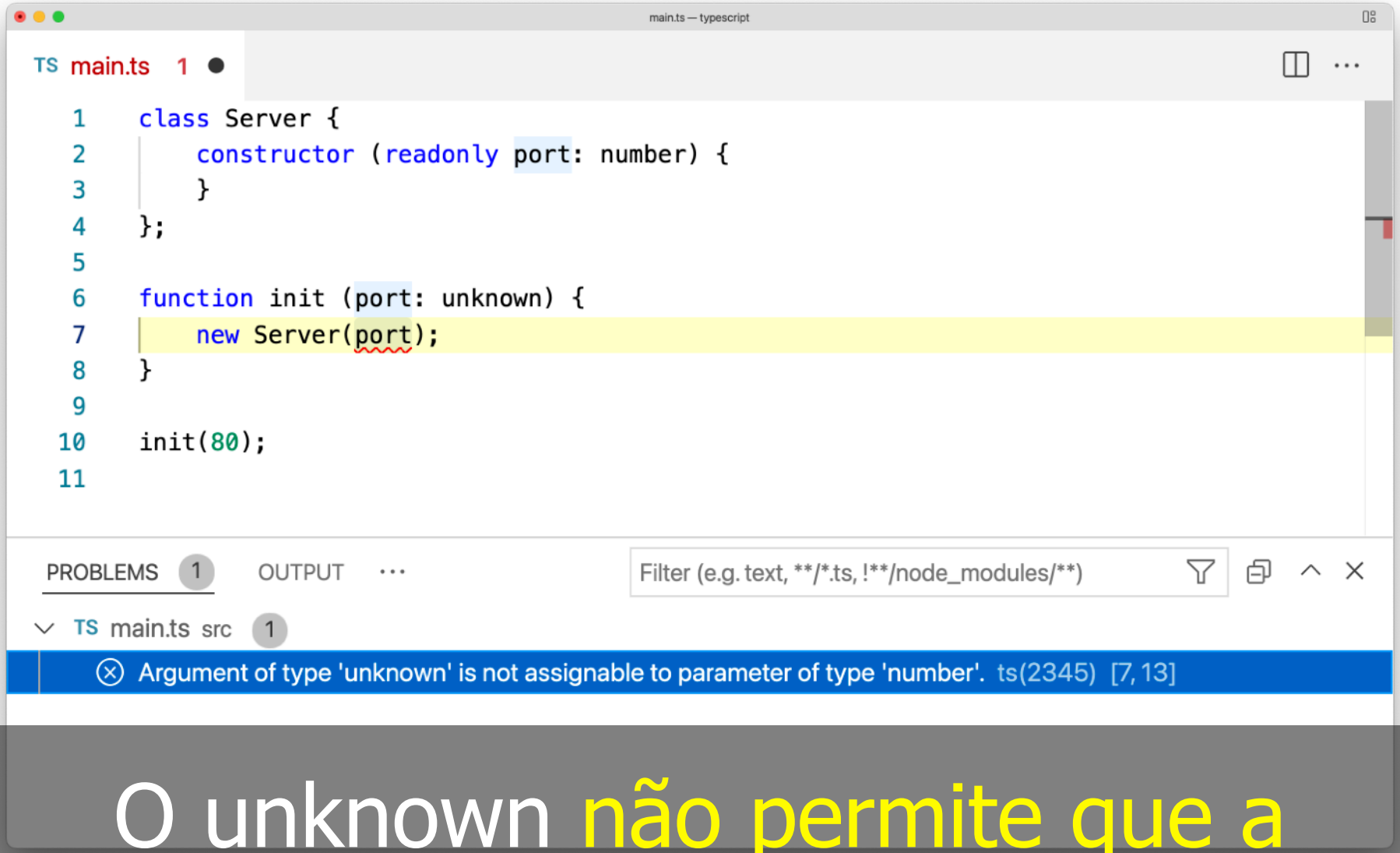
Muitas vezes não sabemos exatamente o tipo que queremos utilizar, nesse caso existem algumas possibilidades como o any e o unknown

O any indica que o tipo pode ser qualquer coisa



O unknown indica que o tipo é desconhecido

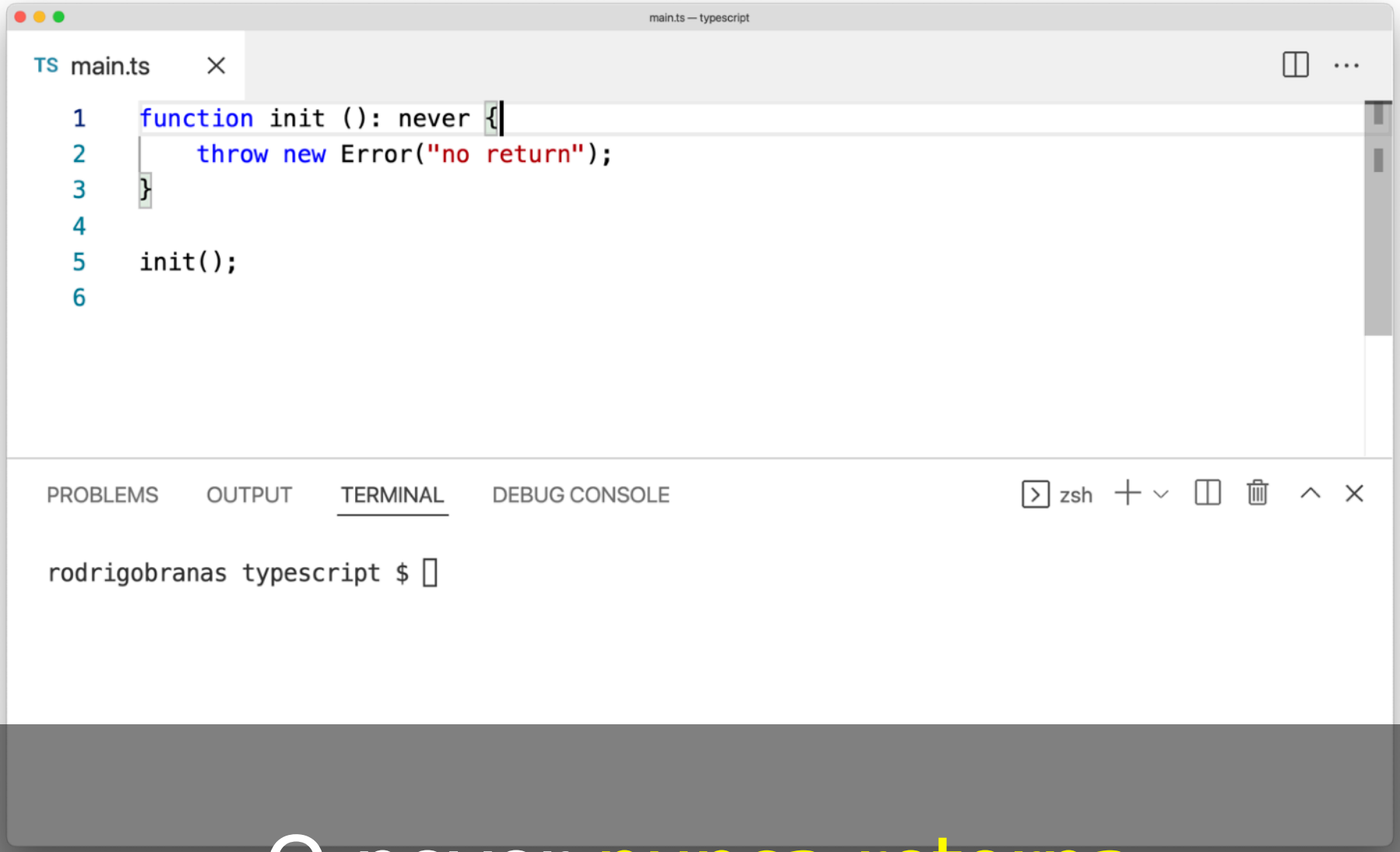
O any simplesmente desliga o type-checking



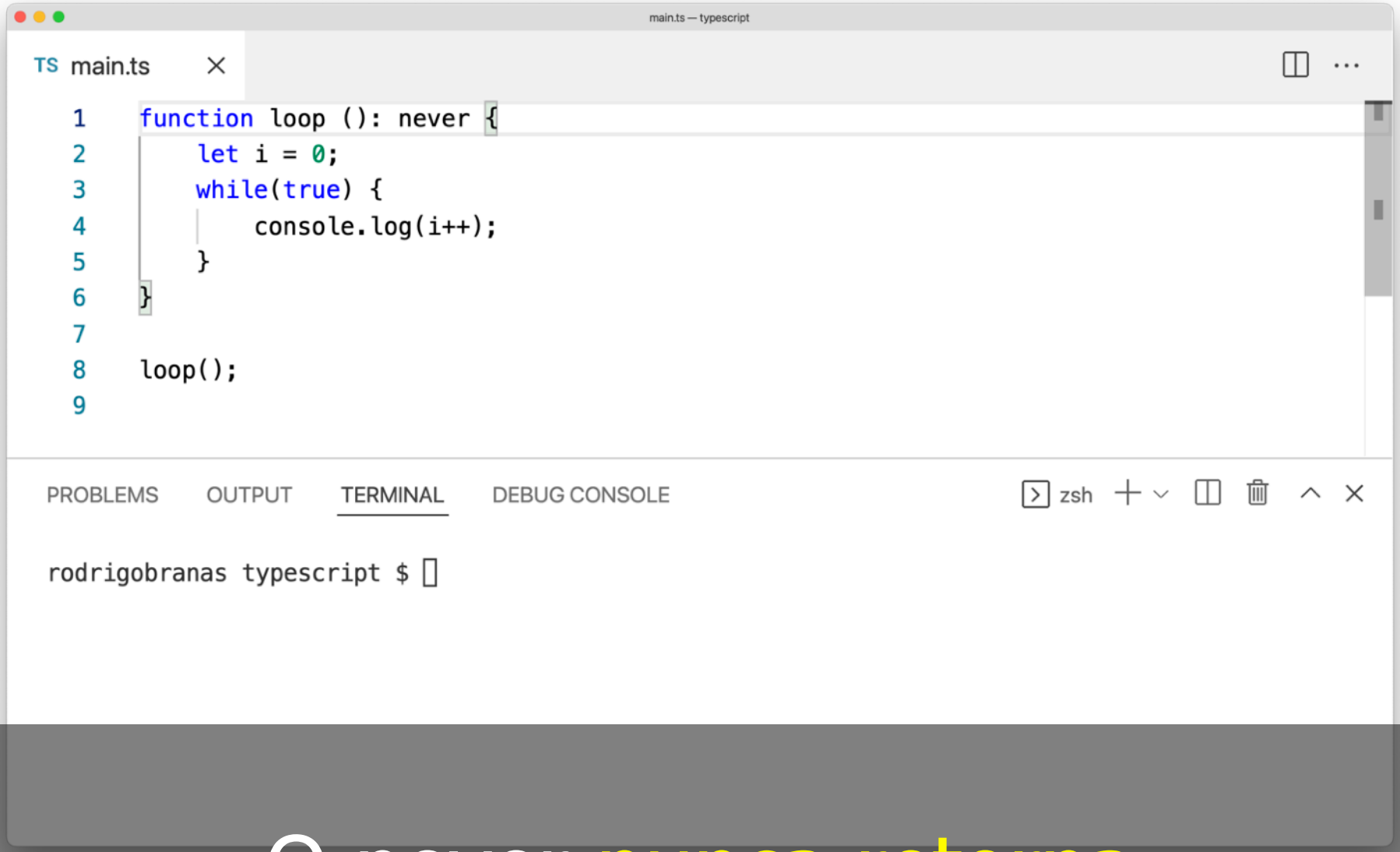
O unknown não permite que a variável seja utilizada em outro tipo

never

O tipo never indica que algo nunca vai acontecer, por exemplo, o retorno de uma function



O never nunca retorna

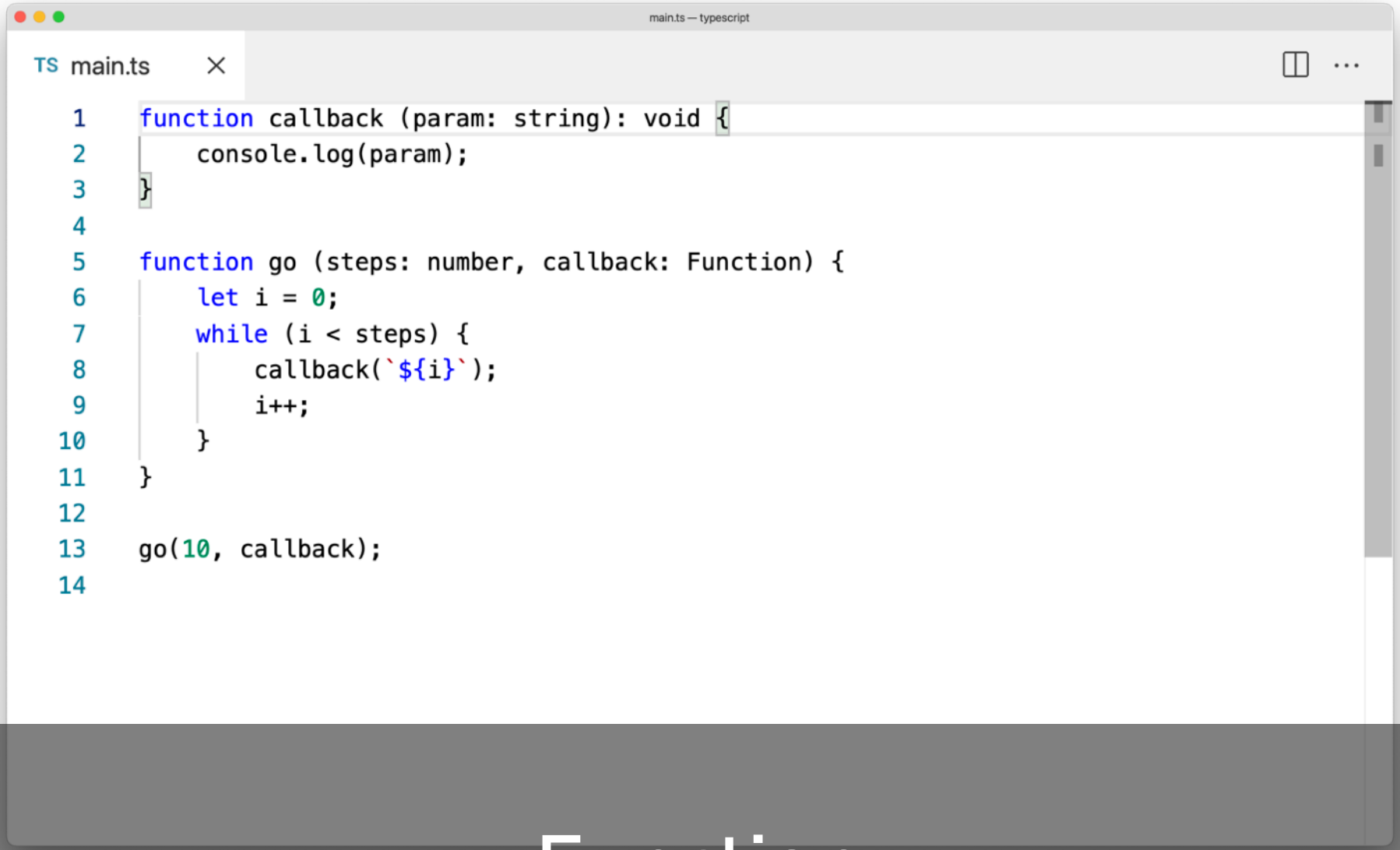


O never nunca retorna

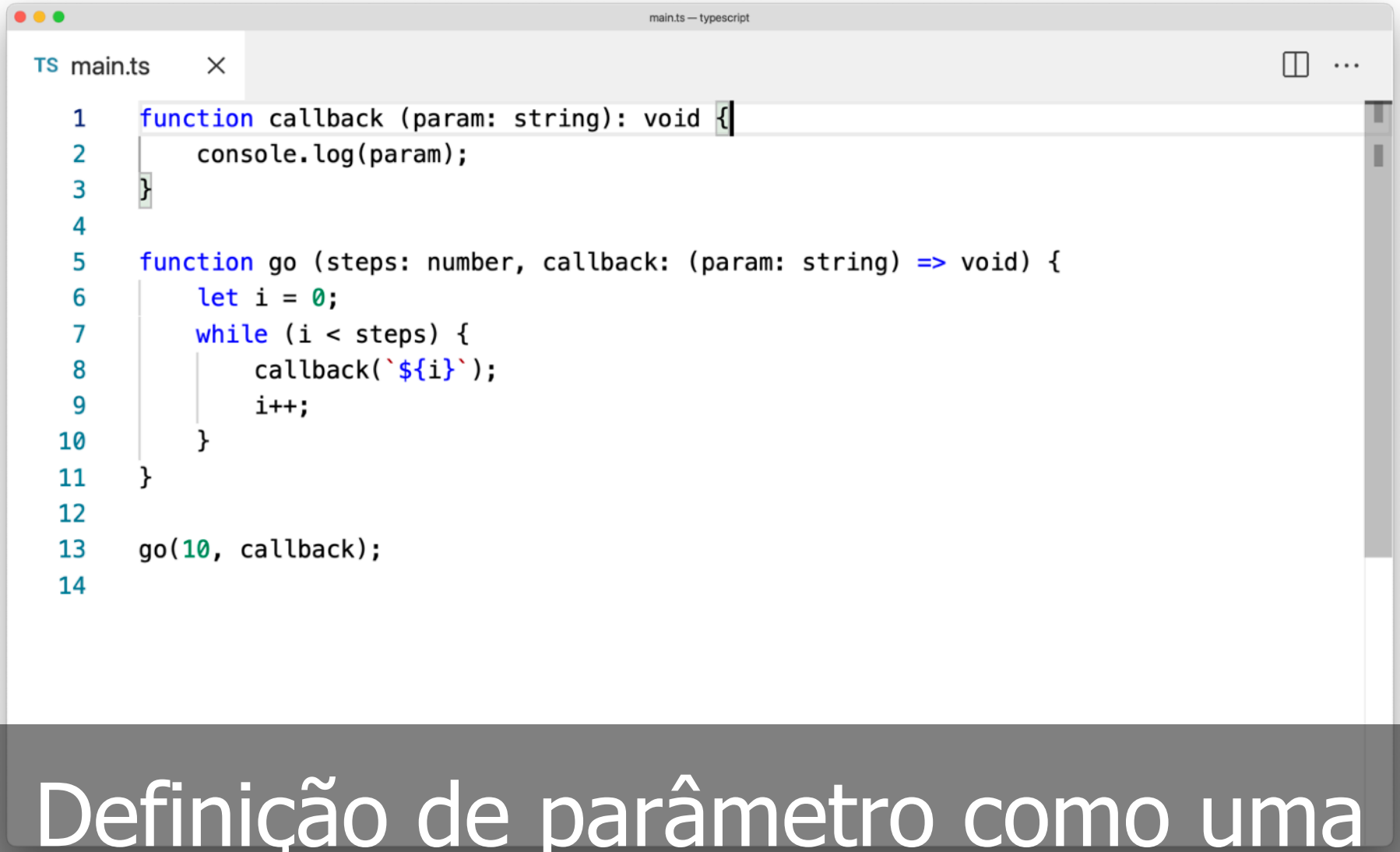

Qual é a diferença entre void e never?

Function

Parâmetros podem ser do tipo Function


main.ts — typescript
TS main.ts
function callback (param: string): void {
console.log(param);
}
function go (steps: number, callback: Function) {
let i = 0;
while (i < steps) {
callback(`${i}`);
i++;
}
}
go(10, callback);

Function

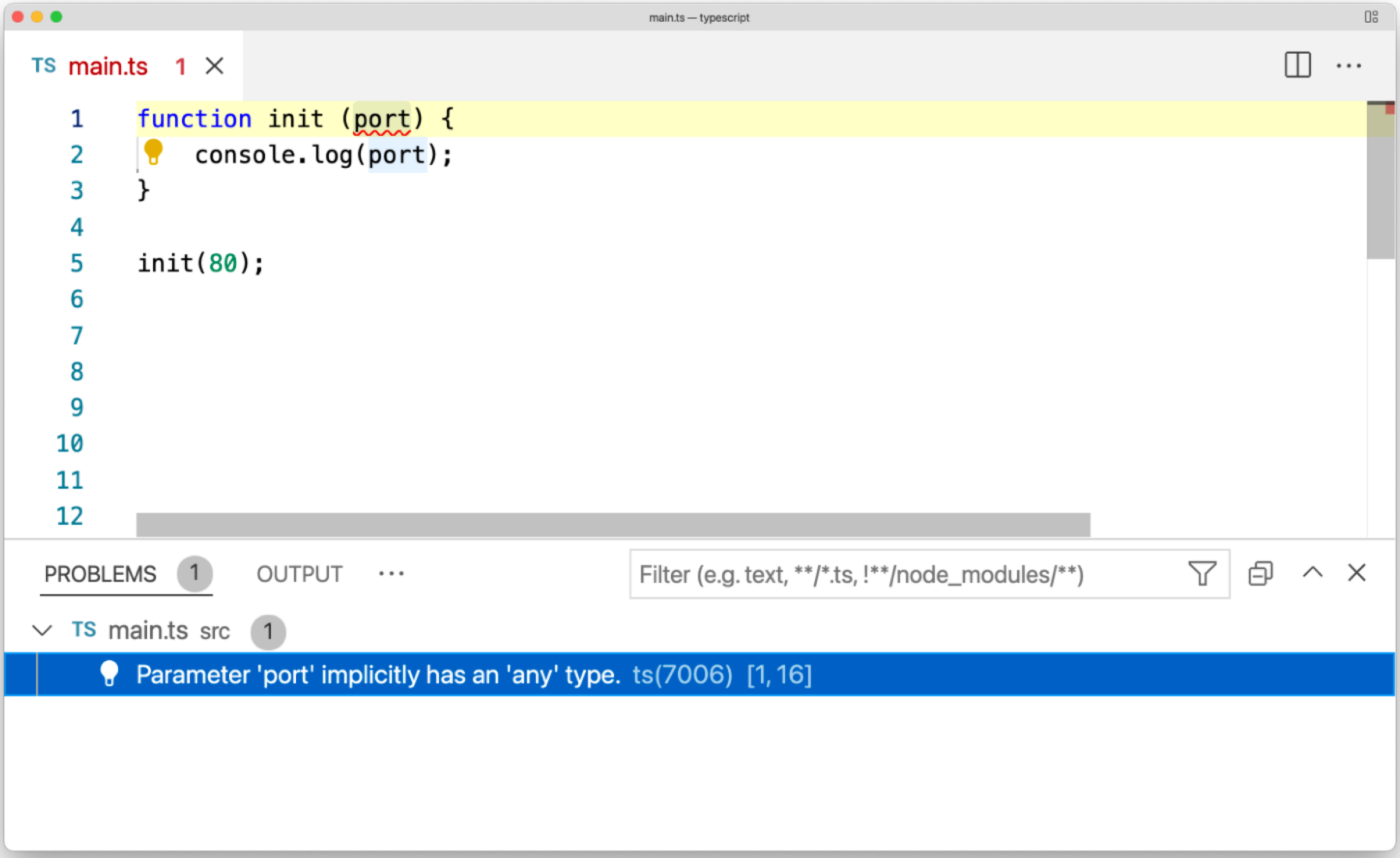


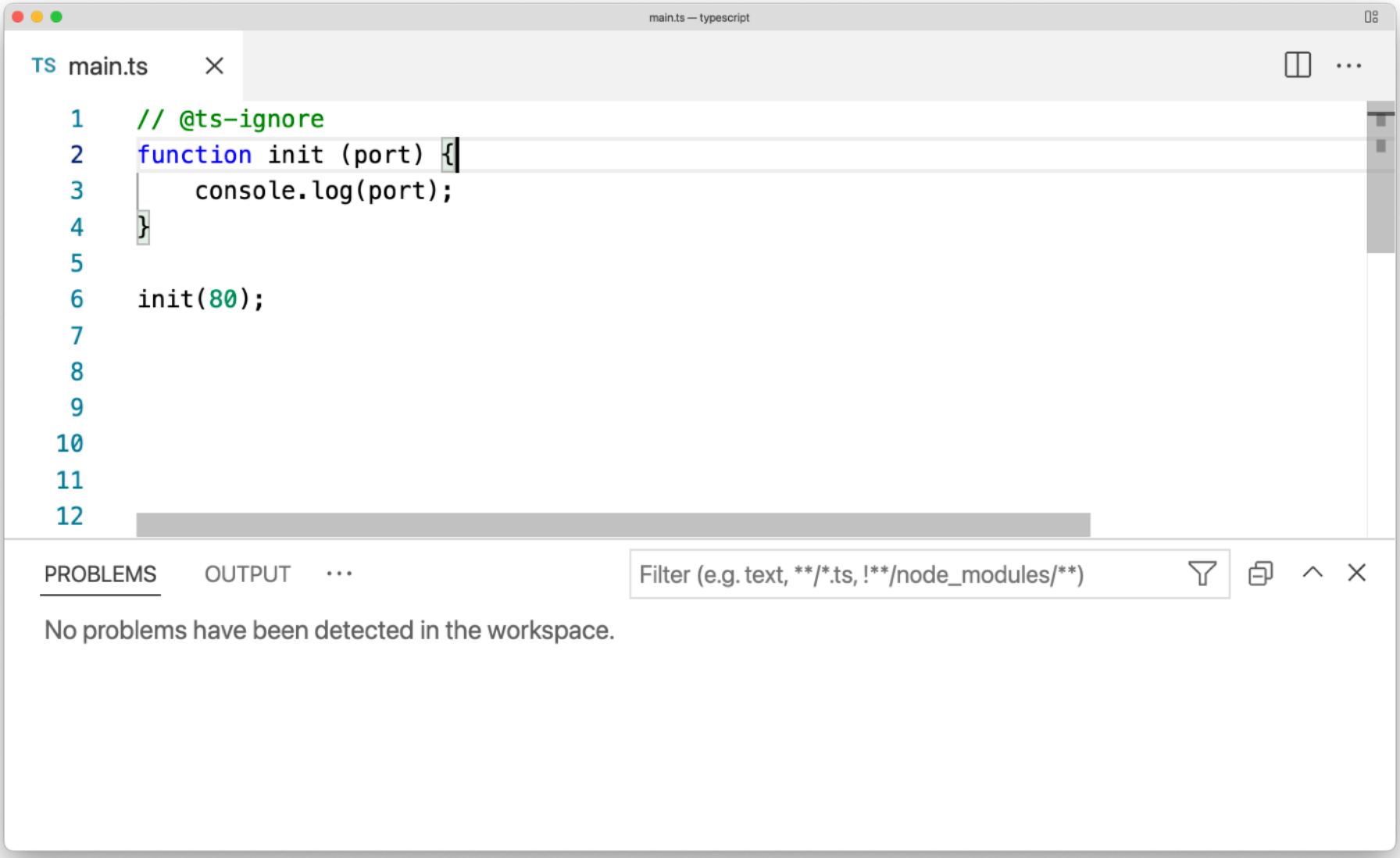
Definição de parâmetro como uma definição de função

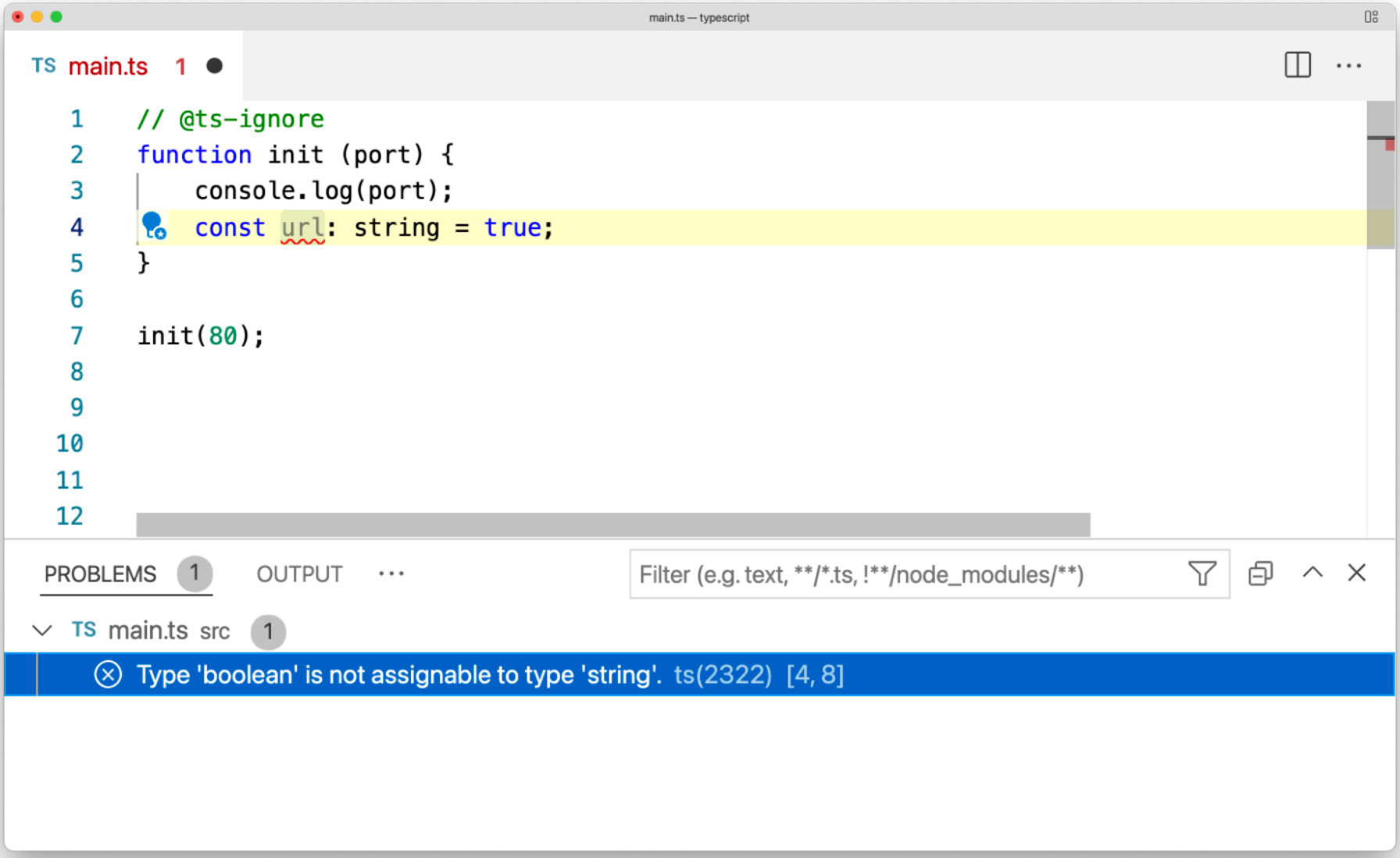
Definição de parâmetro como uma definição de função

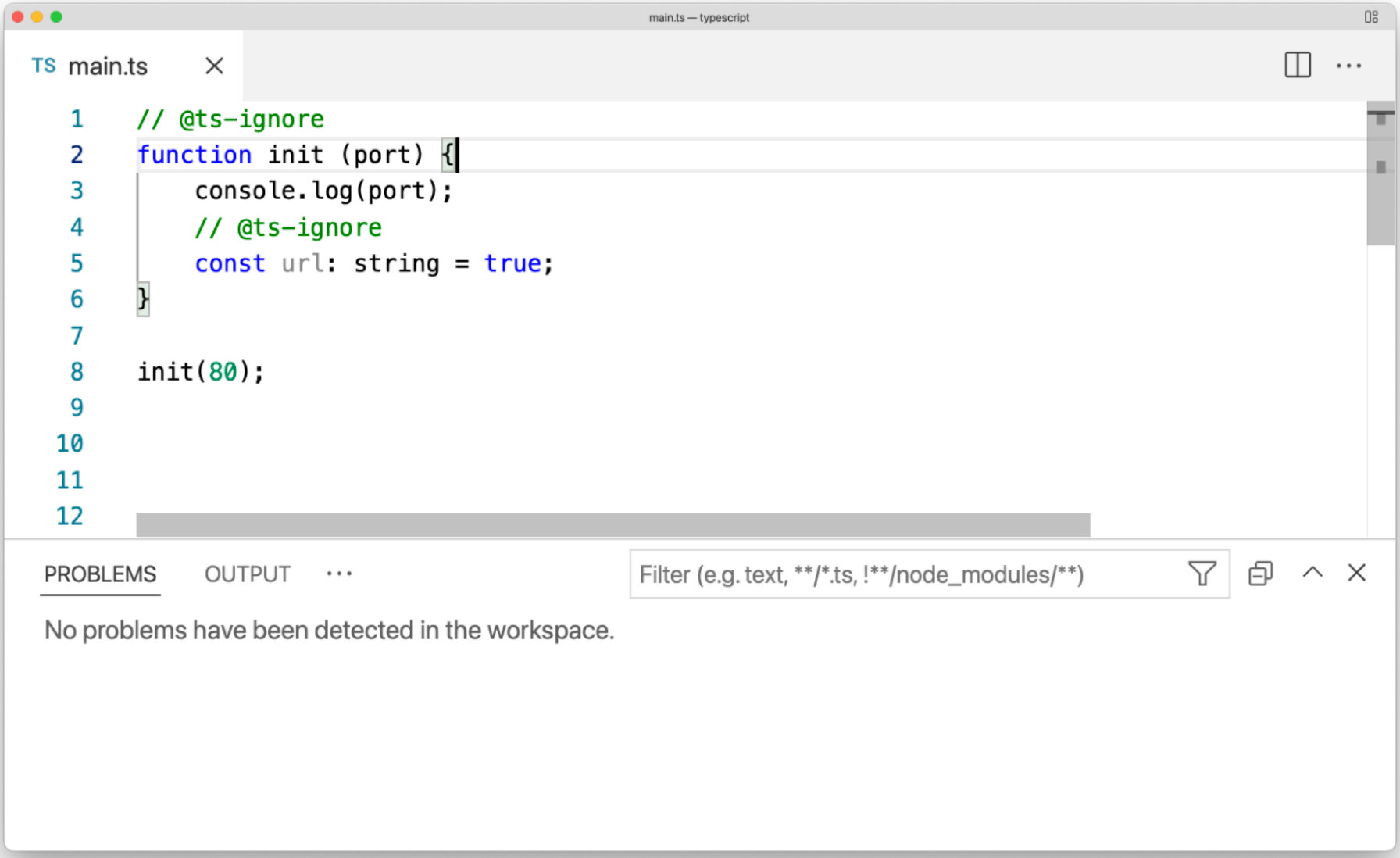
Suppress Comments

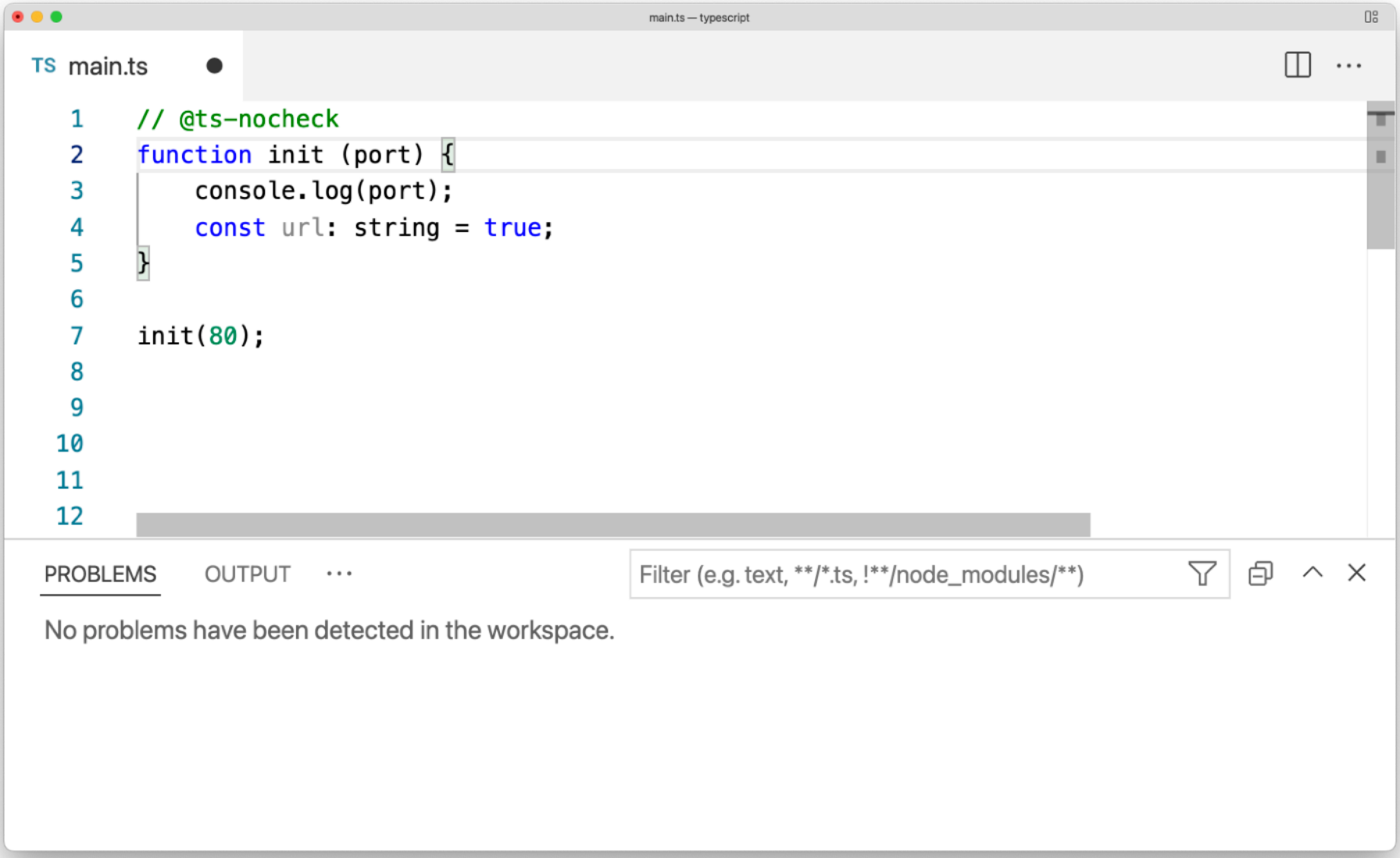
Muitas vezes é necessário desligar a verificação do TypeScript em determinados pontos do código-fonte e para isso é possível utilizar comentários que orientam o transpiler


main.ts — typescript
TS main.ts 1
1 function init (port) {
2 console.log(port);
3 }
4
5 init(80);
6
7
8
9
10
11
12
PROBLEMS 1
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 1
Parameter 'port' implicitly has an 'any' type. ts(7006) [1, 16]


main.ts — typescript
TS main.ts
// @ts-ignore
function init (port) {
console.log(port);
}
init(80);
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.

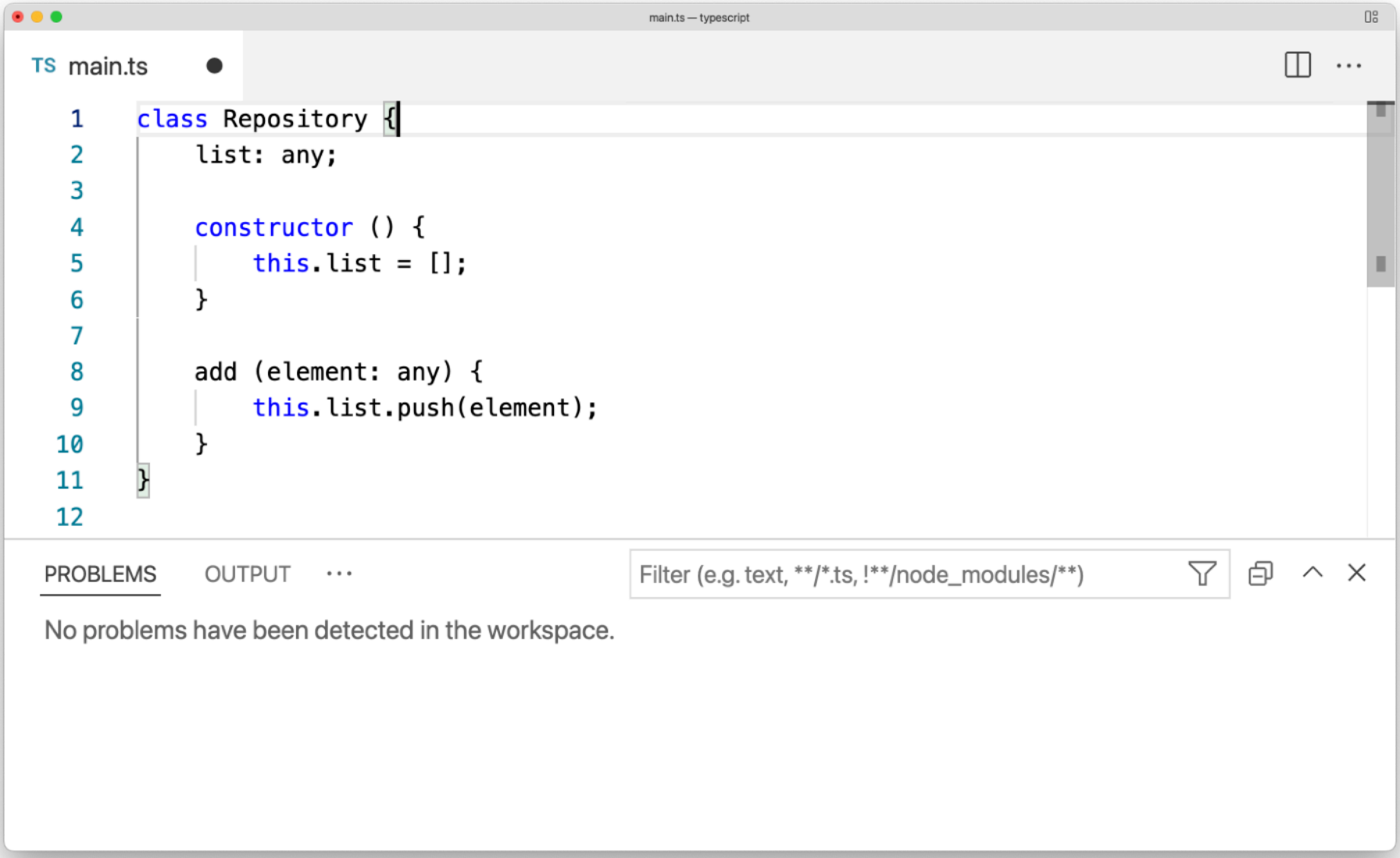

main.ts — typescript
TS main.ts 1
1 // @ts-ignore
2 function init (port) {
3 console.log(port);
4 const url: string = true;
5 }
6
7 init(80);
8
9
10
11
12
PROBLEMS 1
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 1
Type 'boolean' is not assignable to type 'string'. ts(2322) [4, 8]

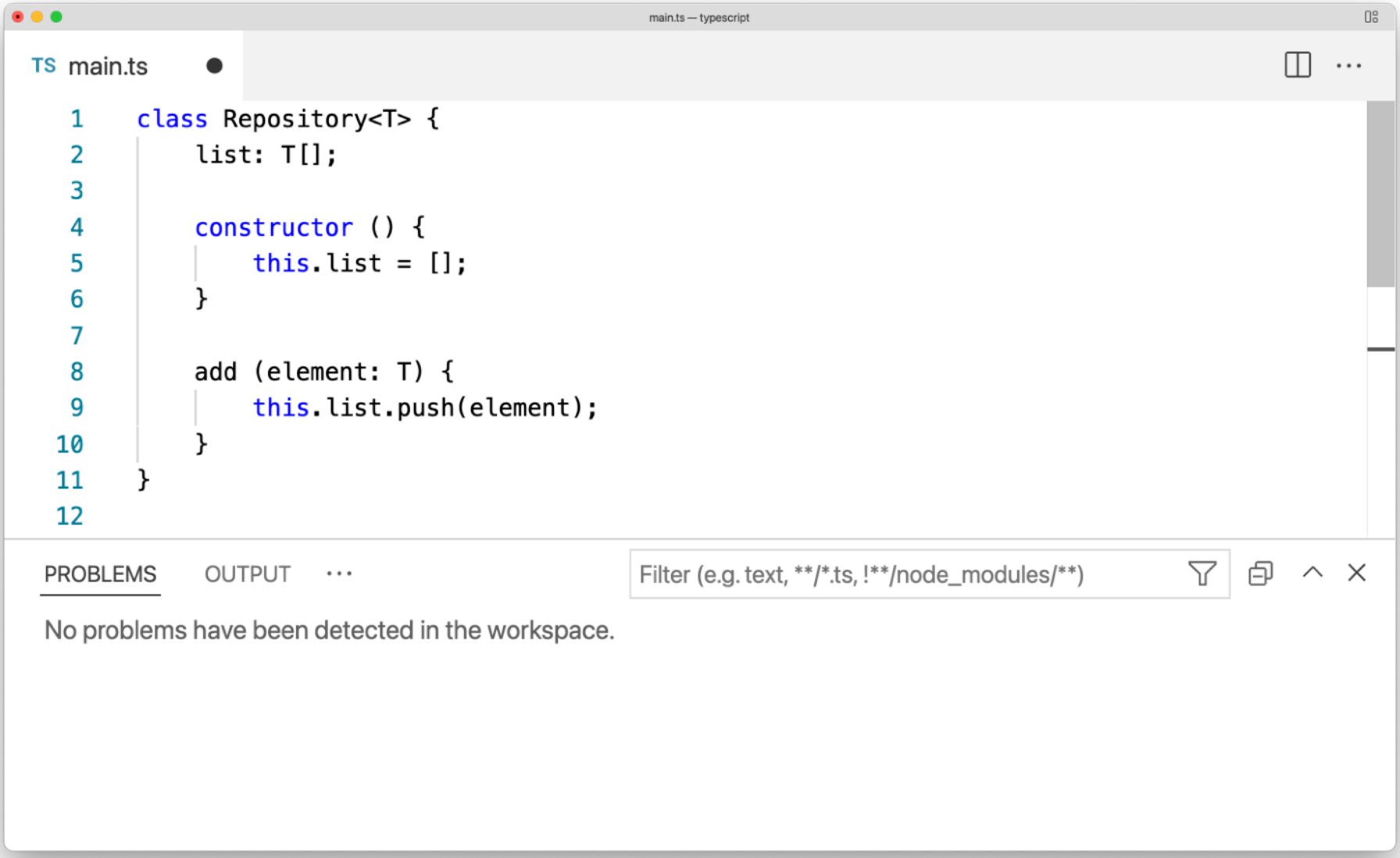

main.ts — typescript
TS main.ts
1 // @ts-ignore
2 function init (port) {
3 console.log(port);
4 // @ts-ignore
5 const url: string = true;
6 }
7
8 init(80);
9
10
11
12
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.

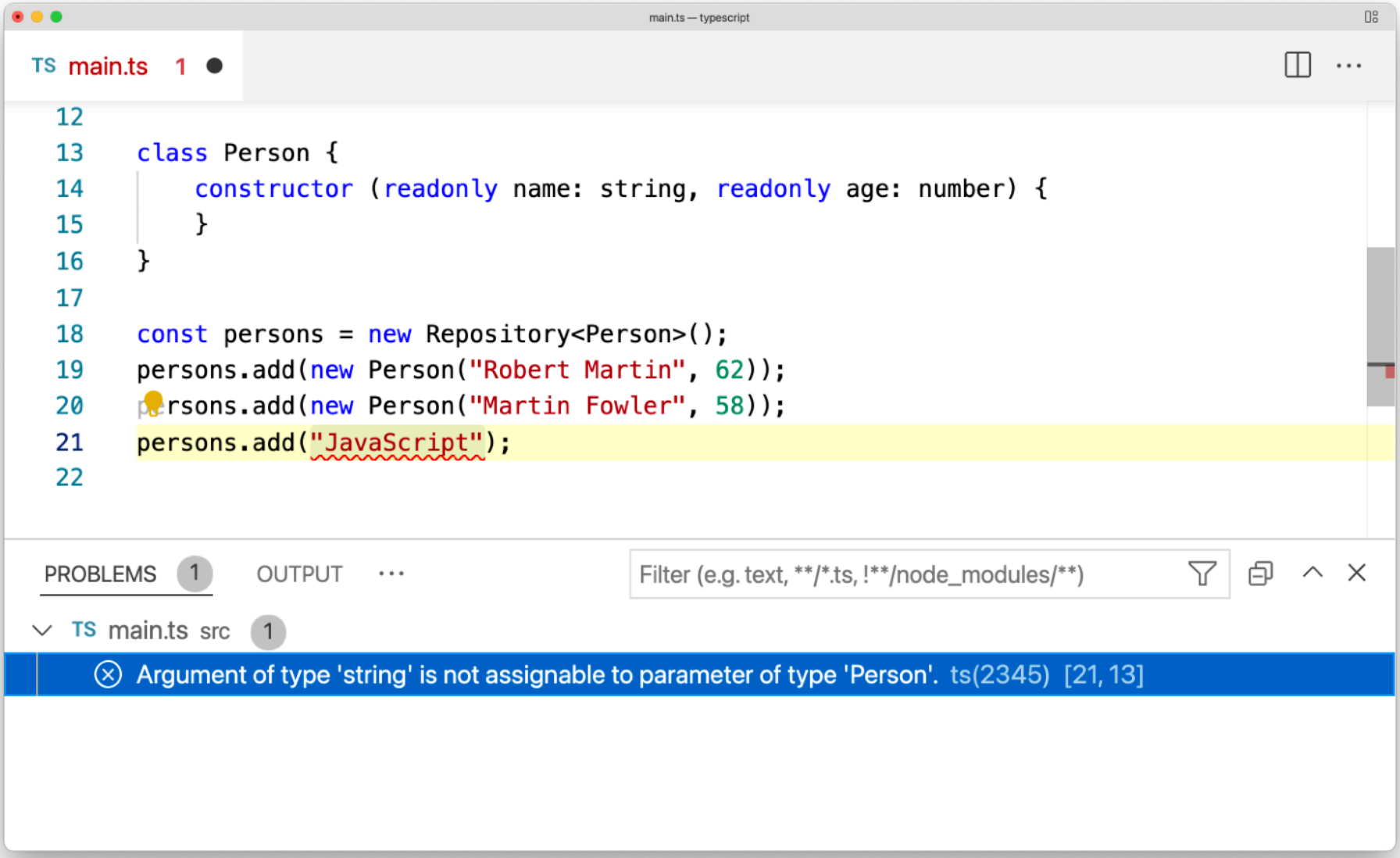

main.ts — typescript
TS main.ts
// @ts-nocheck
function init (port) {
console.log(port);
const url: string = true;
}
init(80);
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.

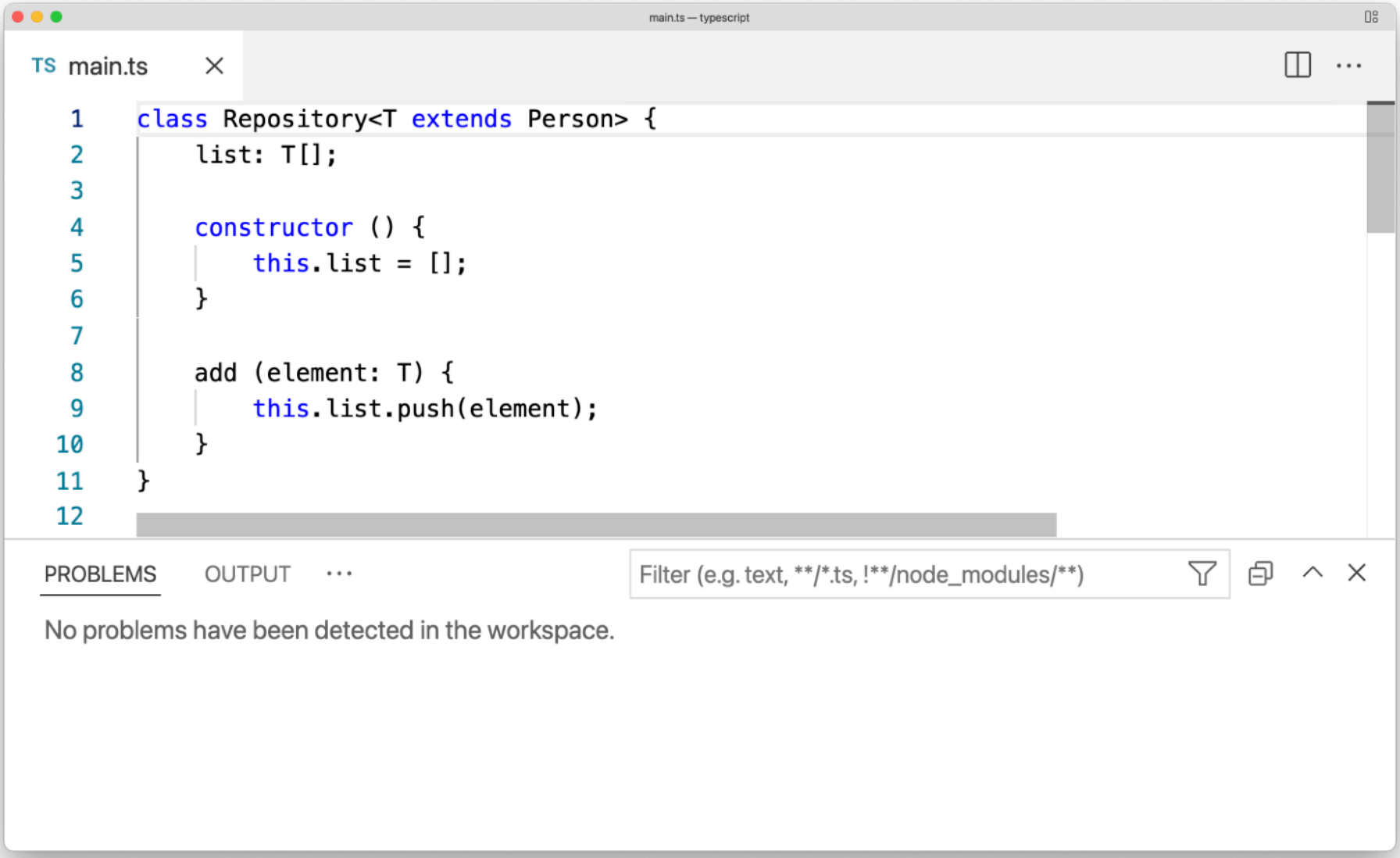
Generics

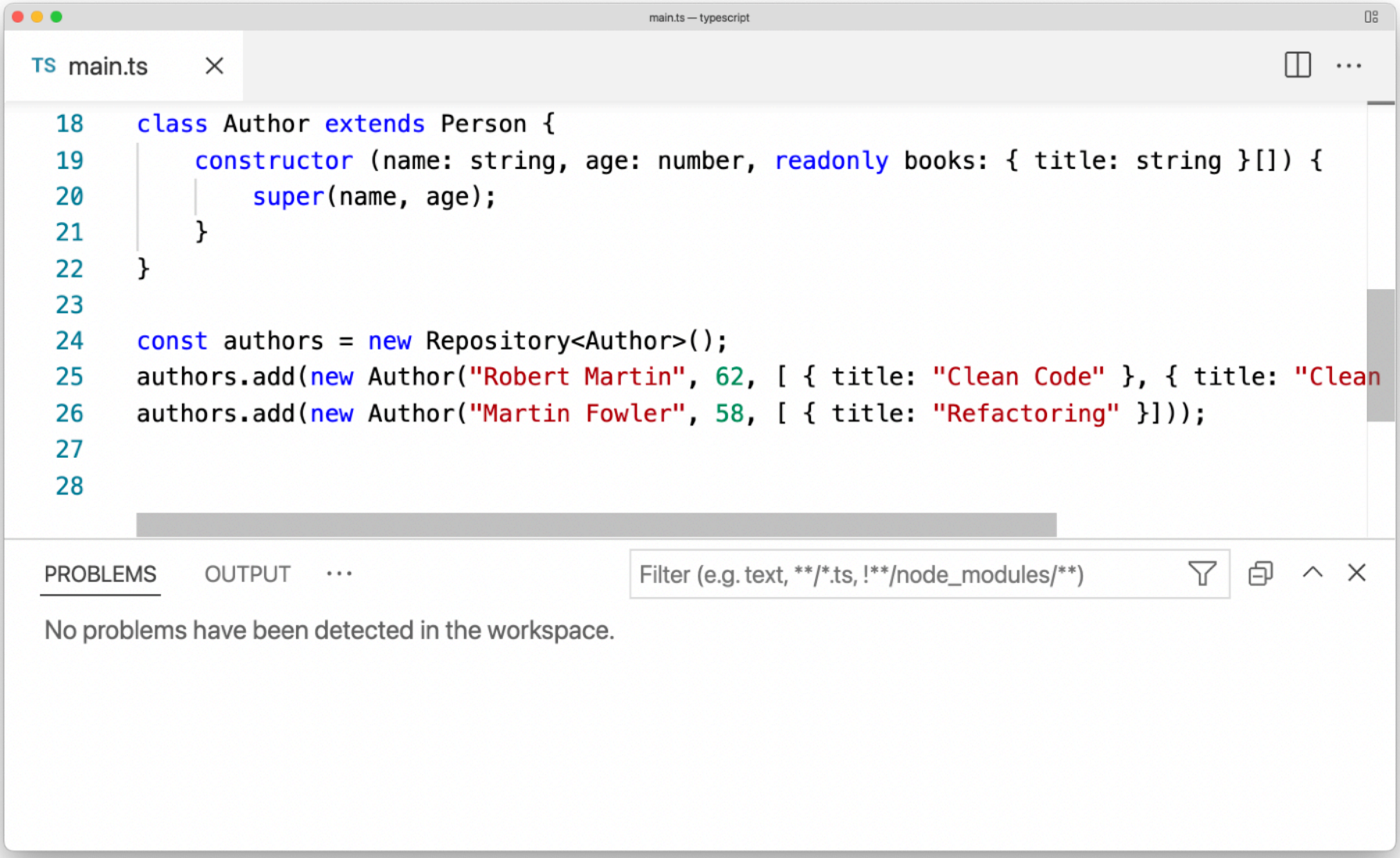
É possível tipar classes permitindo um controle ainda maior sobre a interação com os objetos, inclusive com herança

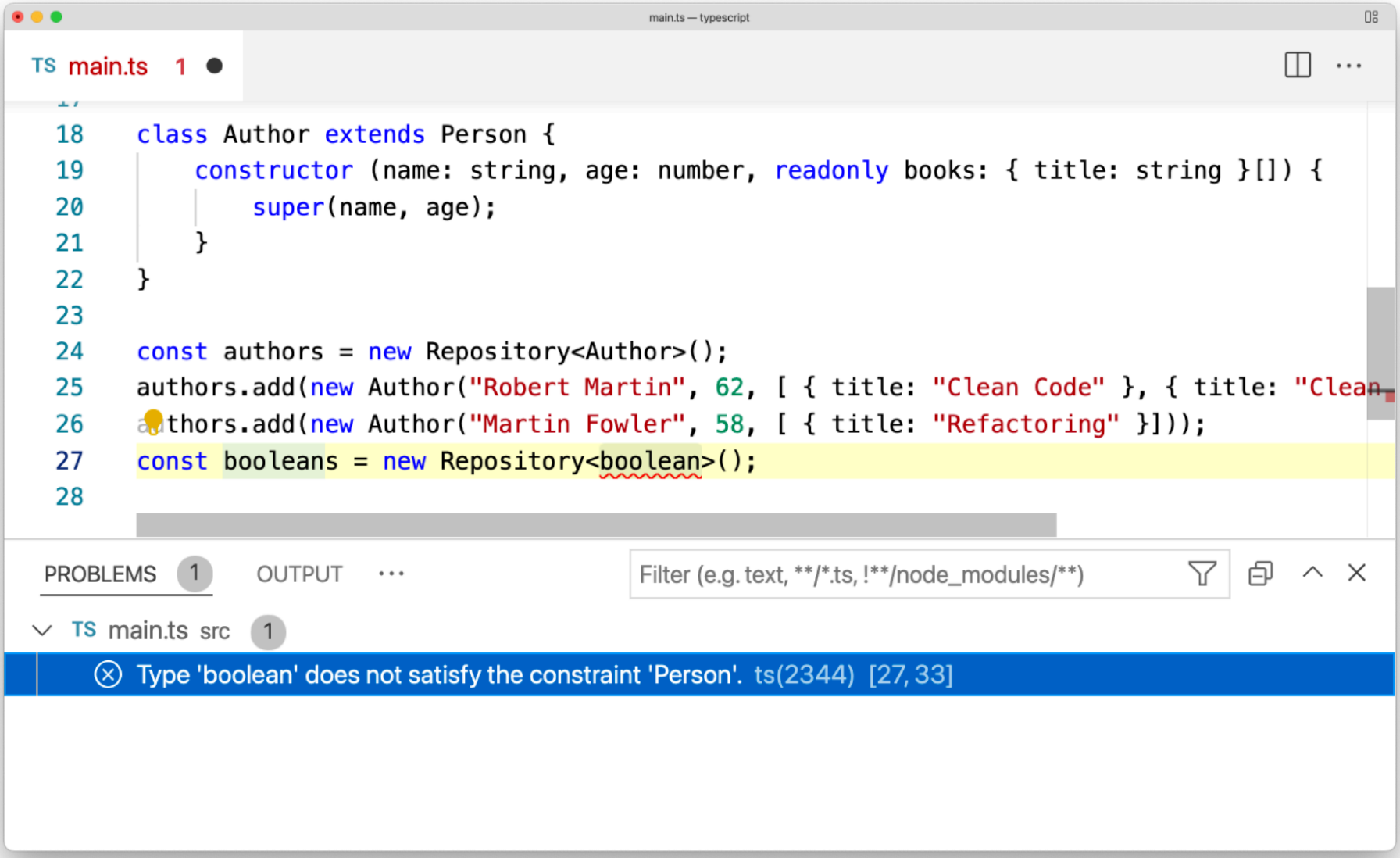

main.ts — typescript
TS main.ts
class Repository {
list: any;
constructor () {
this.list = [];
}
add (element: any) {
this.list.push(element);
}
}
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.


main.ts — typescript
TS main.ts
class Repository<T> {
list: T[];
constructor () {
this.list = [];
}
add (element: T) {
this.list.push(element);
}
}
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.


main.ts — typescript
TS main.ts 1
12
13 class Person {
14 constructor (readonly name: string, readonly age: number) {
15 }
16 }
17
18 const persons = new Repository<Person>();
19 persons.add(new Person("Robert Martin", 62));
20 persons.add(new Person("Martin Fowler", 58));
21 persons.add("JavaScript");
22
PROBLEMS 1
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 1
Argument of type 'string' is not assignable to parameter of type 'Person'. ts(2345) [21, 13]

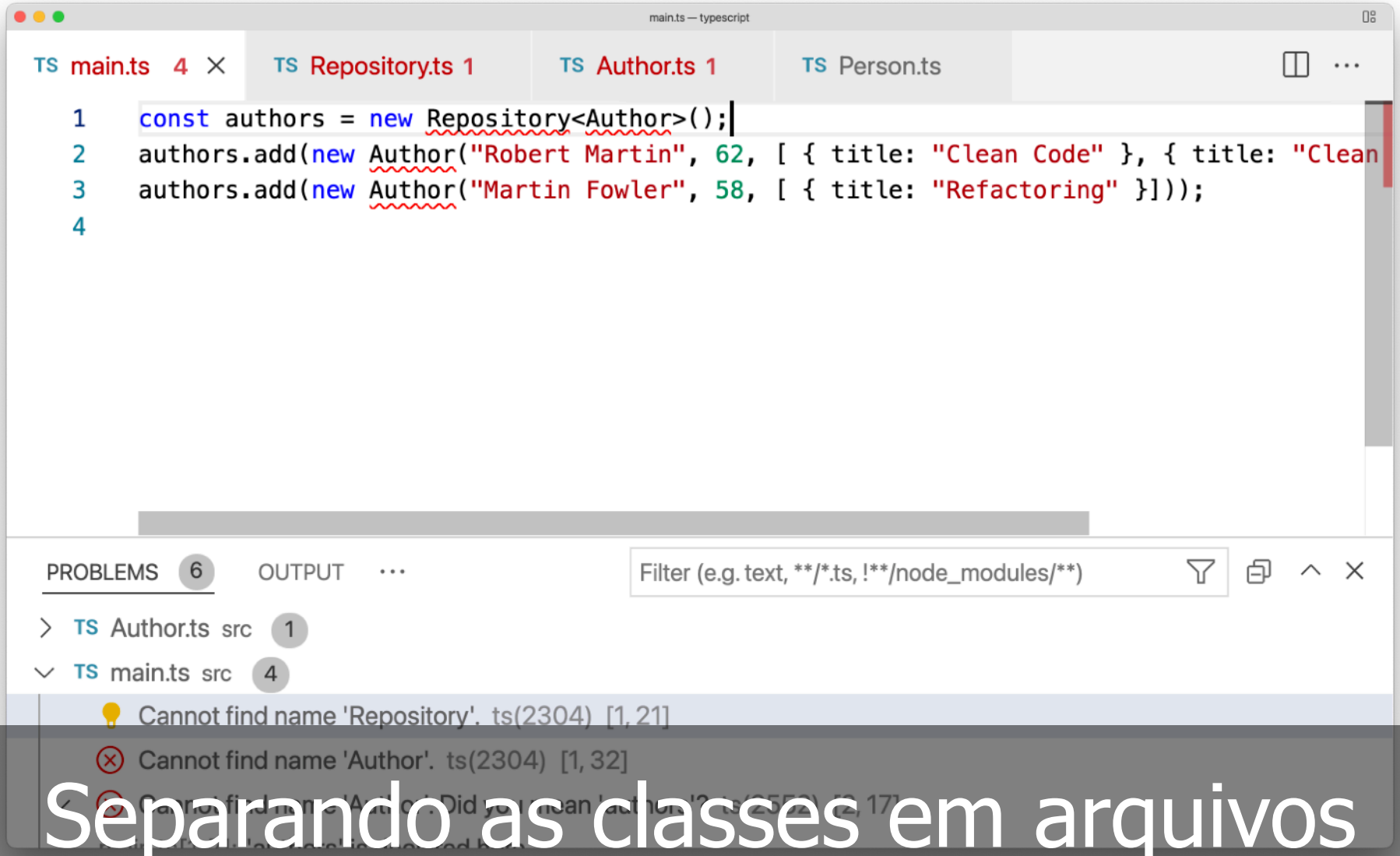

main.ts — typescript
TS main.ts
class Repository<T extends Person> {
list: T[];
constructor () {
this.list = [];
}
add (element: T) {
this.list.push(element);
}
}
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.


main.ts — typescript
TS main.ts
18 class Author extends Person {
19 constructor (name: string, age: number, readonly books: { title: string }[]) {
20 super(name, age);
21 }
22 }
23
24 const authors = new Repository<Author>();
25 authors.add(new Author("Robert Martin", 62, [{ title: "Clean Code" }, { title: "Clean
26 authors.add(new Author("Martin Fowler", 58, [{ title: "Refactoring" }]));
27
28
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.

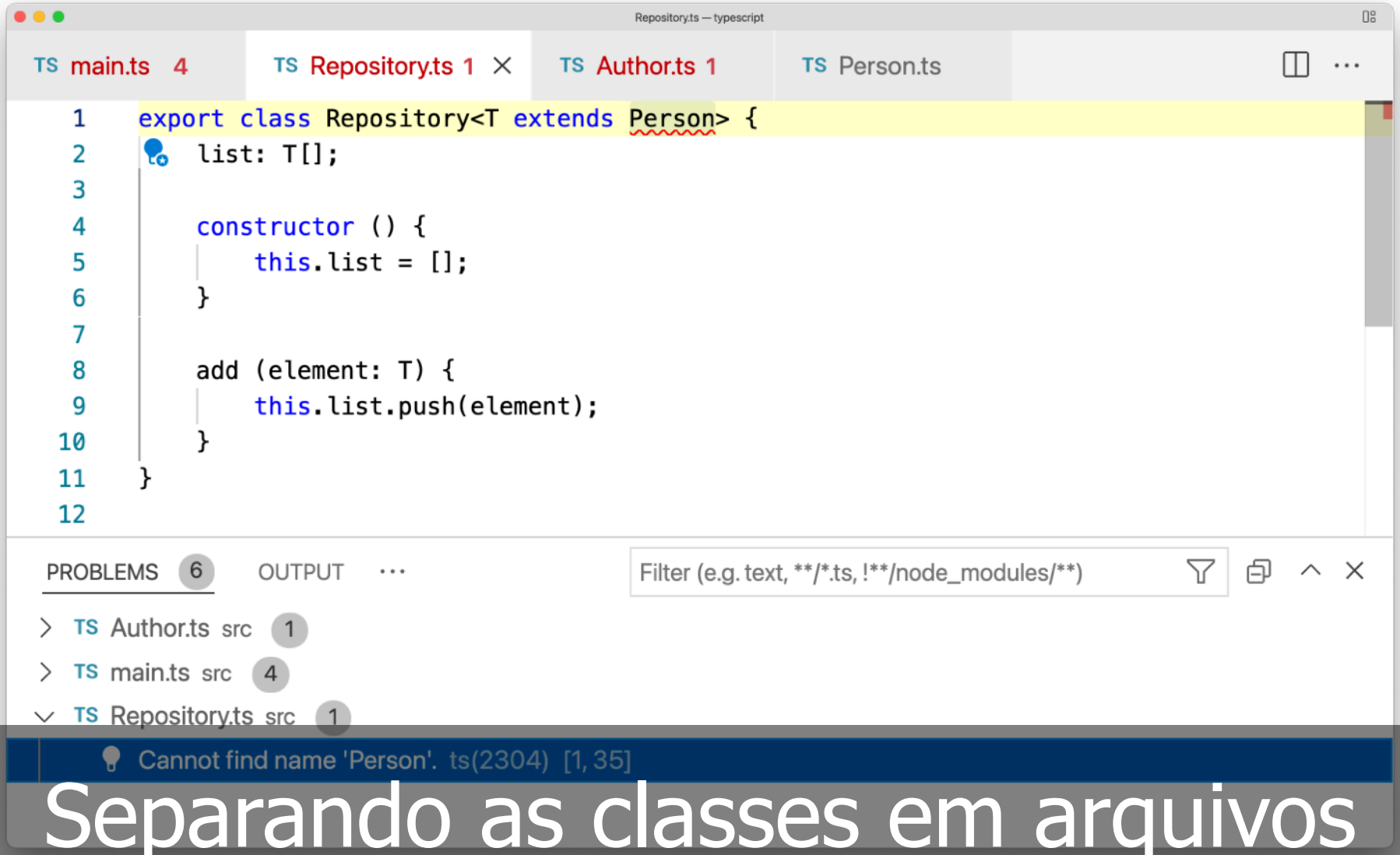

main.ts — typescript
TS main.ts 1
class Author extends Person {
constructor (name: string, age: number, readonly books: { title: string }[]) {
super(name, age);
}
}
const authors = new Repository<Author>();
authors.add(new Author("Robert Martin", 62, [{ title: "Clean Code" }, { title: "Clean
authors.add(new Author("Martin Fowler", 58, [{ title: "Refactoring" }]));
const booleans = new Repository<boolean>();
PROBLEMS 1
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 1
Type 'boolean' does not satisfy the constraint 'Person'. ts(2344) [27, 33]

Modules

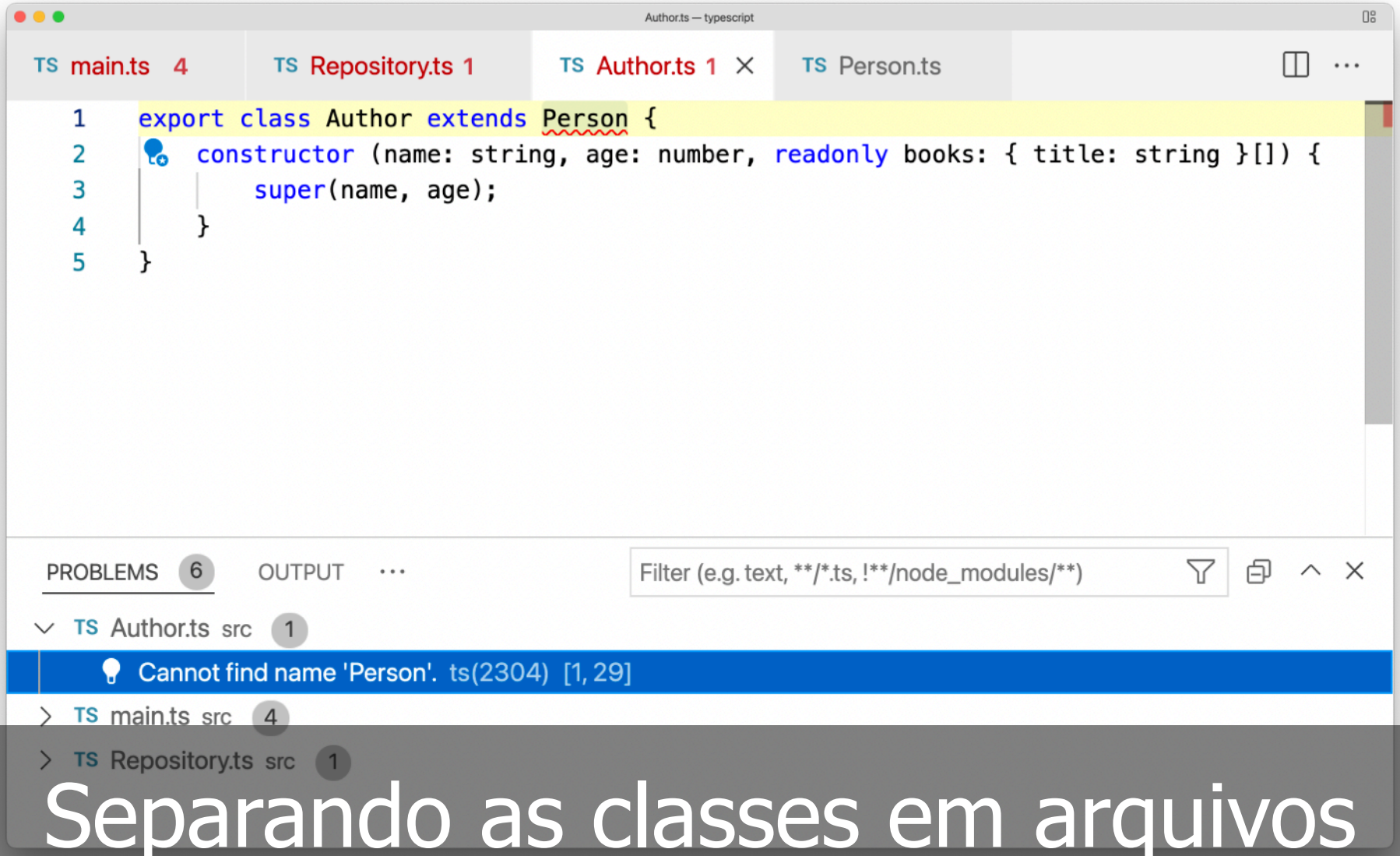
O TypeScript suporta o novo sistema de módulos criado no ES6 por meio das palavras-chave import e export



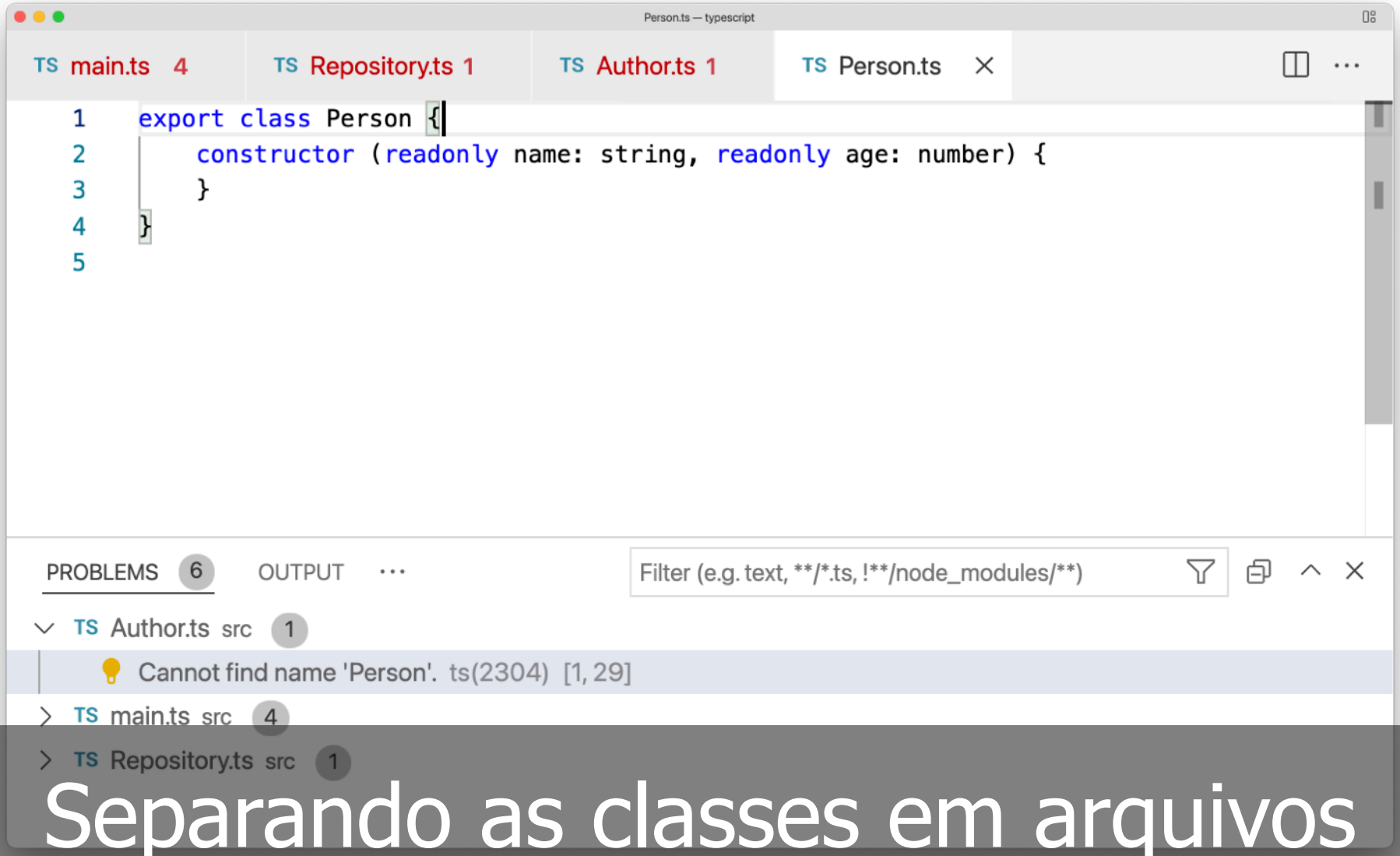
Separando as classes em arquivos diferentes



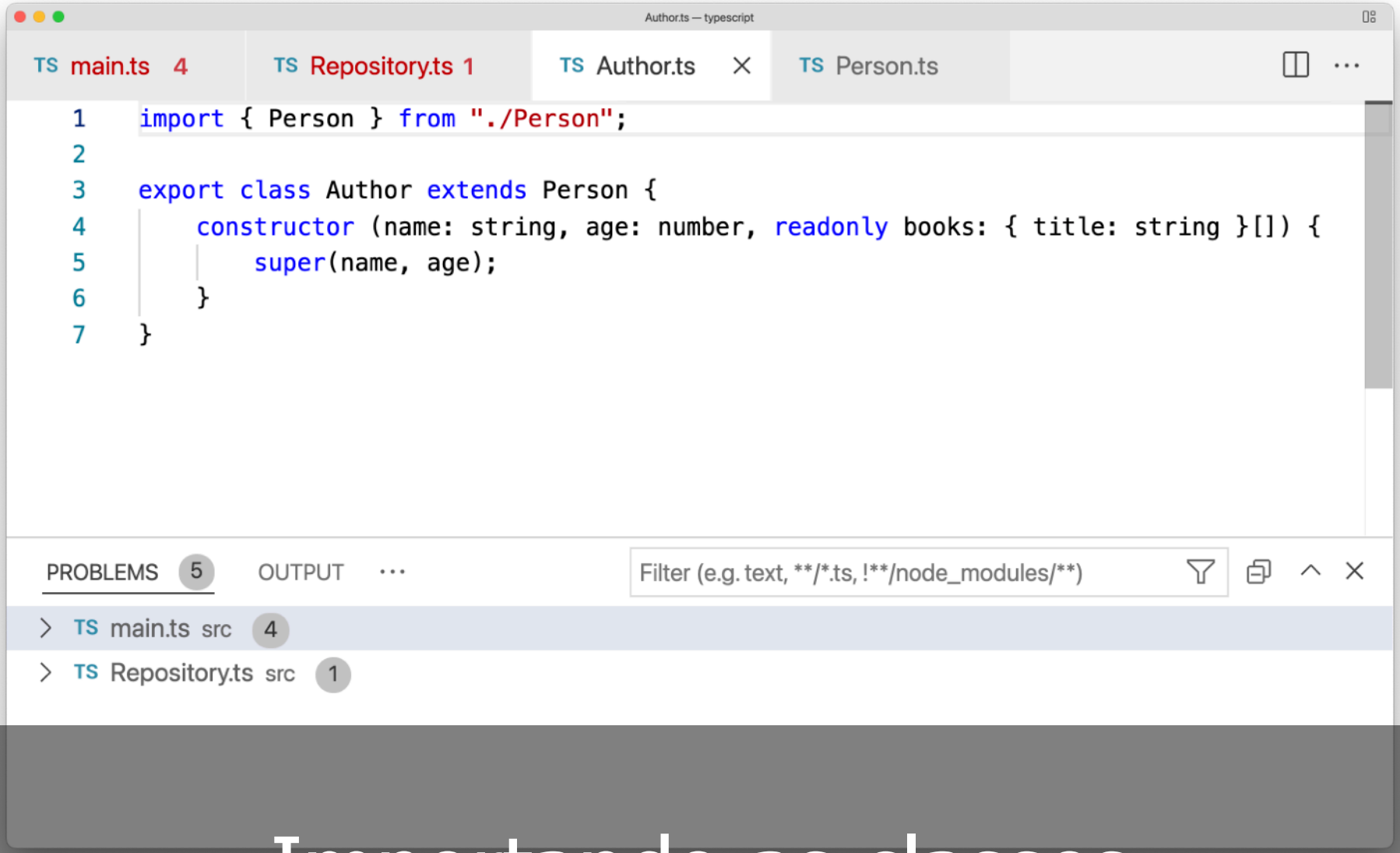
Separando as classes em arquivos diferentes e exportando



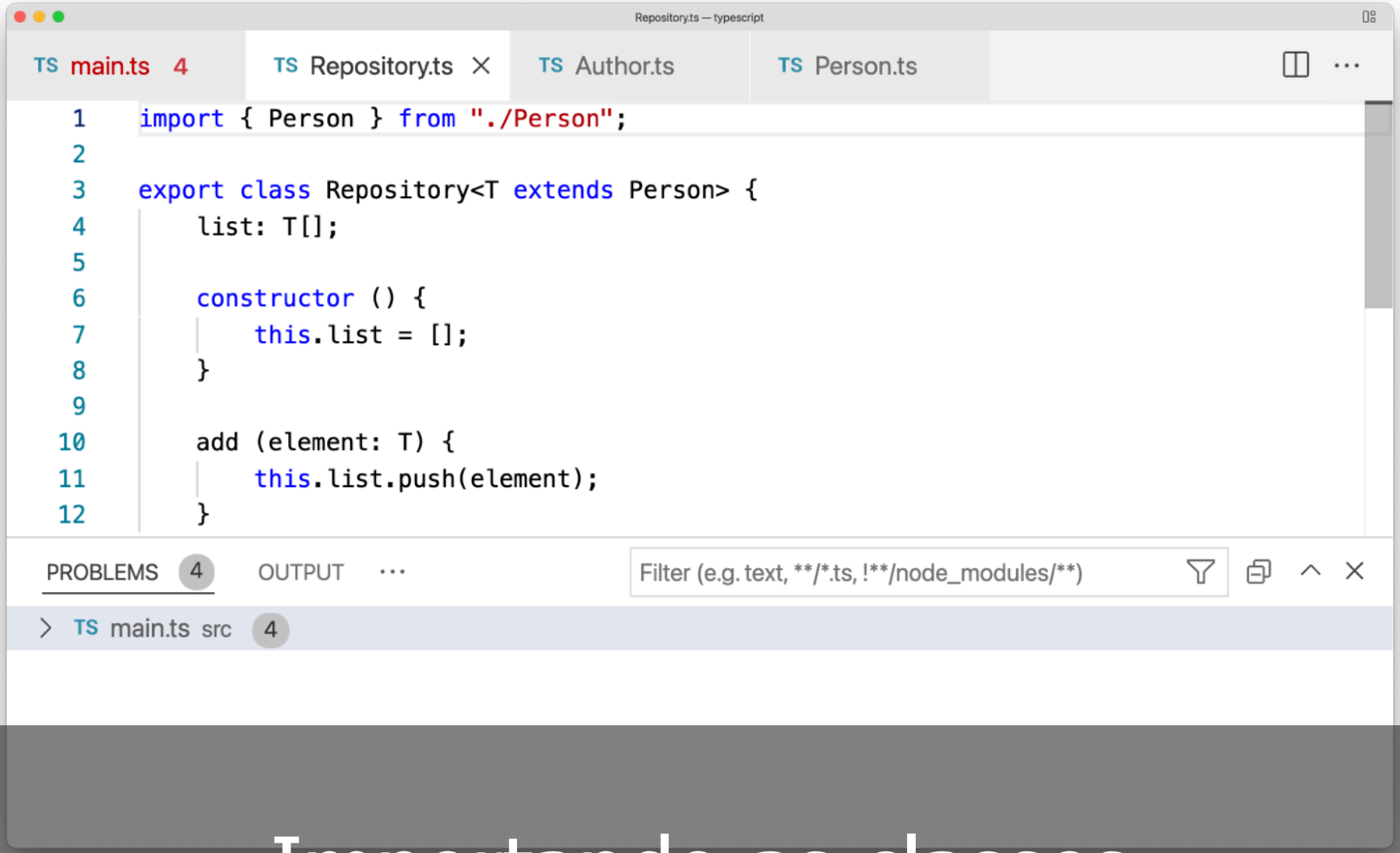
Separando as classes em arquivos diferentes e exportando



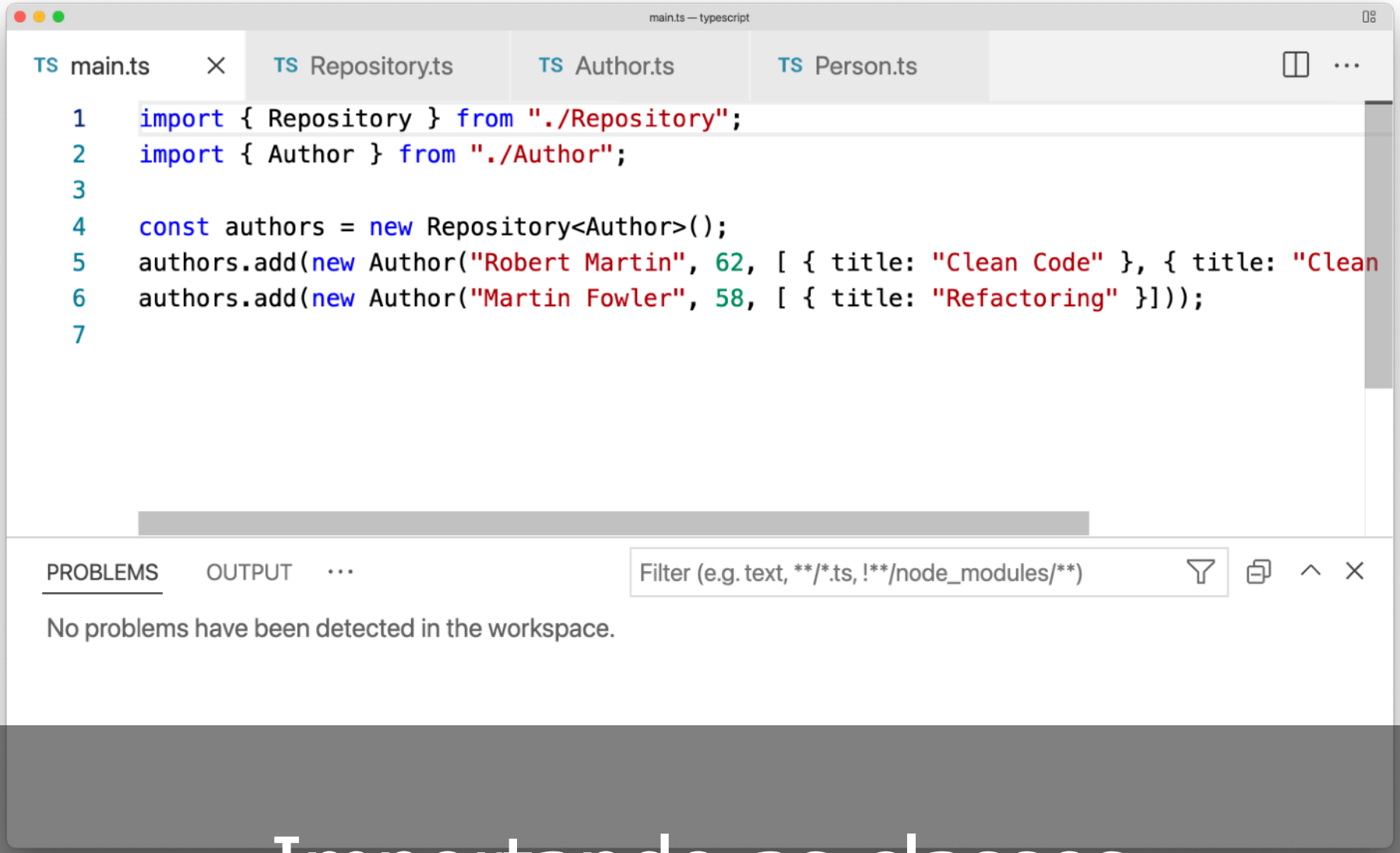
Separando as classes em arquivos diferentes e exportando



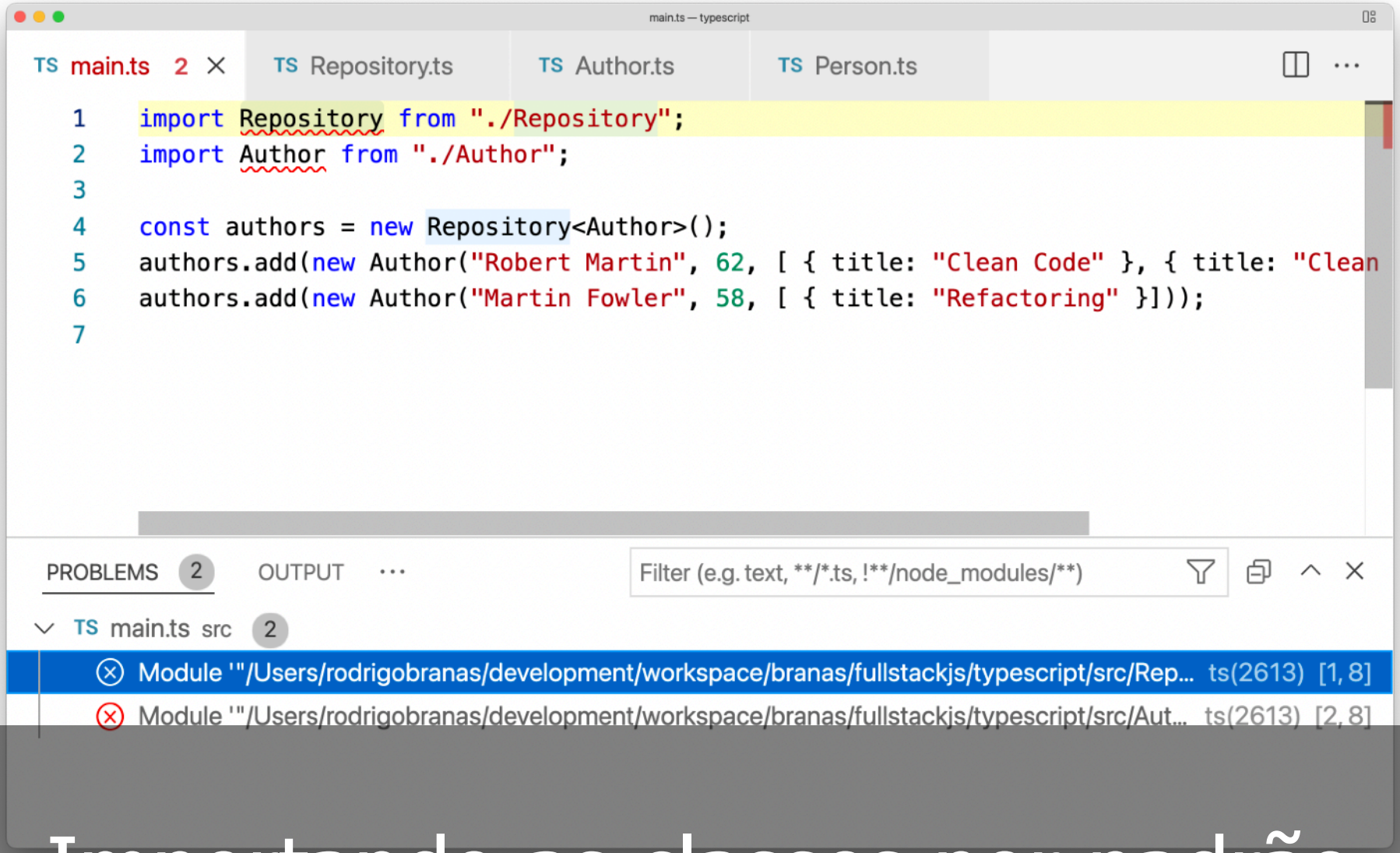
Importando as classes



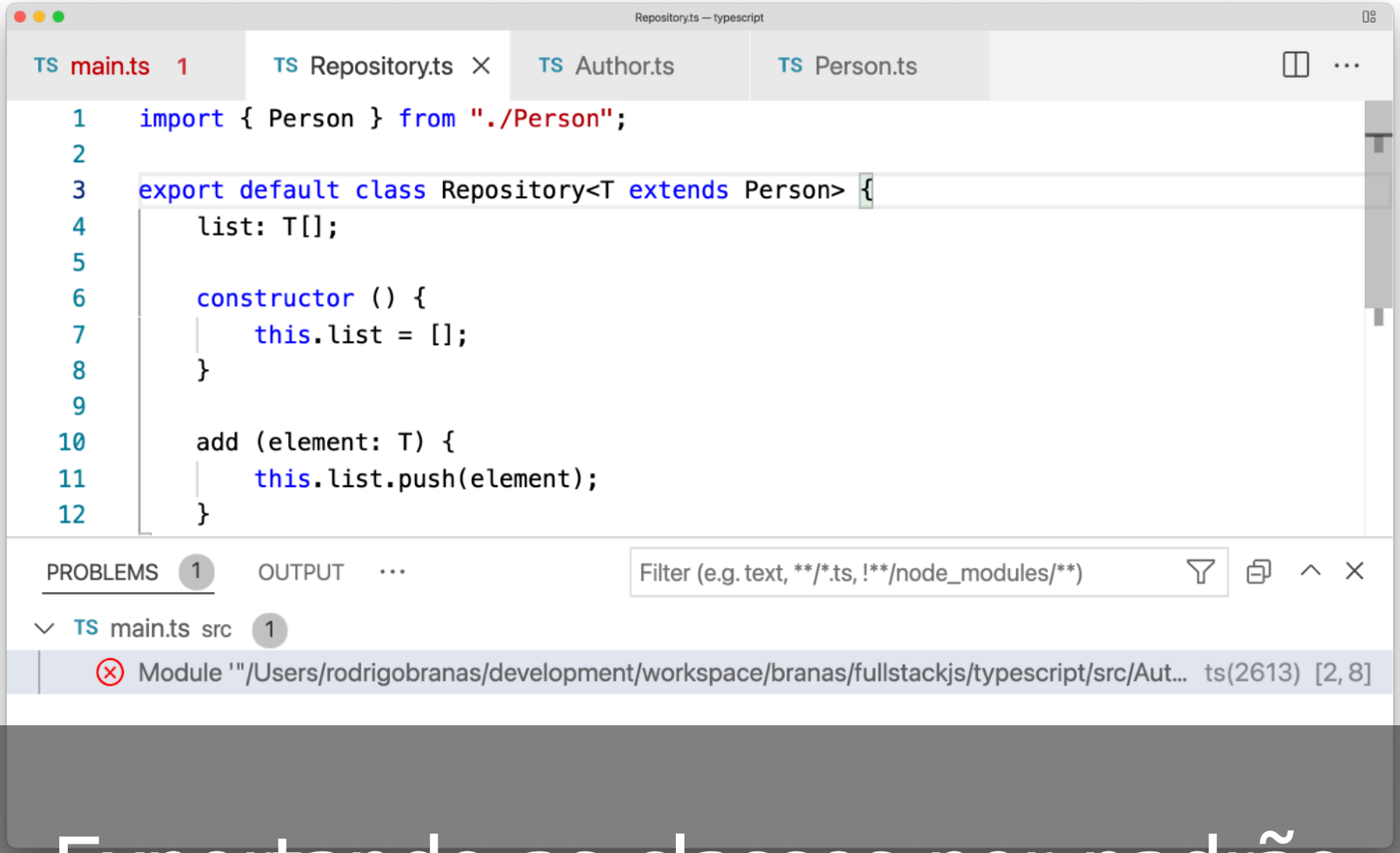
Importando as classes



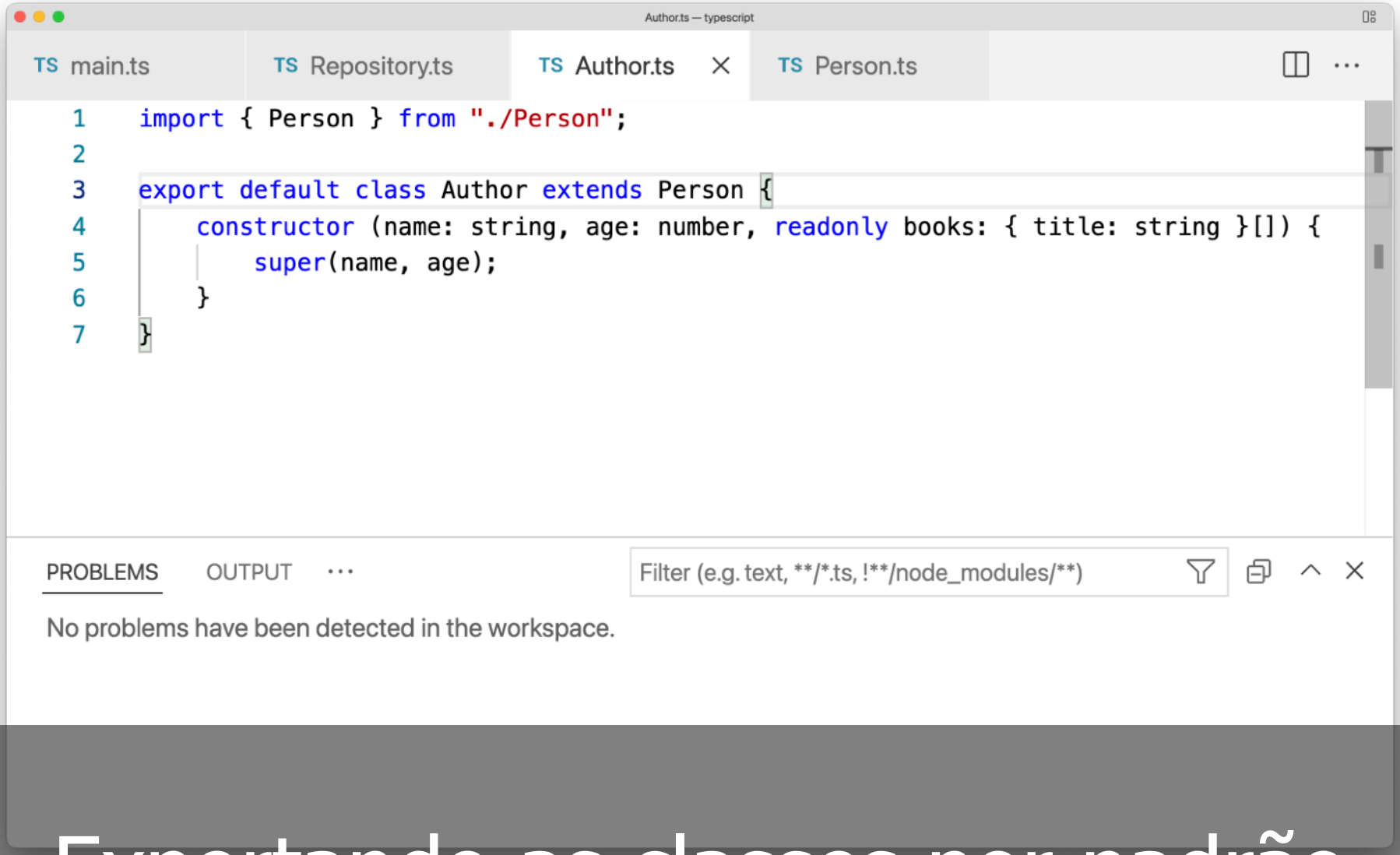
Importando as classes



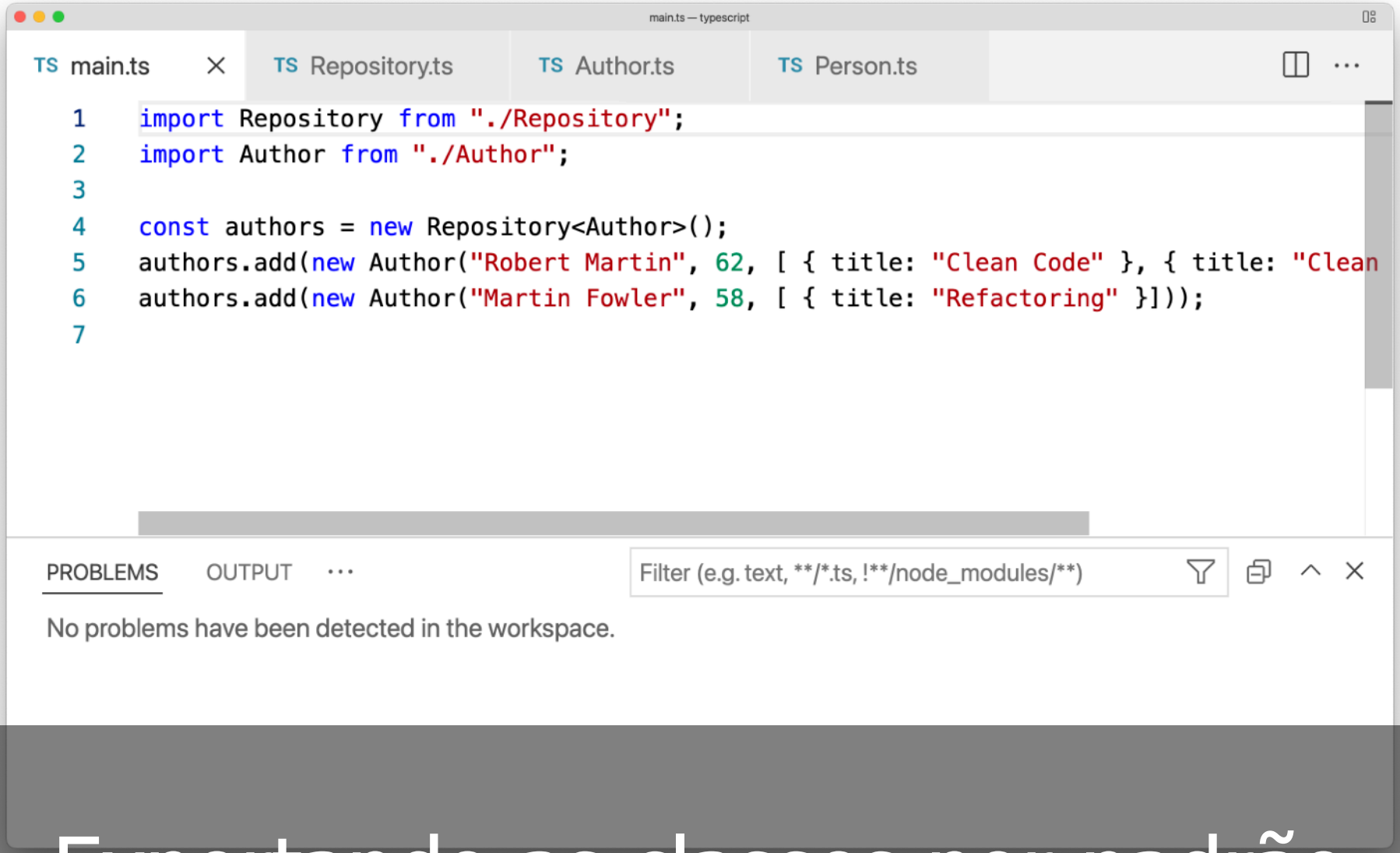
Importando as classes por padrão



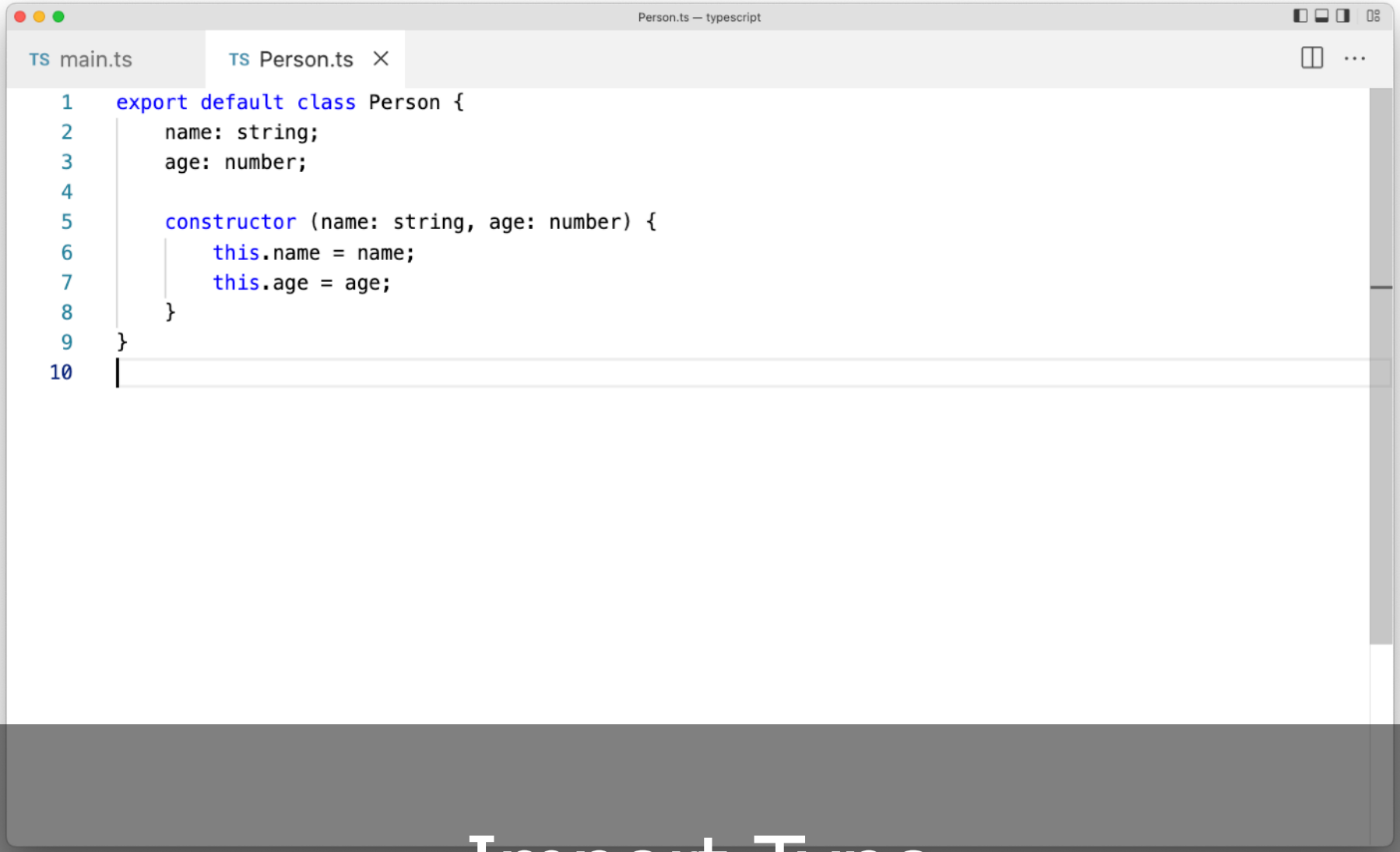
Exportando as classes por padrão



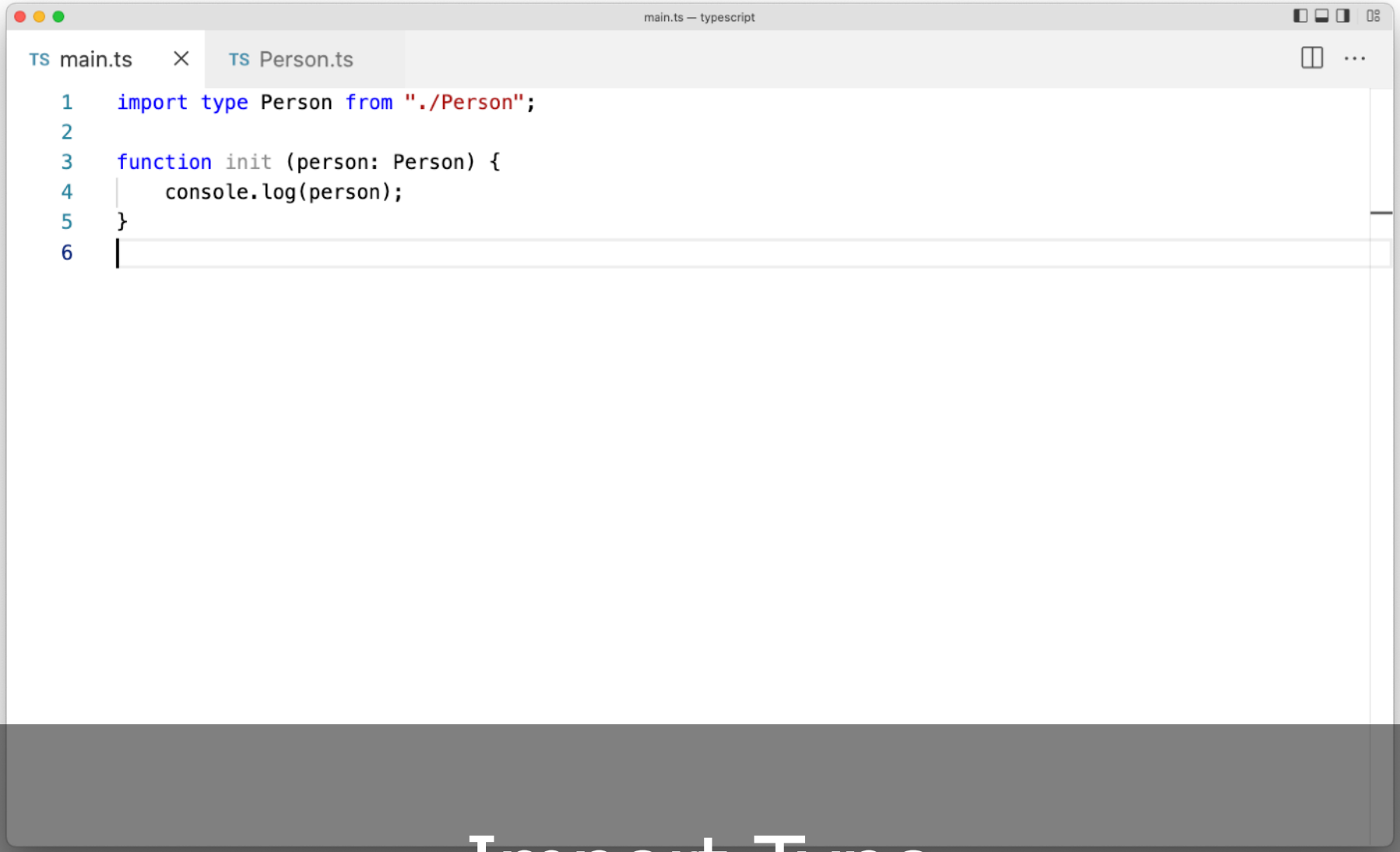
Exportando as classes por padrão



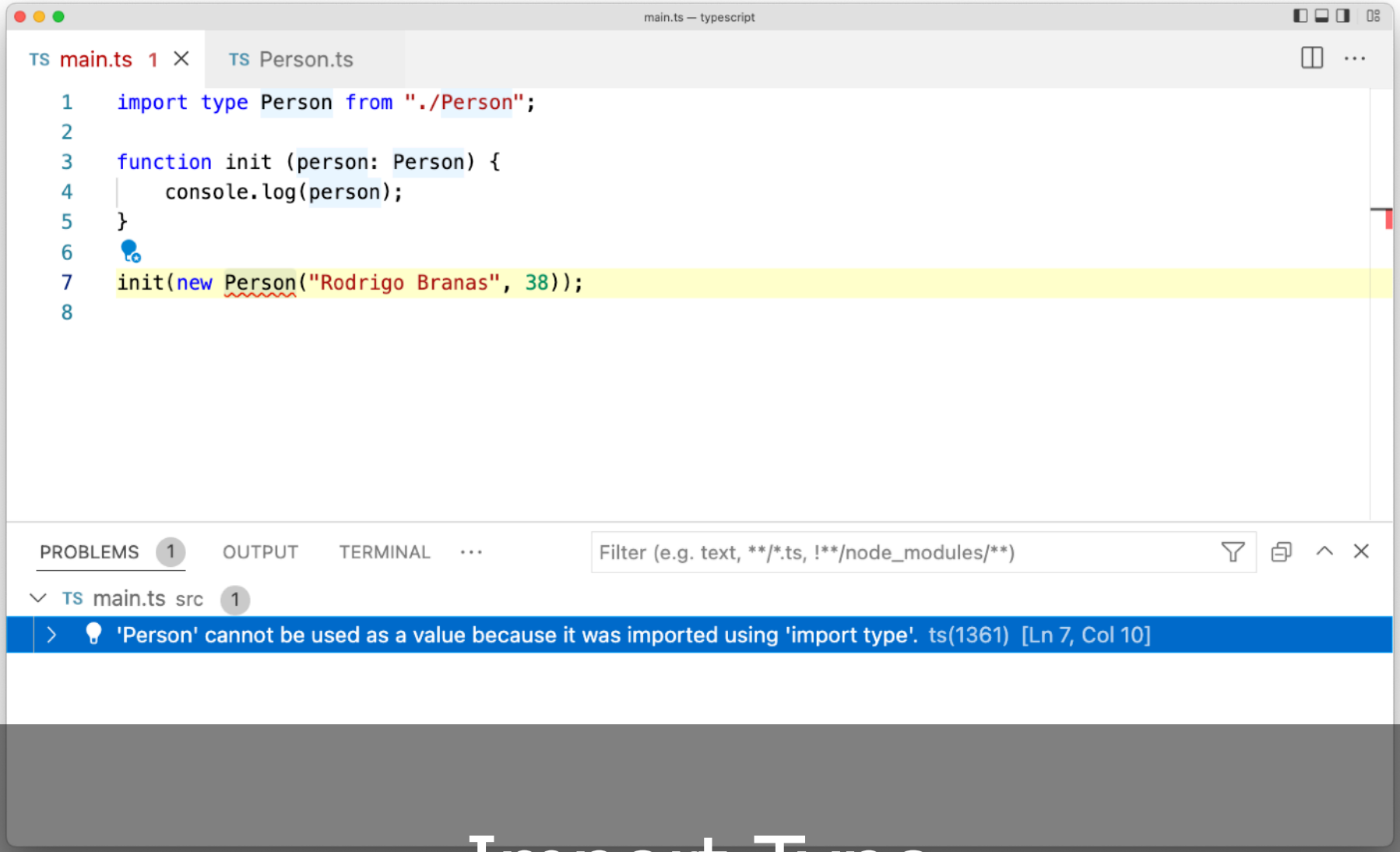
Exportando as classes por padrão


Person.ts — typescript
TS main.ts
TS Person.ts
export default class Person {
name: string;
age: number;
constructor (name: string, age: number) {
this.name = name;
this.age = age;
}
}

Import Type


main.ts — typescript
TS main.ts
TS Person.ts
1 import type Person from "./Person";
2
3 function init (person: Person) {
4 console.log(person);
5 }
6

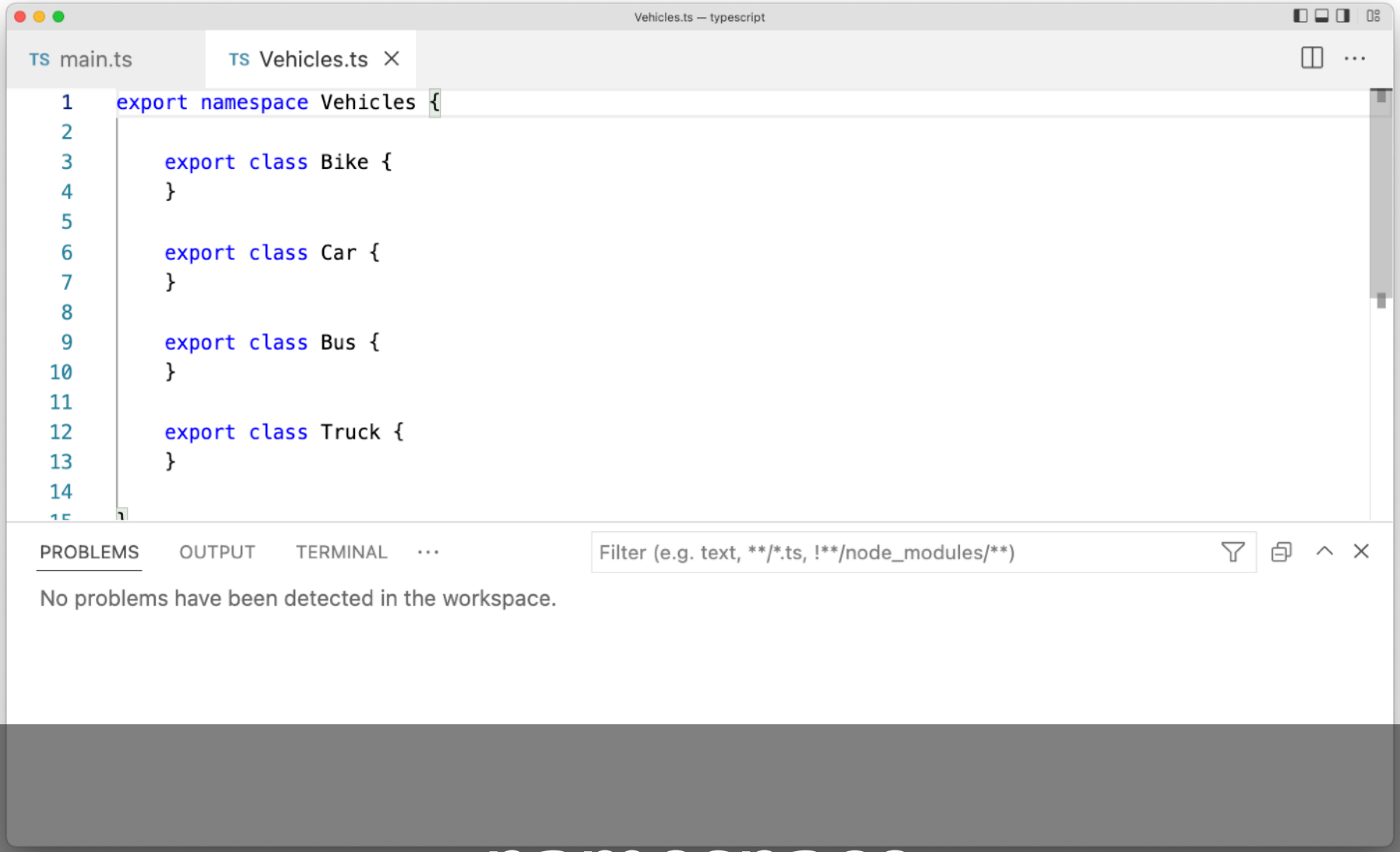
Import Type


main.ts — typescript
TS main.ts 1
TS Person.ts
1 import type Person from "./Person";
2
3 function init (person: Person) {
4 console.log(person);
5 }
6
7 init(new Person("Rodrigo Branas", 38));
8
PROBLEMS 1
OUTPUT
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
TS main.ts src 1
'Person' cannot be used as a value because it was imported using 'import type'. ts(1361) [Ln 7, Col 10]

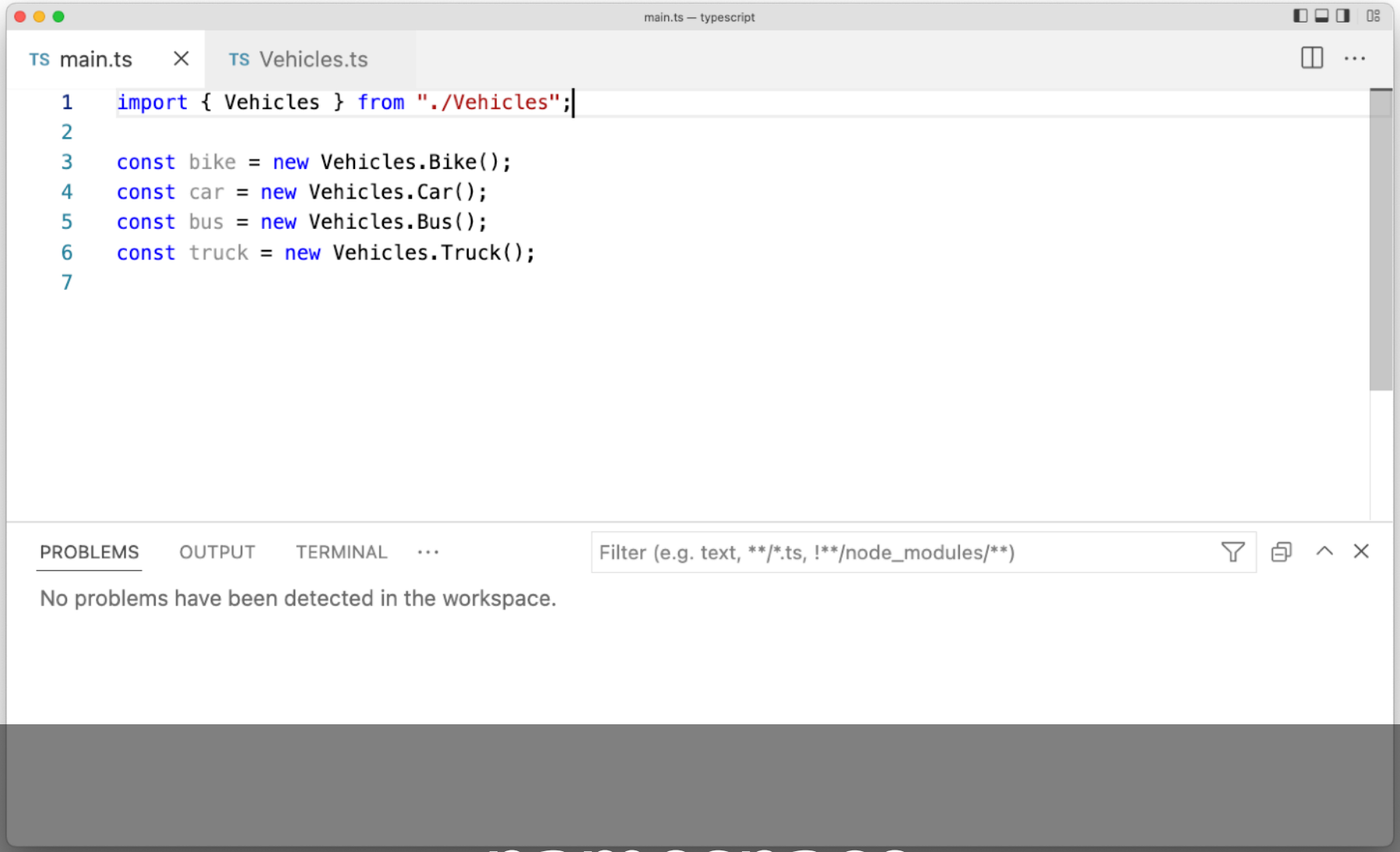
Import Type

Namespace

Um namespace é um basicamente um objeto que atua como uma coleção de variáveis, funções e classes, sendo importante para mediar conflitos de duplicação de nomes, principalmente em linguagens sem modularização


Vehicles.ts — typescript
TS main.ts
TS Vehicles.ts
export namespace Vehicles {
export class Bike {
}
export class Car {
}
export class Bus {
}
export class Truck {
}
PROBLEMS
OUTPUT
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.

namespace


main.ts — typescript
TS main.ts
TS Vehicles.ts
1 import { Vehicles } from "./Vehicles";
2
3 const bike = new Vehicles.Bike();
4 const car = new Vehicles.Car();
5 const bus = new Vehicles.Bus();
6 const truck = new Vehicles.Truck();
7
PROBLEMS
OUTPUT
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.

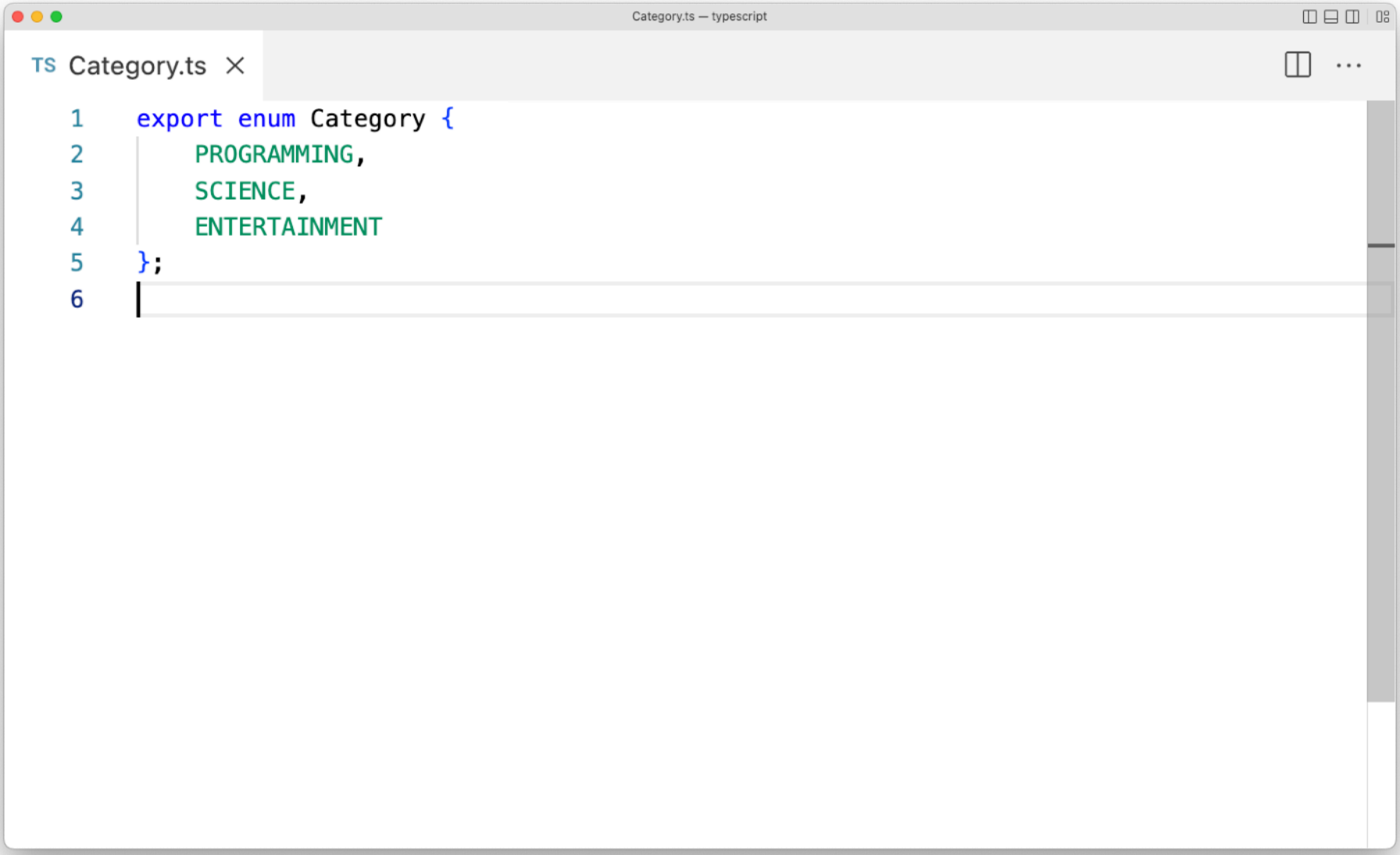
namespace

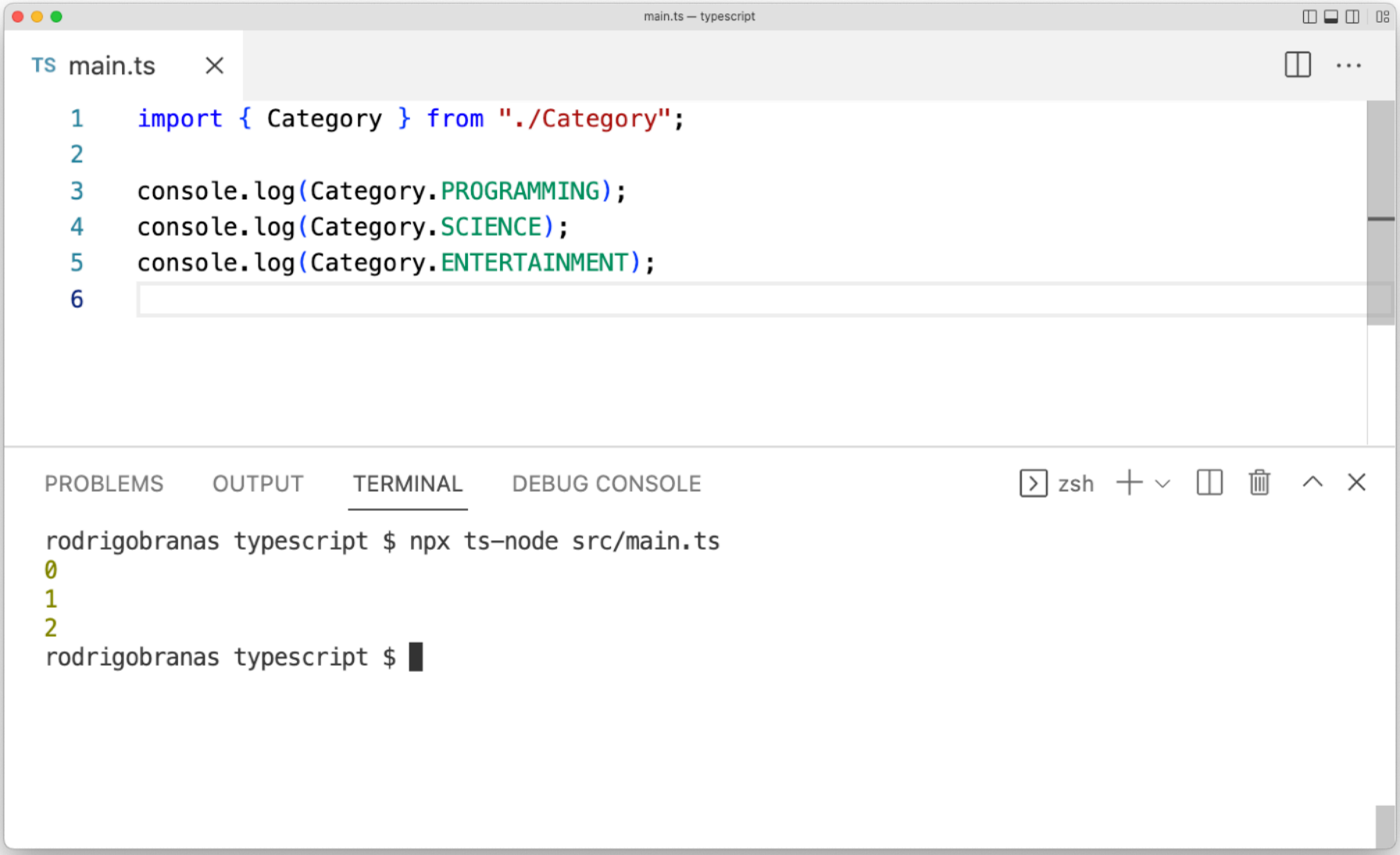
Qual é a diferença entre um module um namespace?

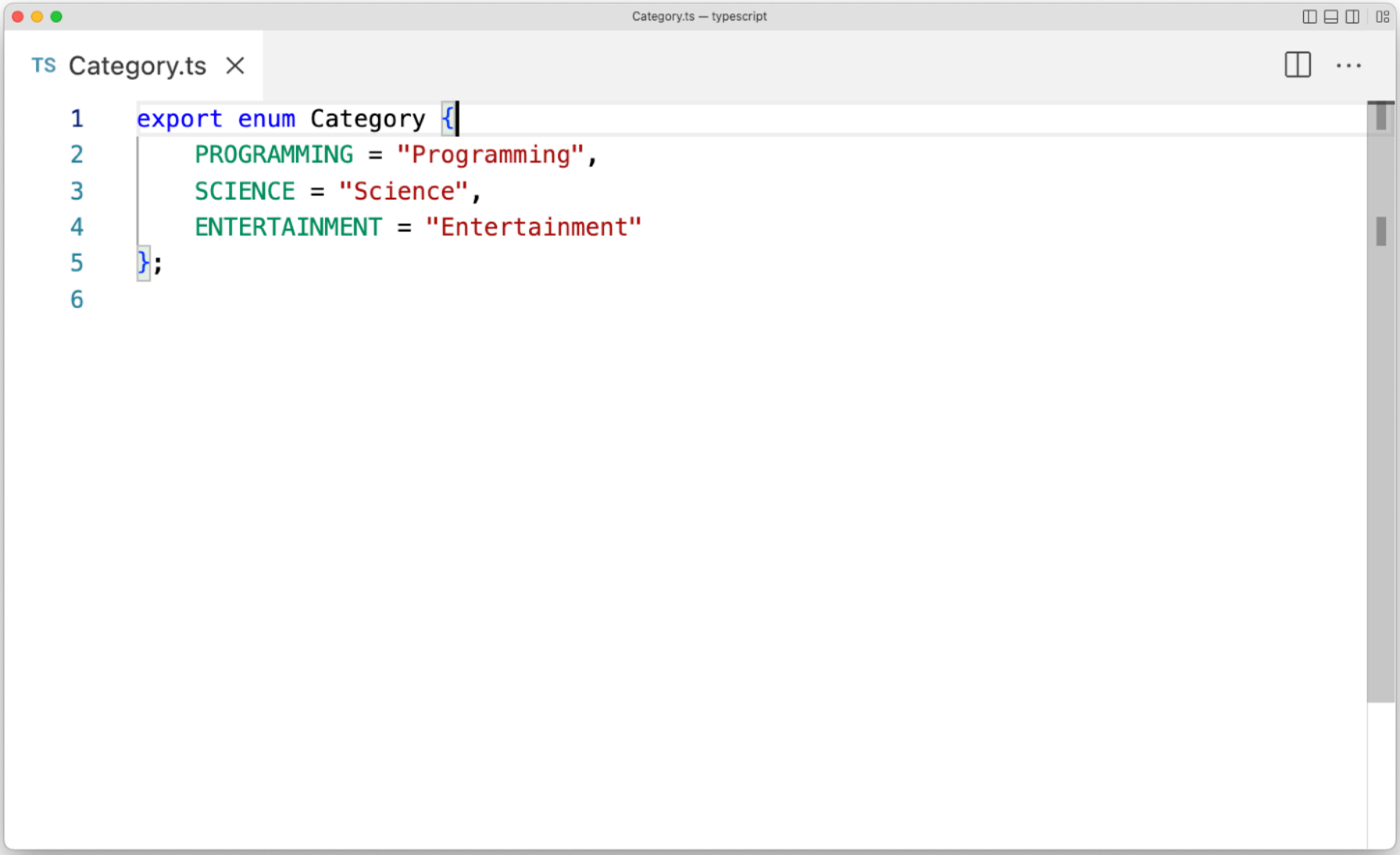
Como modules são padrão no ECMAScript, utilize modules ao invés de namespaces

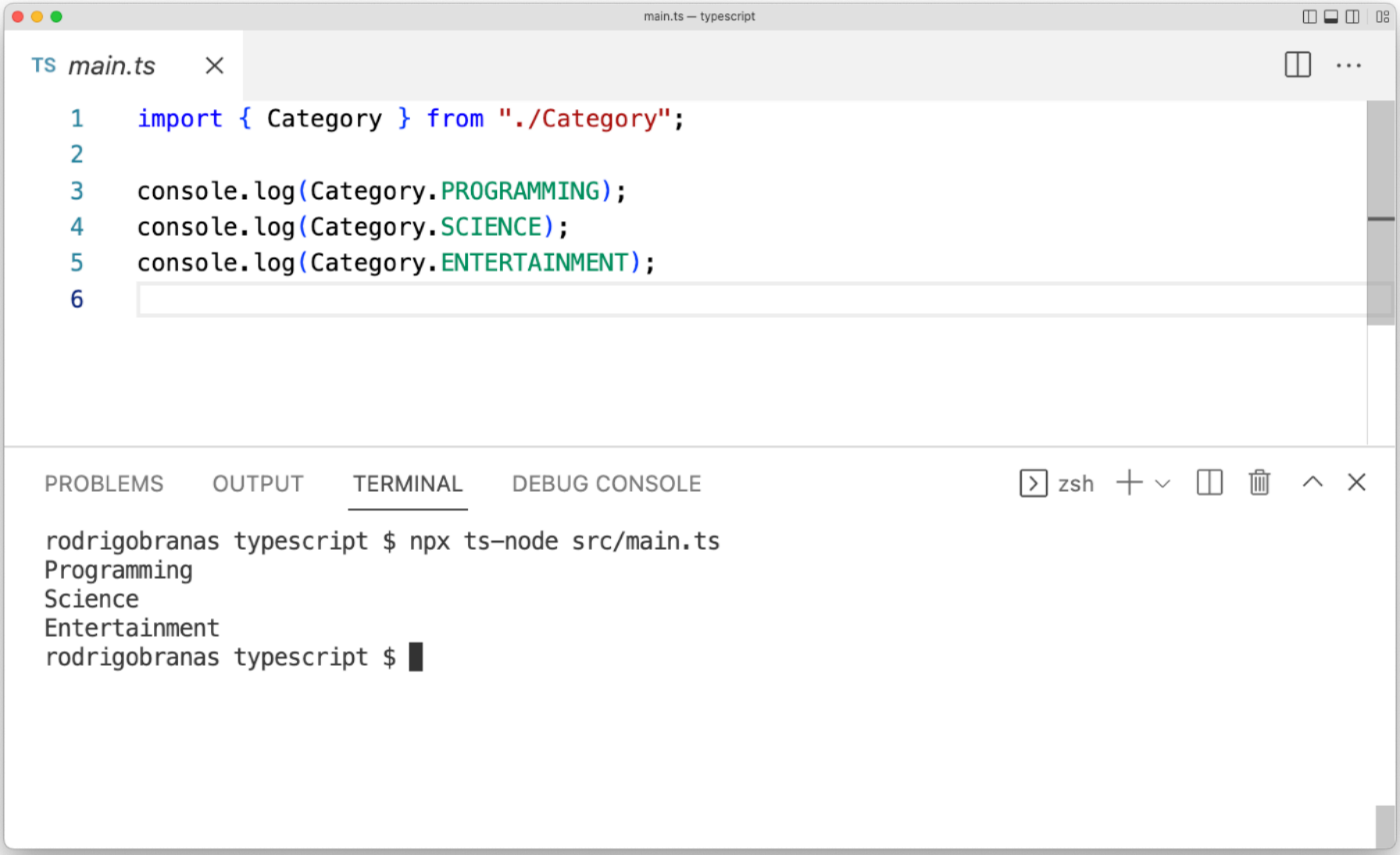
Enums

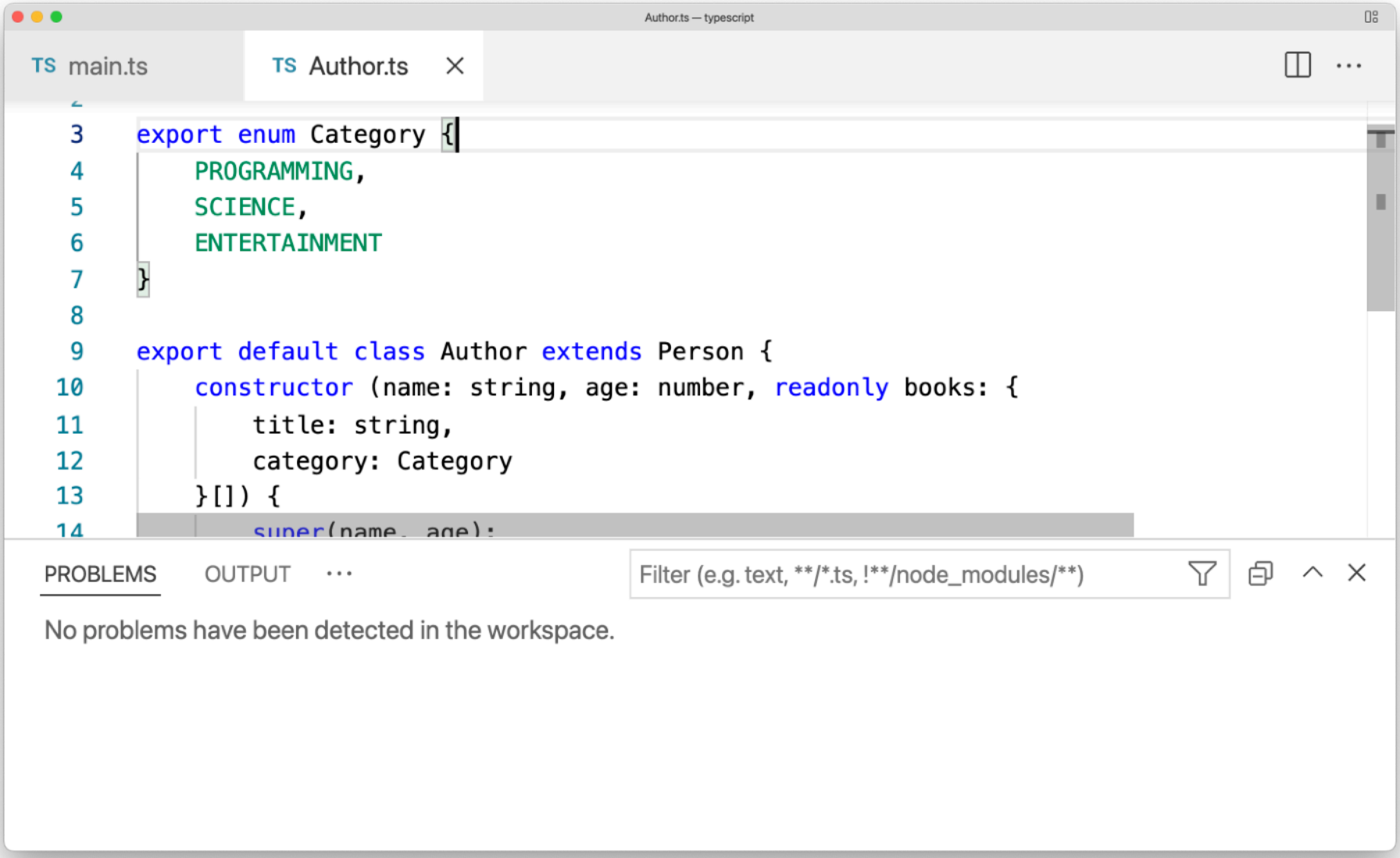
Com os enums é possível definir um conjunto de nomes padronizado e utilizado em uma variável ou propriedade de um objeto

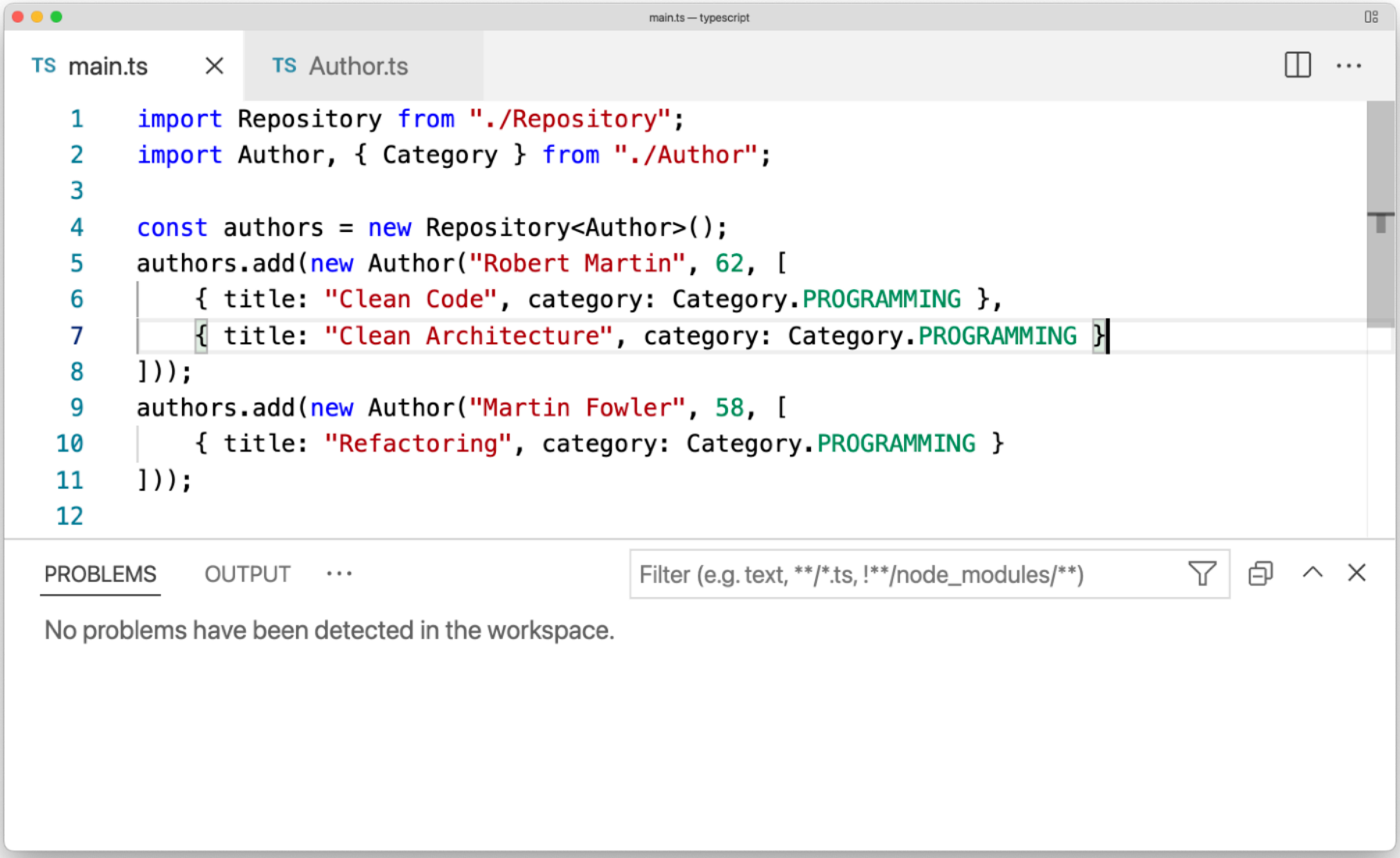

Category.ts — typescript
TS Category.ts
export enum Category {
PROGRAMMING,
SCIENCE,
ENTERTAINMENT
};


main.ts — typescript
TS main.ts
1 import { Category } from "./Category";
2
3 console.log(Category.PROGRAMMING);
4 console.log(Category.SCIENCE);
5 console.log(Category.ENTERTAINMENT);
6
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
zsh
rodrigobranas typescript $ npx ts-node src/main.ts
0
1
2
rodrigobranas typescript $


Category.ts — typescript
TS Category.ts
1 export enum Category {
2 PROGRAMMING = "Programming",
3 SCIENCE = "Science",
4 ENTERTAINMENT = "Entertainment"
5 };
6


main.ts — typescript
TS main.ts
1 import { Category } from "./Category";
2
3 console.log(Category.PROGRAMMING);
4 console.log(Category.SCIENCE);
5 console.log(Category.ENTERTAINMENT);
6
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
zsh
rodrigobranas typescript $ npx ts-node src/main.ts
Programming
Science
Entertainment
rodrigobranas typescript $


Author.ts — typescript
TS main.ts
TS Author.ts
export enum Category {
PROGRAMMING,
SCIENCE,
ENTERTAINMENT
}
export default class Author extends Person {
constructor (name: string, age: number, readonly books: {
title: string,
category: Category
}[]) {
super(name, age);
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.


main.ts — typescript
TS main.ts
TS Author.ts
1 import Repository from "./Repository";
2 import Author, { Category } from "./Author";
3
4 const authors = new Repository<Author>();
5 authors.add(new Author("Robert Martin", 62, [
6 { title: "Clean Code", category: Category.PROGRAMMING },
7 { title: "Clean Architecture", category: Category.PROGRAMMING }
8]));
9 authors.add(new Author("Martin Fowler", 58, [
10 { title: "Refactoring", category: Category.PROGRAMMING }
11]));
12
PROBLEMS
OUTPUT
Filter (e.g. text, **/*.ts, !**/node_modules/**)
No problems have been detected in the workspace.

Decorators

Os decorators são úteis para criar uma orientação à aspectos, de forma mais desacoplada, em classes, métodos, métodos de acesso, propriedades e parâmetros

A opção experimentalDecorators habilita a utilização de decorators

O class decorator é aplicado imediatamente antes da declaração de uma classe e permite interagir com a função construtora da classe

O property decorator é aplicado imediatamente antes da declaração de uma propriedade e permite interagir com a propriedade

O method decorator é aplicado imediatamente antes da declaração de um método e permite interagir com a assinatura do método