



TypeScript

## Utility Types

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

A screenshot of a TypeScript code editor window titled "main.ts – typescript". The code editor displays the following TypeScript code:

```
1  interface Todo {  
2      priority: number,  
3      description: string,  
4      done: boolean  
5  }  
6  
7  const todo: Todo = {  
8      priority: 1,  
9      description: "Limpar a casa",  
10     done: false  
11 }  
12
```

The "todo" object has a red underline under its "description" property, indicating a type error. The "description" property is typed as "string", but its value is a string containing spaces, which is not a valid TypeScript string literal.

Below the code editor, the "PROBLEMS" tab is selected in the bottom navigation bar. The status bar below the navigation bar shows the message "No problems have been detected in the workspace."

# Partial<Type>

The screenshot shows a code editor window titled "main.ts — typescript". The code is as follows:

```
12
13     function updateTodo (updatedTodo: Todo) {
14         return {...todo, ...updatedTodo };
15     }
16     ^
17     updateTodo({
18         priority: 2,
19         description: "Limpar a casa e a garagem"
20     });
21
22
23
```

Line 17 contains a call to `updateTodo` with an argument object. The properties `priority` and `description` are underlined with red squiggly lines, indicating a TypeScript error. A yellow lightbulb icon is positioned above the opening brace of the argument object.

Below the code editor, the VS Code interface includes:

- PROBLEMS tab (1)
- OUTPUT tab
- ... button
- Filter input field (e.g. text, \*\*/\*.ts, \*\*/node\_modules/\*\*)
- Filter icon
- Close icon

The PROBLEMS tab shows one error, which is expanded to show the following message:

```
> ⚠ Argument of type '{ priority: number; description: string; }' is not assignable to parameter... ts(2345) [17, 12]
```

# Partial<Type>

A screenshot of a code editor window titled "main.ts — typescript". The code editor displays the following TypeScript code:

```
12
13     function updateTodo (updatedTodo: Partial<Todo>) {
14         return {...todo, ...updatedTodo };
15     }
16
17     updateTodo({
18         priority: 2,
19         description: "Limpar a casa e a garagem"
20     });
21
22
23
```

The line "description: "Limpar a casa e a garagem"" is highlighted in red, indicating a syntax error. The code editor interface includes tabs for "PROBLEMS", "OUTPUT", and "...". Below the tabs, a message states "No problems have been detected in the workspace."

# Partial<Type>

The screenshot shows a code editor window titled "main.ts — typescript". The code in the editor is:

```
12
13     function updateTodo (updatedTodo: Pick<Todo, "done">) {
14         return {...todo, ...updatedTodo };
15     }
16
17     updateTodo({
18         done: true
19 });
20
21
22
23
```

The word "done" is highlighted in red, indicating a potential error or warning. The code uses the `Pick<Type, ...>` utility type from the TypeScript standard library.

Below the editor, the "PROBLEMS" tab is selected, showing the message: "No problems have been detected in the workspace."

# Pick<Type, ...>

The screenshot shows a code editor window titled "main.ts — typescript". The code is as follows:

```
const todo: Todo = {
    priority: 1,
    description: "Limpar a casa",
    done: false
}

function updateTodo (updatedTodo: Omit<Todo, "priority" | "description">) {
    return {...todo, ...updatedTodo };
}

updateTodo({
    done: true
})
```

The "PROBLEMS" tab is selected, showing the message: "No problems have been detected in the workspace."

# Omit<Type, ...>

A screenshot of a TypeScript code editor window titled "main.ts – typescript". The code editor displays the following TypeScript code:

```
1  interface Todo {  
2      priority?: number,  
3      description: string,  
4      done: boolean  
5  }  
6  
7  const todo: Todo = {  
8      description: "Limpar a casa",  
9      done: false  
10 }  
11  
12
```

The code editor interface includes tabs for "PROBLEMS", "OUTPUT", and "...". Below the tabs, a message states "No problems have been detected in the workspace." The bottom right corner of the editor has icons for filtering, expanding, and closing.

# Required<Type>

The screenshot shows a Microsoft Visual Studio Code interface. The main editor window displays a file named `main.ts` with the following content:

```
1 interface Todo {
2     priority?: number,
3     description: string,
4     done: boolean
5 }
6
7 const todo: Required<Todo> = {
8     description: "Limpar a casa",
9     done: false
10}
11
12
```

A yellow status bar highlights the line `const todo: Required<Todo> = {`. The code editor has syntax highlighting for TypeScript, with `Todo` and `Required` being blue, and `number`, `string`, and `boolean` being grey. A small blue circular icon with a white question mark is positioned next to the `todo` declaration.

Below the editor, the VS Code interface includes:

- A navigation bar with tabs: PROBLEMS (highlighted), OUTPUT, and ...
- A search bar labeled "Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)" with a magnifying glass icon.
- A sidebar with a tree view showing `TS main.ts src` with 1 error.
- A bottom status bar showing the error message: "Property 'priority' is missing in type '{ description: string; done: false; }' but required in type 'Re... ts(2741) [7, 7]"

# Required<Type>

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor window.

The terminal window at the bottom has the following text:

```
MacBook-Pro:~ user$ cd /Users/user/Desktop
MacBook-Pro:Desktop user$ ls
main.ts  node_modules  package-lock.json  package.json
```

The code editor window is titled "main.ts – typescript". It contains the following TypeScript code:

```
1 interface Todo {
2     priority?: number,
3     description: string,
4     done: boolean
5 }
6
7 const todo: Todo = {
8     description: "Limpar a casa",
9     done: false
10}
11
12 todo.done = true;
```

The code editor shows syntax highlighting for TypeScript, including blue for keywords and red for strings. The status bar at the bottom of the code editor window displays "PROBLEMS", "OUTPUT", and "...".

# Readonly<Type>

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface running on a Mac. The title bar says "main.ts — typescript". The code editor window contains the following TypeScript code:

```
TS main.ts 1 ●
```

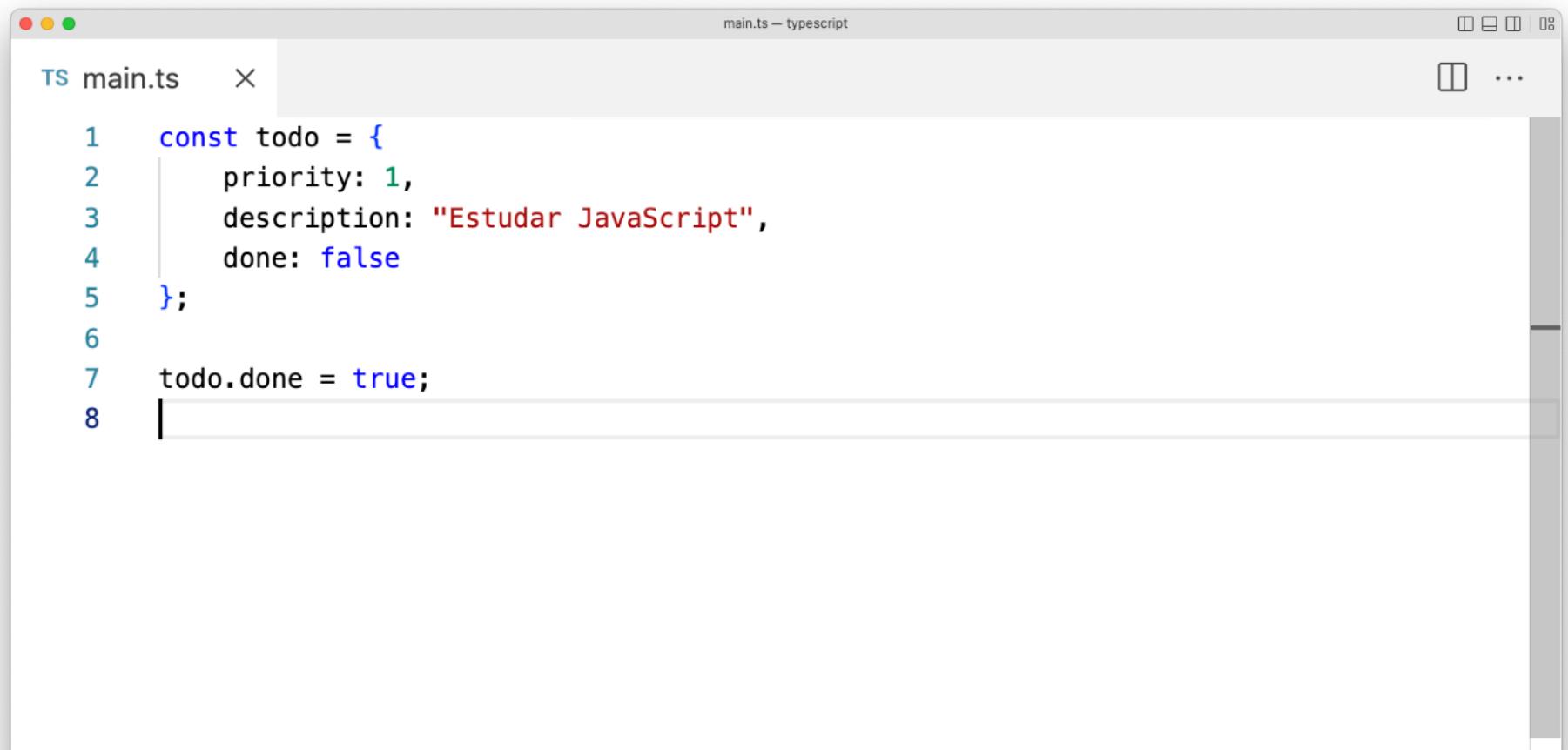
```
1 interface Todo {  
2     priority?: number,  
3     description: string,  
4     done: boolean  
5 }  
6  
7 const todo: Readonly<Todo> = {  
8     description: "Limpiar a casa",  
9     done: false  
10 }  
11  
12 todo.done = true;
```

The line "todo.done = true;" is highlighted in yellow, indicating it's the current line of interest. A red squiggle underlines the word "done" in the line above, indicating a TypeScript error. The status bar at the bottom shows "PROBLEMS 1" and "OUTPUT 1". The Problems panel displays a single error message:

Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

✖ Cannot assign to 'done' because it is a read-only property. ts(2540) [12, 6]

# Readonly<Type>



```
main.ts — typescript
TS main.ts ×
```

```
1 const todo = {
2   priority: 1,
3   description: "Estudar JavaScript",
4   done: false
5 };
6
7 todo.done = true;
8 |
```

Uma outra opção é utilizar o  
**as const**

The screenshot shows a VS Code interface with the following details:

- Title Bar:** main.ts — typescript
- Code Editor:** A file named "main.ts" containing the following TypeScript code:

```
1 const todo = {
2     priority: 1,
3     description: "Estudar JavaScript",
4     done: false
5 } as const;
6
7 todo.done = true;
```
- Problems Panel:** Located at the bottom left, it shows 1 error for "main.ts".
  - Filter:** Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)
  - Errors:** TS main.ts src 1
    - ⊗ Cannot assign to 'done' because it is a read-only property. ts(2540) [Ln 7, Col 6]

Com o `as const` o objeto também se torna **imutável**, somente leitura

Existem diversos outros utility types como **Exclude**,  
**Extract**, **NonNullable**, **Parameters** entre outros

## Type Declaration

O TypeScript tem dois tipos de arquivos, .ts e .d.ts sendo que este é utilizado para a declaração de tipos, fundamental na interoperabilidade com dependências na linguagem JavaScript

## JavaScript Support

O grupo de configurações JavaScript Support é utilizado para **habilitar e validar a interoperabilidade com a linguagem JavaScript**, copiando os arquivos para o outFile e outDir ou checando questões relacionadas a tipagem ou mesmo das bibliotecas que estão sendo utilizadas

A opção **allowJs** habilita a utilização da linguagem JavaScript em conjunto com TypeScript, incorporando ambas na pasta de destino

A opção **checkJs** realiza uma verificação em arquivos escritos na linguagem JavaScript em relação a tipos, parâmetros na invocação de funções e muito mais

A screenshot of a Mac OS X application window titled "Translator.js — typescript". The window contains three tabs: "main.ts", "Translator.js X", and "Repository.ts". The "Translator.js X" tab is currently active, showing the following TypeScript code:

```
1  export function translate (text) {
2      if (text === "Clean Code") return "Código Limpo";
3      if (text === "Refactoring") return "Refatoração";
4  }
5
```

main.ts — typescript

TS main.ts    JS Translator.js    TS Repository.ts

```
1 import Author, { Category } from "./entity/Author";
2 import Repository from "./infra/Repository";
3 import { translate } from "./infra/Translator";
4
5 const authors = new Repository<Author>();
6 authors.add(new Author("Robert Martin", 62, [
7   { title: "Clean Code", category: Category.PROGRAMMING }
8 ]));
9 authors.add(new Author("Martin Fowler", 58, [
10   { title: "Refactoring", category: Category.PROGRAMMING }
11 ]));
12
13 for (const author of authors) {
14   for (const book of author.books) {
15     console.log(translate(book.title));
16   }
17 }
18
```

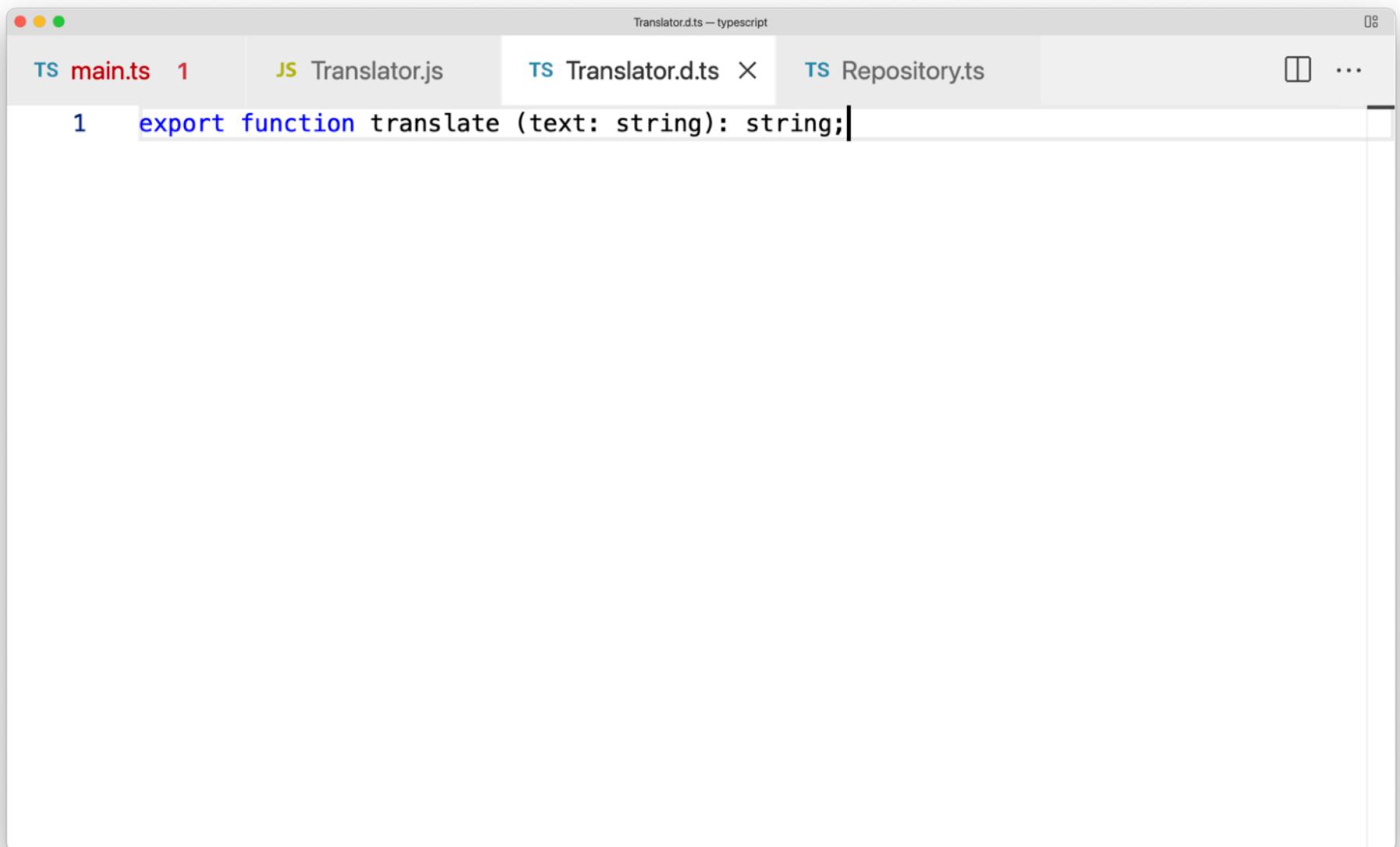
main.ts — typescript

TS main.ts    X JS Translator.js    TS Translator.d.ts    TS Repository.ts    ⬤ ...

```
6   authors.add(new Author("Robert Martin", 62, [
7     { title: "Clean Code", category: Category.PROGRAMMING }
8   )));
9   authors.add(new Author("Martin Fowler", 58, [
10    { title: "Refactoring", category: Category.PROGRAMMING }
11  )));
12
13  for (const author of authors) {
14    for (const book of author.books) {
15      console.log(translate(book.title));
16    }
17 }
```

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

rodrigobranas typescript \$ npx tsc  
rodrigobranas typescript \$ node dist/main  
Código Limpo  
Refatoração  
rodrigobranas typescript \$



Translator.d.ts — typescript

TS main.ts 1    JS Translator.js    TS Translator.d.ts ×    TS Repository.ts    ⬤ ...

```
1 export function translate (text: string): string;
```

main.ts — typescript

TS main.ts 1 × JS Translator.js TS Translator.d.ts TS Repository.ts ⋮

```
6 authors.add(new Author("Robert Martin", 62, [
7   { title: "Clean Code", category: Category.PROGRAMMING }
8 ]);
9 authors.add(new Author("Martin Fowler", 58, [
10   { title: "Refactoring", category: Category.PROGRAMMING }
11 ]);
12
13 for (const author of authors) {
14   for (const book of author.books) {
15     console.log(translate(author.age));
16   }
17 }
```

PROBLEMS 1 OUTPUT ... Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

TS main.ts src 1

✖ Argument of type 'number' is not assignable to parameter of type 'string'. ts(2345) [15, 25]

A opção **declaration** indica que o transpilador deve criar um arquivo .d.ts, que é o arquivo responsável pela definição dos tipos

# Type Checking

É possível configurar exatamente o comportamento das verificações refletindo as políticas estabelecidas no projeto, assim como seria feito com qualquer outro tipo de linter

A opção **strict** habilita diversas verificações de tipagem como:

`strictNullChecks`

`strictFunctionTypes`

`strictBindCallApply`

`strictPropertyInitialization`

`noImplicitAny`

`noImplicitThis`

`useUnknownInCatchVariables`

`alwaysStrict`

A screenshot of a TypeScript code editor interface. The main editor window shows a file named "main.ts" with the following code:

```
TS main.ts 1 ●  
1 if (true) {  
2     console.log("Yes");  
3 } else {  
4     console.log("No");  
5 }  
6
```

The line "4 console.log("No");" is highlighted with a yellow background, indicating it is unreachable code. The status bar at the bottom of the editor window displays "allowUnreachableCode: false".

Below the editor, the "PROBLEMS" tab is active, showing one error: "Unreachable code detected. ts(7027) [4, 2]". The "OUTPUT" and "..." tabs are also visible.

allowUnreachableCode: false

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface running on a Mac. The title bar indicates the file is named "main.ts — typescript". The main editor window displays the following TypeScript code:

```
TS main.ts 1 ●  
1 function init () {  
2   const port = 80;  
3 }  
4
```

The line "const port = 80;" is highlighted with a yellow background, and the word "port" is underlined with a wavy green line, indicating it is unused.

Below the editor is the VS Code status bar, which includes tabs for "PROBLEMS", "OUTPUT", and "...". The "PROBLEMS" tab is active, showing a count of 1 error. The error message in the Problems panel is:

Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

TS main.ts src 1  
💡 'port' is declared but its value is never read. ts(6133) [2, 8]

# noUnusedLocals: true

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The main editor window displays a file named `main.ts` with the following content:

```
1 const numbers = [1, 2, 3];
2
3 primary: for (const number of numbers) {
4     console.log(number);
5 }
6
```

A yellow lightbulb icon is positioned next to the word `primary`, indicating a potential issue. The code editor has a light gray background with syntax highlighting.

Below the editor is the VS Code status bar, which includes icons for file operations and a battery level indicator.

The bottom of the screen features the VS Code interface:

- PROBLEMS** (1): The Problems view is open, showing one error: `Unused label. ts(7028) [3, 1]`.
- OUTPUT**
- ...**
- Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)**: A search bar for filtering problems.
- X**: Close button for the Problems view.

allowUnusedLabels: false

The screenshot shows a Microsoft Visual Studio Code interface. The main editor window displays a file named `main.ts` with the following content:

```
1 function init (port) {  
2     console.log(port);  
3 }  
4  
5 init(80);  
6
```

The word `port` is underlined with a red squiggly line, indicating a TypeScript error. A yellow lightbulb icon is positioned next to the first occurrence of `port`. The status bar at the top right shows the file name `main.ts — typescript` and a battery icon.

Below the editor is the **PROBLEMS** view, which contains one error entry:

- TS main.ts src 1  
Parameter 'port' implicitly has an 'any' type. ts(7006) [1, 16]

The **OUTPUT** tab is also visible in the bottom navigation bar.

# noImplicitAny: true

A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows "Rectangle.ts — typescript". The main editor area displays the following TypeScript code:

```
1  export default class Rectangle {
2      width: number;
3      height: number;
4
5      constructor (width: number, height: number) {
6          this.width = width;
7          this.height = height;
8      }
9
10     getArea() {
11         return function () {
12             return this.width * this.height;
13         }
14     }
15 }
```

The code editor highlights the word "this" in blue. A yellow status bar at the bottom indicates a warning or error on line 12, column 11. The status bar also shows the file name "Rectangle.ts" and the line number "12".

The bottom left shows the "PROBLEMS" tab with a count of 2, and the "OUTPUT" tab. The "PROBLEMS" tab displays two errors:

- > ✖ 'this' implicitly has type 'any' because it does not have a type annotation. ts(2683) [12, 11]
- > ✖ 'this' implicitly has type 'any' because it does not have a type annotation. ts(2683) [12, 24]

The bottom right of the status bar includes a "Filter" input field, a magnifying glass icon, and icons for expanding/collapsing the problems list and closing the panel.

# noImplicitThis: true

shapes.ts — typescript

TS shapes.ts 1 ×

```
10
11     export class Rectangle {
12         constructor (readonly x: number, readonly y: number) {
13             }
14
15         getArea () {
16             if (this.x && this.y) {
17                 return this.x * this.y;
18             }
19         }
20     }
21 }
22 }
```

PROBLEMS 1 OUTPUT TERMINAL ... Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

⚠️ TS shapes.ts src 1

⚠️ Not all code paths return a value. ts(7030) [15, 3]

# noImplicitReturns: true

shapes.ts — typescript

TS shapes.ts 1 ×

```
1  export namespace Shapes {
2      export class Square {
3          constructor (readonly x: number) {
4          }
5
6          get area () {
7              return this.x ** 2;
8          }
9
10         static calculateArea (x: number, y: number) {
11             return x ** 2;
12         }
13     }
14 }
```

PROBLEMS 1 OUTPUT TERMINAL ... Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

TS shapes.ts src 1

💡 'y' is declared but its value is never read. ts(6133) [10, 36]

noUnusedParameters: true

Square.ts — typescript

```
TS Square.ts 1 ×
```

```
4     x: number;
5
6     constructor (x: number) {
7         super();
8         this.x = x;
9     }
10
11    calculateArea(): number {
12        return this.x ** 2;
13    }
14
15 }
```

PROBLEMS 1 OUTPUT TERMINAL ... Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

TS Square.ts src 1

This member must have an 'override' modifier because it overrides a member in the base class 'Shape'. ts(4114) [11, 2]

# noImplicitOverride: true

The screenshot shows a Microsoft Visual Studio Code interface. The main editor window displays a file named `main.ts` with the following content:

```
TS main.ts 1 ×
1 const cars = [
2   { brand: "Fiat", model: "Uno" },
3   { brand: "Volkswagen", model: "Gol" },
4   { brand: "Chevrolet", model: "Corsa" }
5 ];
6 const car = cars.find(car => car.brand === "Volkswagen");
7 console.log(car.model);
8
```

The word `car` in the `console.log` statement is underlined with a red squiggly line, indicating a TypeScript error. The status bar at the bottom of the editor shows the message `Object is possibly 'undefined'. ts(2532) [7, 13]`.

Below the editor, the VS Code interface includes a navigation bar with tabs for PROBLEMS (1), OUTPUT, TERMINAL, and ... . To the right of the navigation bar is a search bar labeled "Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)". Further to the right are icons for filtering, expanding/collapsing, and closing the panel.

# strictNullChecks: true

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The main editor window displays a file named `Board.ts` with the following content:

```
1  export default class Board {  
2      name: string;  
3  
4      constructor () {  
5      }  
6  }  
7
```

A yellow rectangular highlight covers the entire code block. A small yellow lightbulb icon is positioned next to the word `name` in the second line. The status bar at the top right indicates `Board.ts — typescript`.

Below the editor is a navigation bar with tabs: `PROBLEMS` (highlighted), `OUTPUT`, `TERMINAL`, and `...`. To the right of the tabs is a search bar labeled `Filter (e.g. text, **/*.ts, !**/node_modules/**)` with a magnifying glass icon.

The `PROBLEMS` tab shows one error: `Property 'name' has no initializer and is not definitely assigned in the constructor. ts(2564) [2, 2]`. This message is displayed in a blue bar. The file path `src` is also visible in the problems list.

# strictPropertyInitialization: true

The screenshot shows a Visual Studio Code interface with the following details:

- EXPLORER** sidebar:
  - TYPESCRIPT** folder:
    - dist**: contains `main.js`, `tsconfig.tsbuildinfo`, and `node_modules`.
    - src**: contains `main.ts`, `package-lock.json`, `package.json`, and `tsconfig.json`.
- main.js** editor tab: The file contains the following code:

```
1 "use strict";
2 console.log("Hello World");
3
```
- Status Bar**: Shows the text `> OUTLINE`.

# alwaysStrict: true

The screenshot shows a Microsoft Visual Studio Code interface. The main editor window displays a file named `main.ts` with the following content:

```
1 function init(option: string) {
2     switch (option) {
3         case "a": {
4             console.log("a");
5         }
6         case "b": {
7             console.log("b");
8         }
9         case "c": {
10            console.log("c");
11        }
12    }
13 }
14 init("a");
```

The code editor has syntax highlighting where `function`, `switch`, `case`, and `console.log` are colored blue, and strings like `"a"`, `"b"`, and `"c"` are highlighted in red. The line `case "a": {` is currently selected and highlighted with a yellow background.

Below the editor is a navigation bar with tabs: PROBLEMS (highlighted), OUTPUT, TERMINAL, and DEBUG CONSOLE. To the right of the tabs is a search bar labeled "Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)".

The PROBLEMS tab shows two errors:

- A warning icon followed by the message "Fallthrough case in switch. ts(7029) [3, 3]".
- A warning icon followed by the message "Fallthrough case in switch. ts(7029) [6, 3]".

# noFallthroughCasesInSwitch: true

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top bar indicates the file is named "main.ts — typescript". The code editor window displays the following TypeScript code:

```
1 type fn = (x: number | string) => number;
2
3 const y = function (x: number): number {
4     return x * 10;
5 }
6
7 const z: fn = y;
8
```

The line "const z: fn = y;" is highlighted with a yellow background, indicating it is the source of an error. In the bottom-left corner of the code editor, there is a red squiggle under the letter "z", which is also underlined in red, further marking the error.

Below the code editor is the VS Code status bar, which includes tabs for PROBLEMS, OUTPUT, TERMINAL, and others. The PROBLEMS tab is active, showing a count of 1 error. The error message is displayed in the PROBLEMS panel:

TS main.ts src 1

×

Type '(x: number) => number' is not assignable to type 'fn'. ts(2322) [7, 7]

# strictFunctionTypes: true

A screenshot of the Visual Studio Code interface. The top bar shows the file name "main.ts — typescript". The code editor window contains the following TypeScript code:

```
TS main.ts 1 ×
1  try {
2    throw new Error();
3 } catch(e) {
4   console.log(e.message);
5 }
6
```

The line "console.log(e.message);" is highlighted with a yellow background, indicating a problem. The status bar at the bottom shows "PROBLEMS 1", "OUTPUT", "TERMINAL", and "DEBUG CONSOLE". The "PROBLEMS" tab is selected, displaying one error: "Object is of type 'unknown'. ts(2571) [4, 14]".

useUnknownInCatchVariables: true

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top bar indicates the file is named "main.ts — typescript". The code editor displays the following TypeScript code:

```
TS main.ts 1 ×  
1 function fn(x: string) {  
2   return parseInt(x);  
3 }  
4  
5 const n1 = fn.call(undefined, "10");  
6  
7 const n2 = fn.call(undefined, false);  
8
```

A yellow highlight covers the line "const n2 = fn.call(undefined, false);". A small yellow lightbulb icon is positioned next to the word "false" on this line, indicating a potential issue.

The bottom navigation bar shows tabs for PROBLEMS (1), OUTPUT, TERMINAL, and ... . The PROBLEMS tab is active, displaying one error message:

Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

TS main.ts src 1

⊗ Argument of type 'boolean' is not assignable to parameter of type 'string'. ts(2345) [7, 31]

# strictBindCallApply: true

The screenshot shows a Microsoft Visual Studio Code interface. The main editor window displays a file named `main.ts` with the following content:

```
TS main.ts 1 ×
1  interface Car {
2    brand: string;
3    model: string;
4    category?: "compact" | "sedan" | "suv";
5  }
6
7  const car: Car = {
8    brand: "Volkswagen",
9    model: "Gol",
10   category: "compact"
11 };
12
13  car.category = "sedan";
14  car.category = undefined;
```

The code uses an optional property `category` of type `"compact" | "sedan" | "suv"`. In line 13, the value `"sedan"` is assigned to `car.category`, which is highlighted in blue. In line 14, the value `undefined` is assigned to `car.category`, which is underlined with a red squiggle, indicating a TypeScript error.

Below the editor, the **PROBLEMS** tab is selected, showing one error. The error message is: `Type 'undefined' is not assignable to type '"compact" | "sedan" | "suv"' with 'exactOptionalPropertyTypes: true'. Cons... ts(2412) [14, 1]`.

# exactOptionalPropertyTypes: true

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The main editor tab is titled "main.ts — typescript". The code in the editor is:

```
1  interface Env {
2      NAME: string;
3      OS: string;
4      [propName: string]: string;
5  }
6
7  declare const env: Env;
8
9  const sysName = env.NAME;
10 const os = env.OS;    string | undefined
11 const nodeEnv = env.NODE_ENV;
12
```

The word "NAME" is highlighted in red, indicating a potential spelling error or a reference to an undeclared variable. A tooltip or status bar above the cursor shows the type "string | undefined".

Below the editor, the "PROBLEMS" tab is selected, showing "No problems have been detected in the workspace." The "OUTPUT", "TERMINAL", and "..." tabs are also visible.

A large, semi-transparent gray overlay at the bottom contains the text "noUncheckedIndexedAccess: true" in white, bold, sans-serif font.

noUncheckedIndexedAccess: true

The screenshot shows a Microsoft Visual Studio Code interface. The main editor window is titled "main.ts — typescript" and contains the following TypeScript code:

```
TS main.ts 1 ×
1  interface Settings {
2    speed: "fast" | "slow";
3    quality: "high" | "low";
4    [key: string]: string;
5  }
6
7  const settings: Settings = {
8    speed: "fast",
9    quality: "high",
10   user: "rodrigo@branas.io"
11 }
12
13 console.log(settings.quality);
14 console.log(settings.user);
```

The code editor has syntax highlighting for TypeScript, with "user" highlighted in red as it is being accessed from an index signature. The status bar at the bottom indicates there is 1 error.

Below the editor is the "PROBLEMS" tab, which is active and shows 1 error. The error message is: "Property 'user' comes from an index signature, so it must be accessed with ['user']. ts(4111) [14, 22]".

noPropertyAccessFromIndexSignature: true