

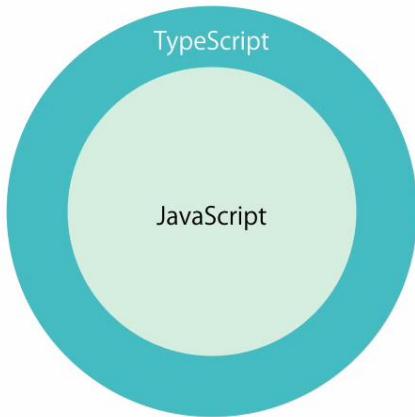
TypeScript Fundamentals

In this section...

- Type annotations
- Arrow functions
- Interfaces
- Classes
- Constructors
- Access modifiers
- Properties
- Modules

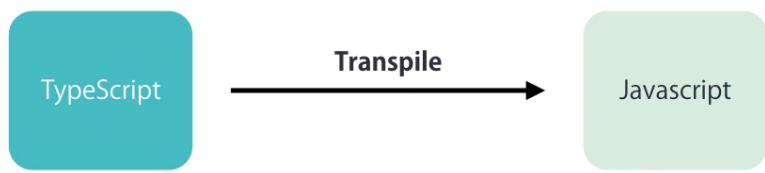
What is TypeScript?

|



TypeScript

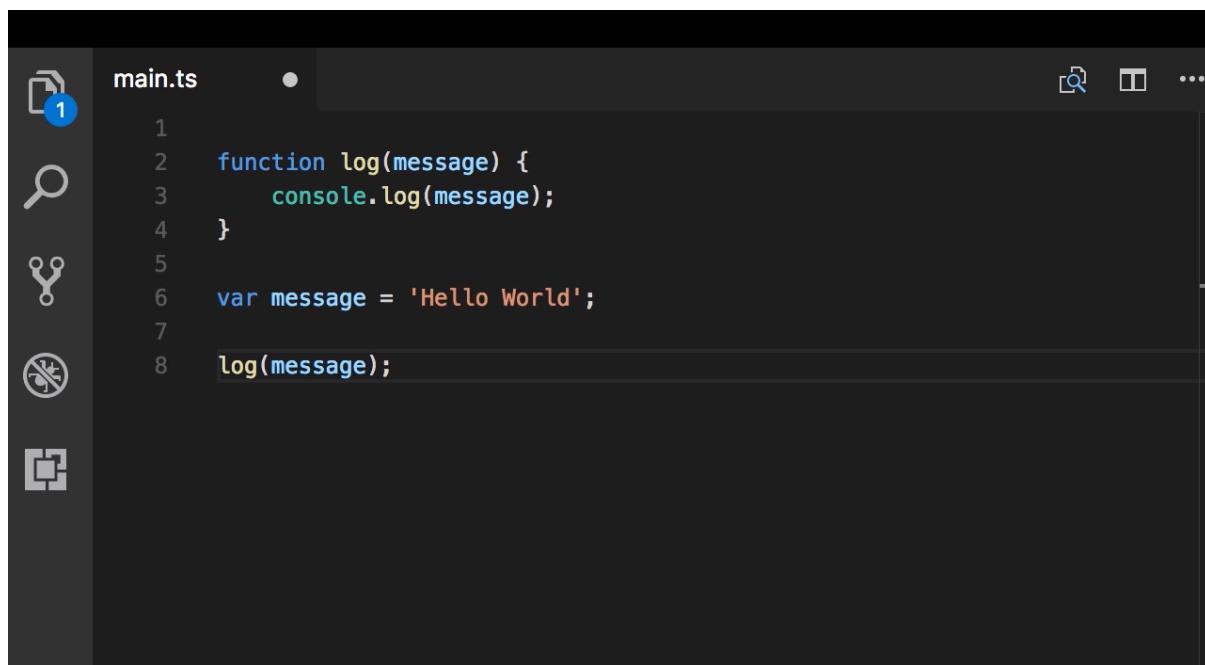
- Strong typing
- Object-oriented features
- Compile-time errors
- Great tooling



```
code $  
code $sudo npm install -g typescript  
Password:  
/usr/local/bin/tsserver -> /usr/local/lib/node_modules/typescript/bin/t  
sserver  
/usr/local/bin/tsc -> /usr/local/lib/node_modules/typescript/bin/tsc  
/usr/local/lib  
└─ typescript@2.3.4  
  
code $
```

```
code $  
code $tsc --version  
Version 2.3.4  
code $
```

```
code $  
code $mkdir ts-hello  
code $cd ts-hello/  
ts-hello $code main.ts
```



The screenshot shows a code editor interface with a dark theme. On the left is a vertical sidebar with icons for file operations (New, Open, Save, Find, Replace, Undo, Redo, and others). The main area displays a single file named "main.ts". The code content is as follows:

```
1  function log(message) {  
2      console.log(message);  
3  }  
4  
5  var message = 'Hello World';  
6  
7  log(message);
```

```
ts-hello $  
ts-hello $tsc main.ts  
ts-hello $ls  
main.js main.ts  
ts-hello $
```

```
ts-hello $  
ts-hello $ng serve
```

Transpiling is automatic in the above case.

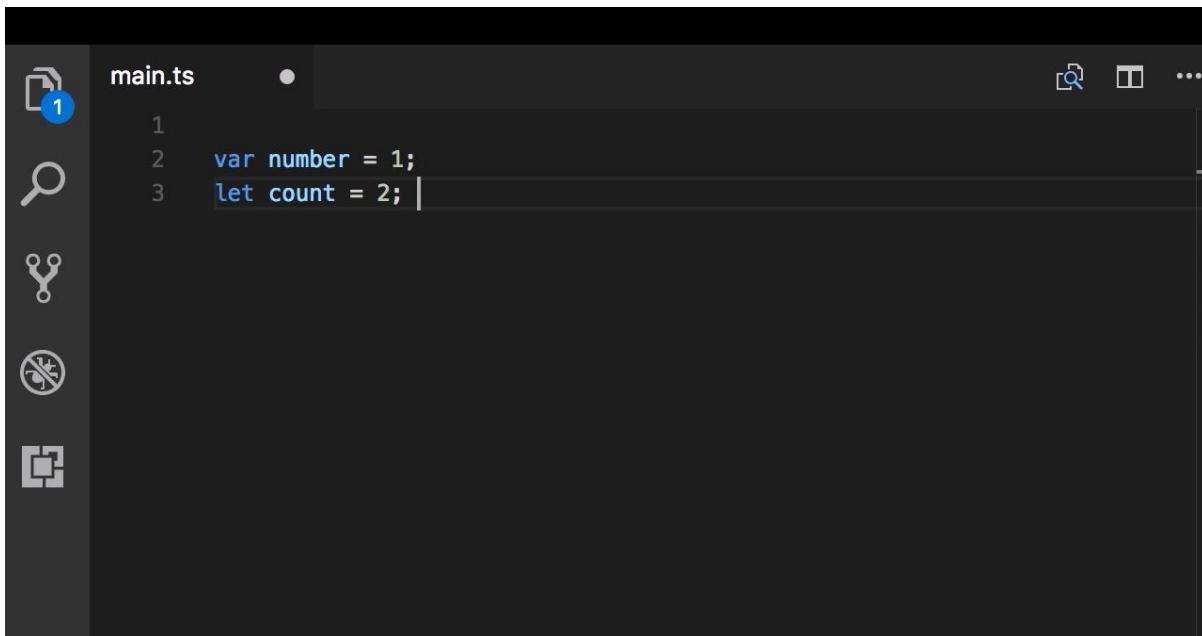
The screenshot shows a dark-themed code editor interface. On the left is a vertical sidebar with five icons: a file, a magnifying glass, a wrench, a circular arrow, and a refresh symbol. The main area has two tabs: 'main.ts' and 'main.js'. The 'main.js' tab is active, displaying the following code:

```
1 function log(message) {  
2     console.log(message);  
3 }  
4 var message = 'Hello World';  
5 log(message);  
6
```

The screenshot shows a terminal window with a dark background. The command 'ts-hello \$' is entered, followed by '\$node main.js', which is then executed. The output 'Hello World' is displayed, and the prompt '\$' appears again.

```
ts-hello $  
ts-hello $node main.js  
Hello World  
ts-hello $
```

Declaring Variables

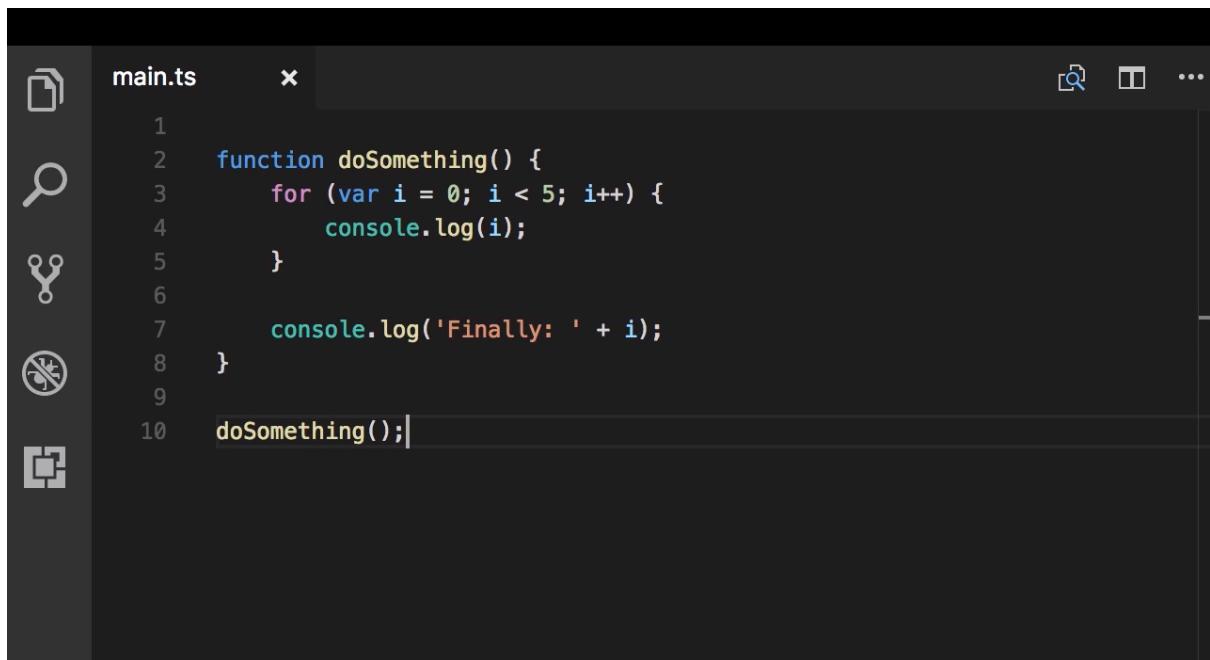


```
main.ts
1
2 var number = 1;
3 let count = 2; |
```

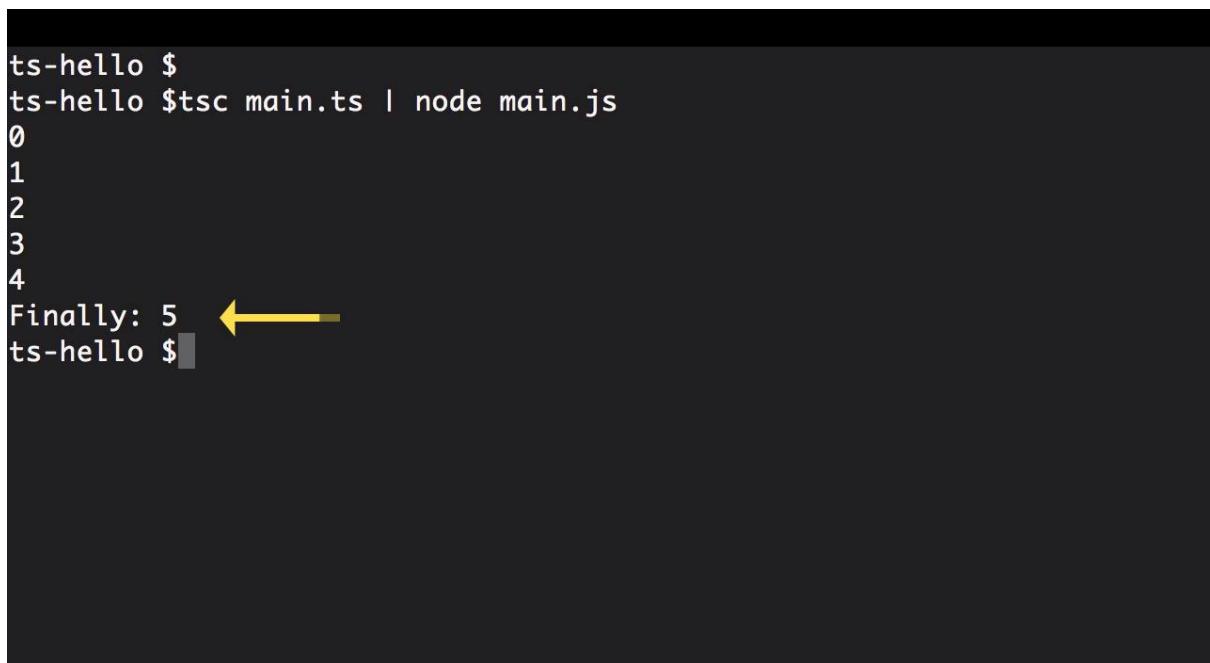
JavaScript Versions

- **ES5 (ECMAScript 5)**: supported by all browsers
- **ES6 (2015)**
- **ES2016**
- **ES2017**

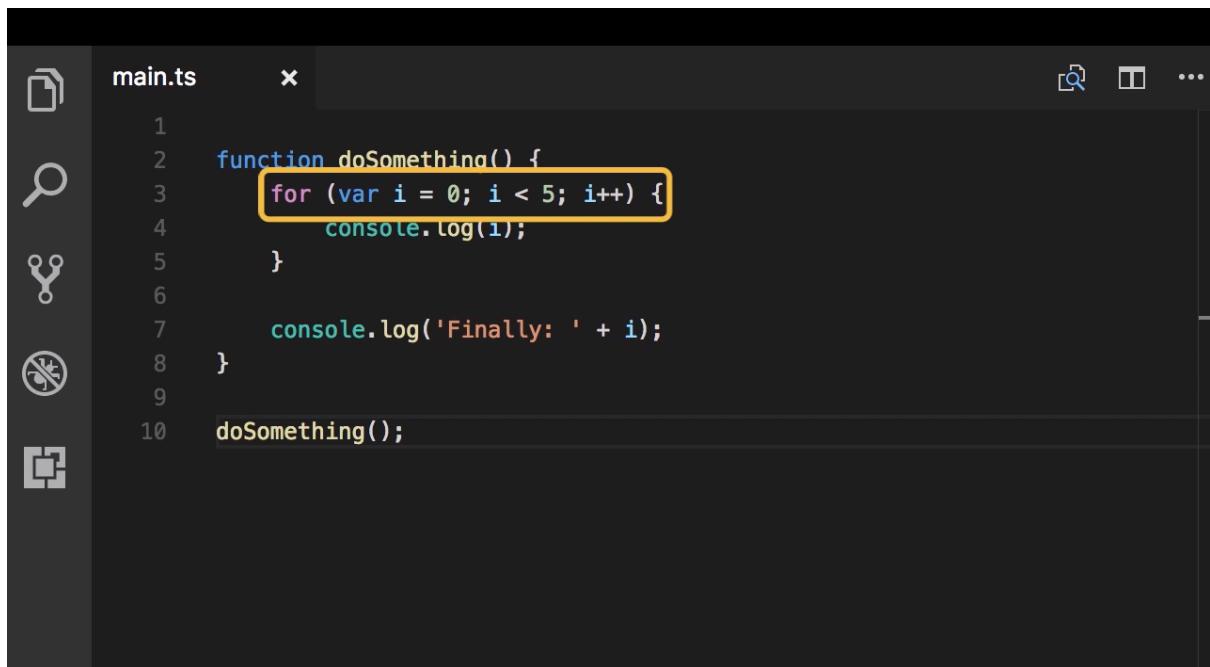
Let keyword



```
main.ts      x
1
2  function doSomething() {
3      for (var i = 0; i < 5; i++) {
4          console.log(i);
5      }
6
7      console.log('Finally: ' + i);
8  }
9
10 doSomething();
```

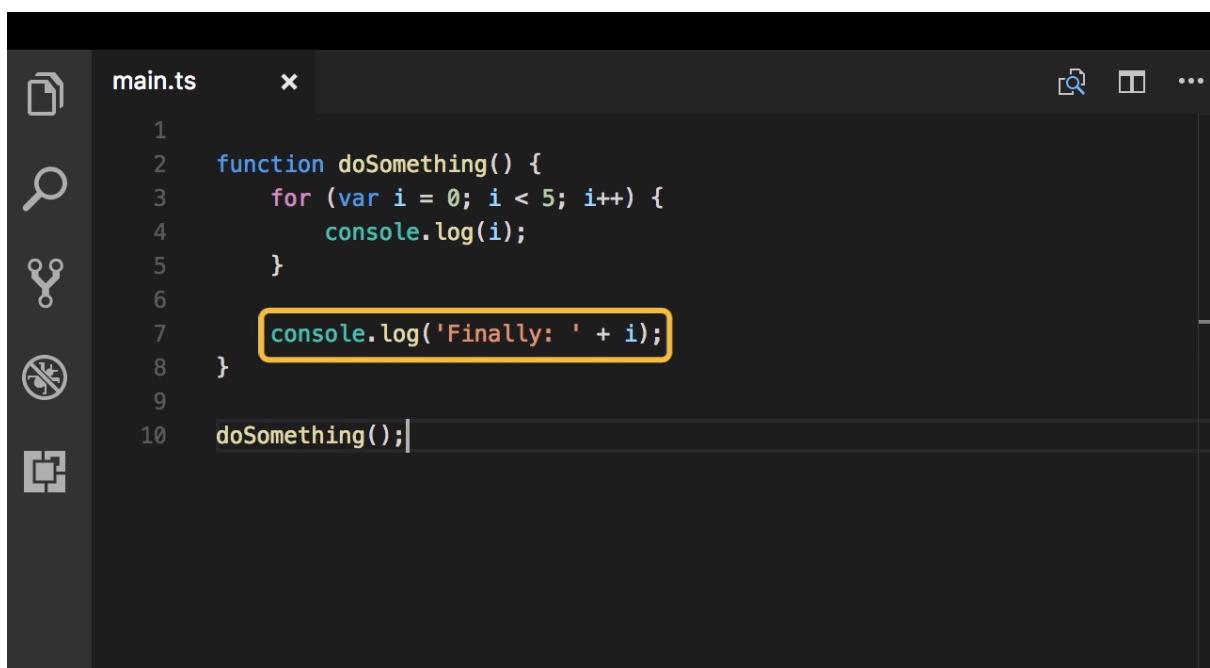


```
ts-hello $  
ts-hello $tsc main.ts | node main.js  
0  
1  
2  
3  
4  
Finally: 5 ←  
ts-hello $
```



A screenshot of a code editor interface showing a file named "main.ts". The code contains a function "doSomething" with a for loop. The for loop's body, which includes a console log statement, is highlighted with a yellow rectangular selection.

```
1
2  function doSomething() {
3      for (var i = 0; i < 5; i++) {
4          console.log(i);
5      }
6
7      console.log('Finally: ' + i);
8  }
9
10 doSomething();
```



A screenshot of a code editor interface showing a file named "main.ts". The code contains a function "doSomething" with a for loop and a final console log statement at the end. The final console log statement is highlighted with a yellow rectangular selection.

```
1
2  function doSomething() {
3      for (var i = 0; i < 5; i++) {
4          console.log(i);
5      }
6
7      console.log('Finally: ' + i);
8  }
9
10 doSomething();|
```

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

```
1
2  function doSomething() {
3      for (var i = 0; i < 5; i++) {
4          console.log(i);
5      }
6
7      console.log('Finally: ' + i);
8  }
9
10 doSomething();
```

The entire function body, from "function doSomething()" to the final closing brace, is highlighted with a yellow box.

A screenshot of a code editor window titled "main.ts". The code editor displays the same TypeScript code as the first screenshot:

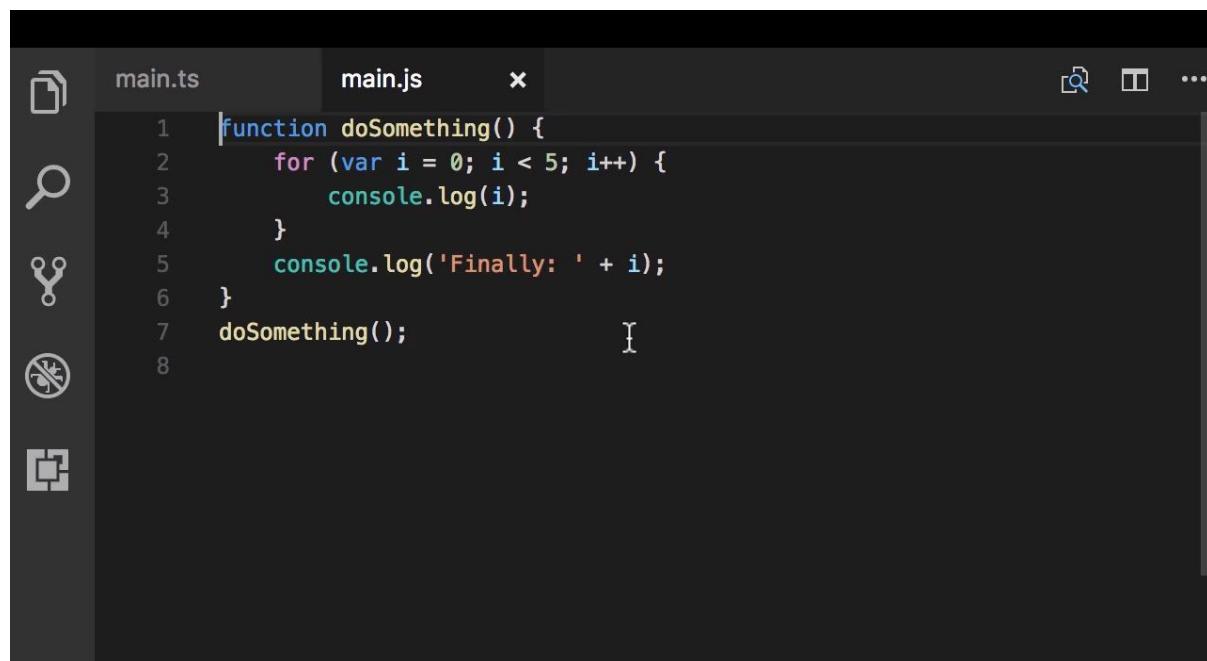
```
1
2  function doSomething() {
3      for (var i = 0; i < 5; i++) {
4          console.log(i);
5      }
6
7      console.log('Finally: ' + i);
8  }
9
10 doSomething();
```

The entire function body is highlighted with a yellow box. A cursor is visible at the end of the file, after the final closing brace of the function definition.

```
main.ts
1
2 function doSomething() {
3     for (let i = 0; i < 5; i++) {
4         console.log(i);
5     }
6
7     console.log('Finally: ' + i);
8 }
9
10 doSomething();
```

```
main.ts
1
2 function doSomething() {
3     for (let i = 0; i < 5; i++) {
4         console.log(i);
5     }
6
7     console.log('Finally: ' + i);
8 }
9
10 doSomething();
```

```
ts-hello $  
ts-hello $rm main.js  
ts-hello $tsc main.ts  
main.ts(7,31): error TS2304: Cannot find name 'i'.  
ts-hello $ls  
main.js main.ts  
ts-hello $
```



The screenshot shows a code editor interface with two tabs: 'main.ts' and 'main.js'. The 'main.ts' tab is active, displaying the following TypeScript code:

```
1 function doSomething() {  
2     for (var i = 0; i < 5; i++) {  
3         console.log(i);  
4     }  
5     console.log('Finally: ' + i);  
6 }  
7 doSomething();
```

The 'main.js' tab shows the generated JavaScript code:

```
function doSomething() {  
    for (var i = 0; i < 5; i++) {  
        console.log(i);  
    }  
    console.log('Finally: ' + i);  
}  
doSomething();
```

The code editor has a dark theme and includes standard file operations like 'New', 'Open', 'Save', 'Find', 'Replace', and 'Copy' in its top bar.

```
ts-hello $  
ts-hello $node main.js  
0  
1  
2  
3  
4  
Finally: 5  
ts-hello $
```

Types

A screenshot of a code editor interface. On the left is a dark sidebar with icons for file, search, cut/copy/paste, and refresh. The main area shows a file named "main.ts". The code contains three lines:

```
1 let count = 5;
2
3 count = 'a';
```

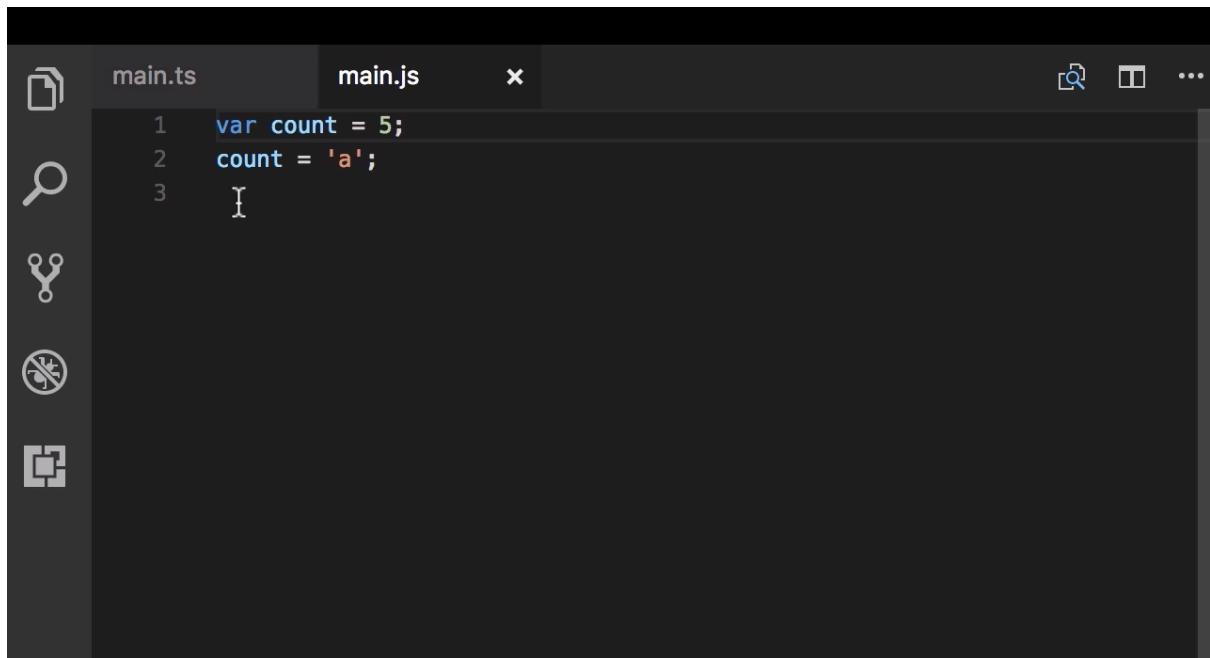
The third line, "count = 'a';", has a red squiggly underline under the variable "count". The status bar at the bottom of the editor window shows the text "VS Code" and "Version 1.60.2".

A screenshot of a code editor interface, similar to the one above. The sidebar icons are identical. The main area shows the same "main.ts" file with the following code:

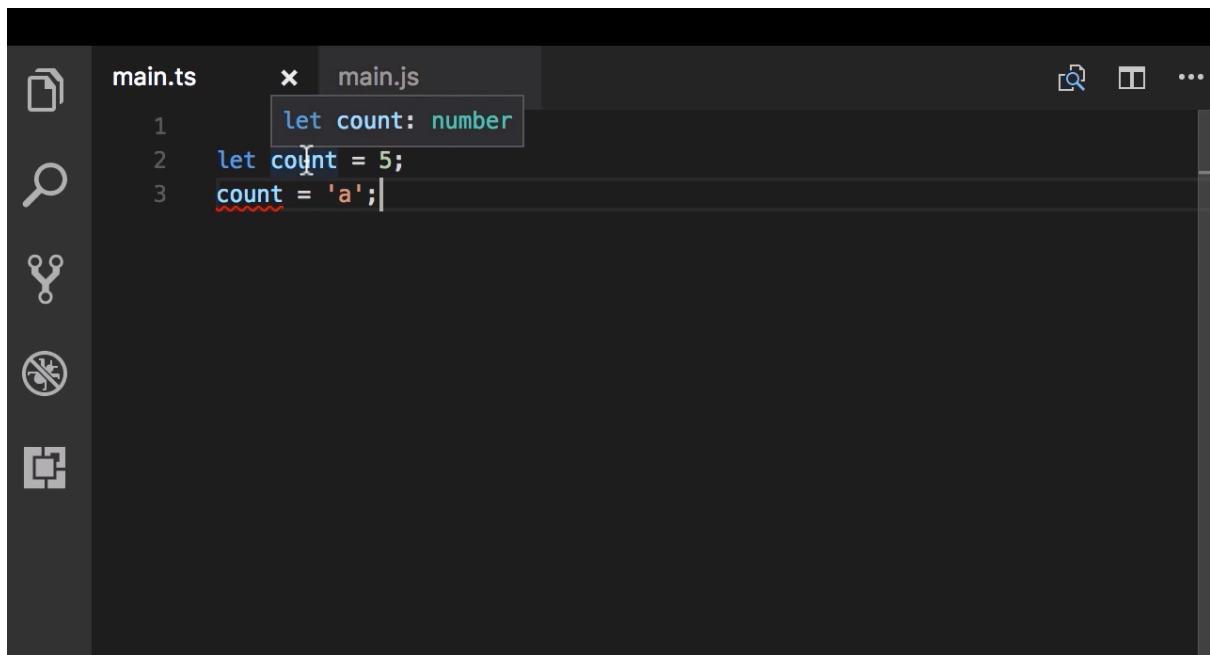
```
1
2 let count: number
3 count = 'a';
```

A tooltip message "[ts] Type '"a'" is not assignable to type 'number'." is displayed above the third line. The status bar at the bottom shows "VS Code" and "Version 1.60.2".

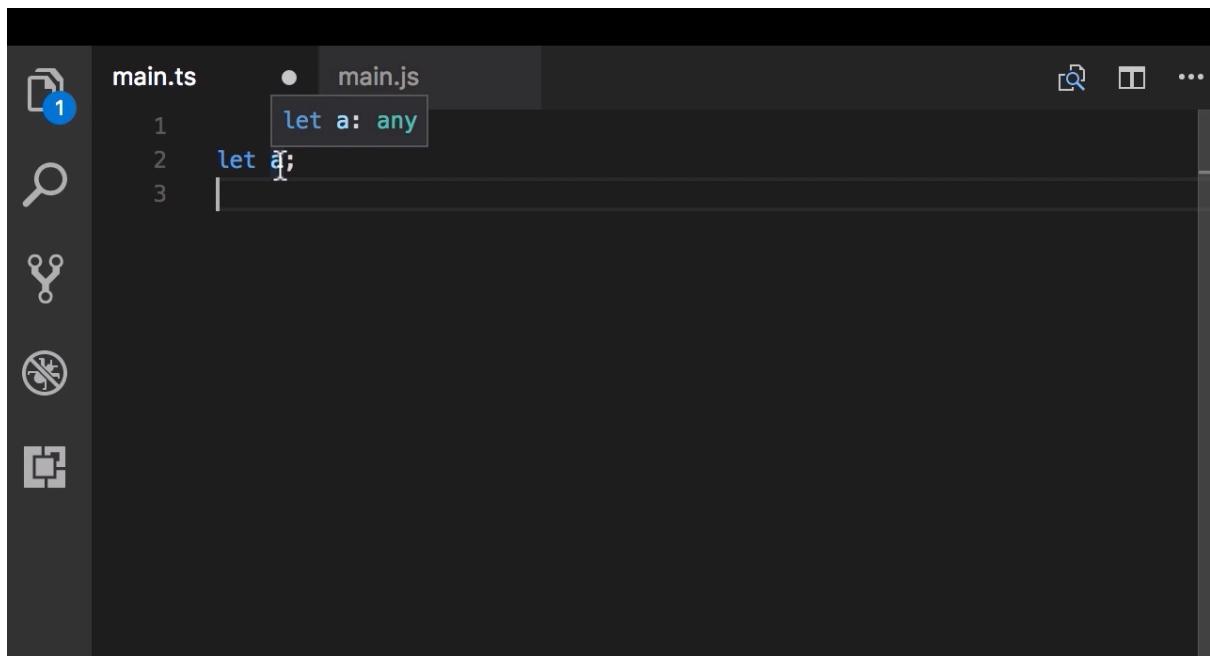
Converts to



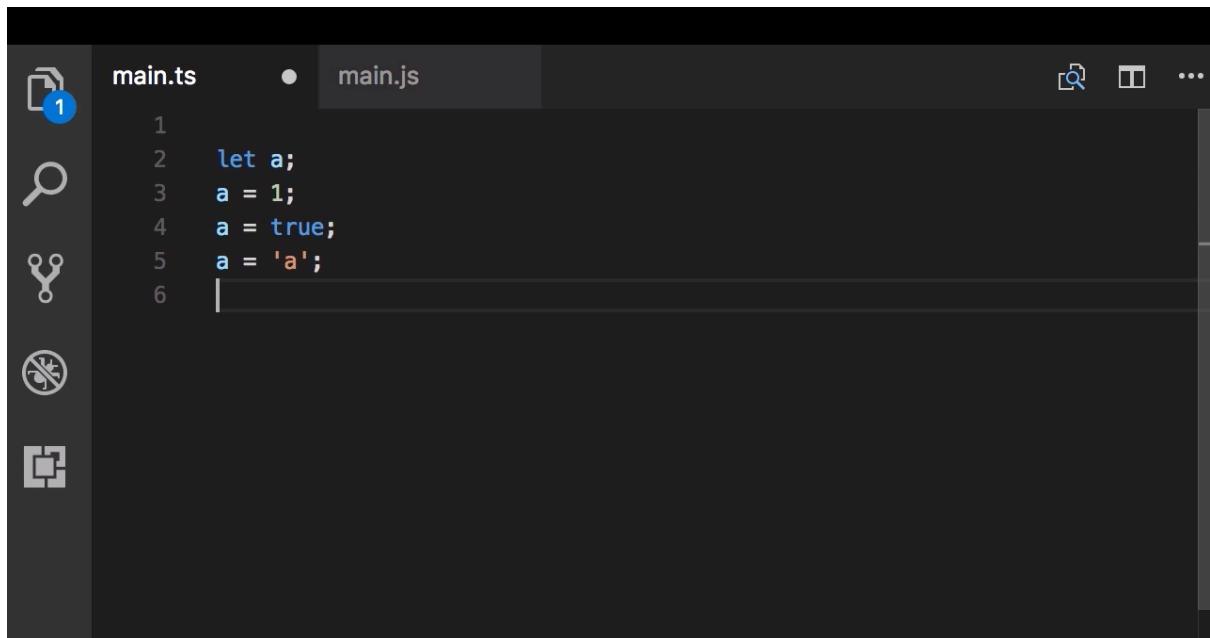
```
main.ts      main.js      x
1  var count = 5;
2  count = 'a';
3  
```



```
main.ts      x  main.js
1  let count: number
2  let count = 5;
3  count = 'a';|
```



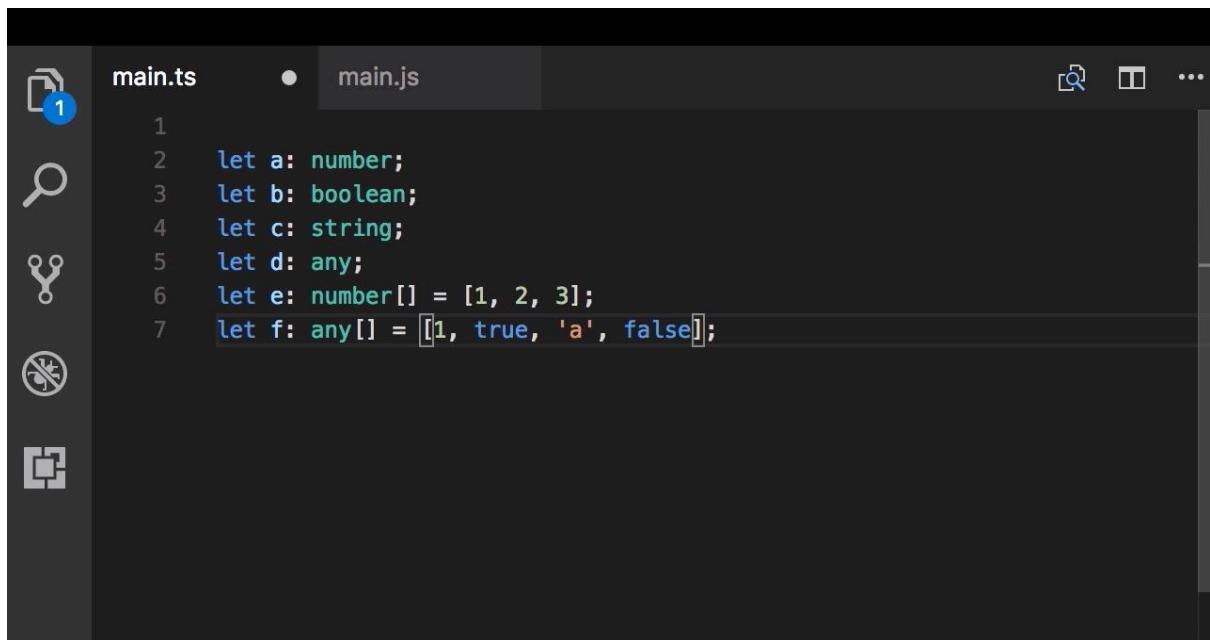
```
main.ts      ● main.js
1  let a: any
2  let a;
3
```



```
main.ts      ● main.js
1
2  let a;
3  a = 1;
4  a = true;
5  a = 'a';
6
```

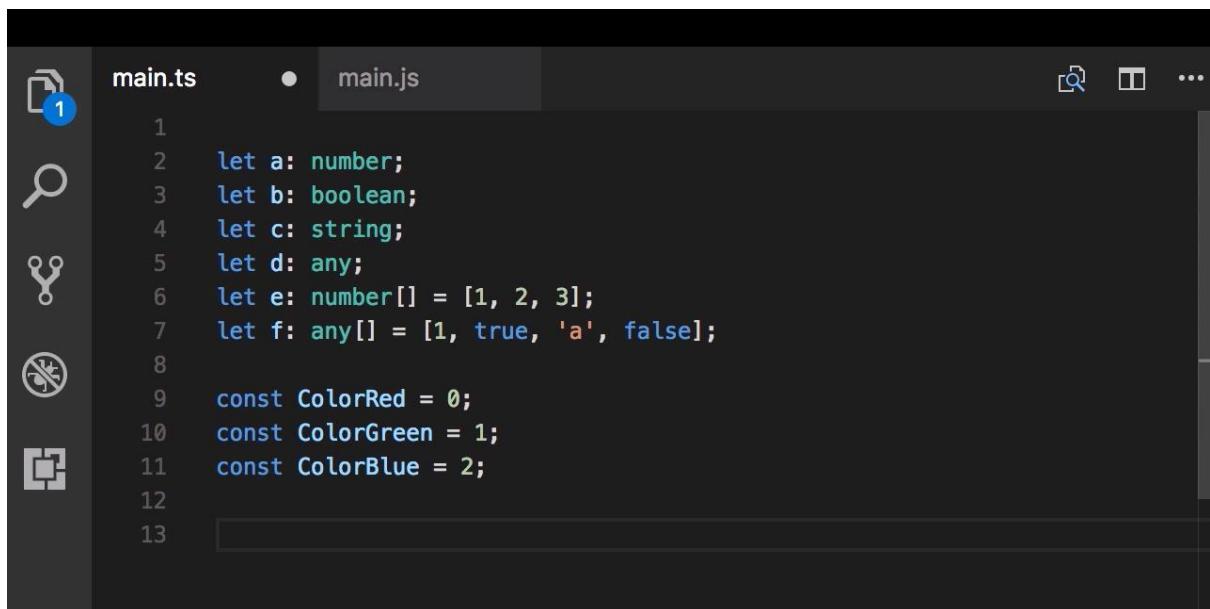
The screenshot shows a code editor interface with a dark theme. On the left is a vertical sidebar with icons: a document with a blue circle containing the number '1', a magnifying glass, a wrench, and a refresh symbol. The main area has two tabs at the top: 'main.ts' (active) and 'main.js'. The 'main.ts' tab contains the following TypeScript code:

```
1 let a: number;
2 a = 1;
3 a = true;
4 a = 'a';
5
```



```
main.ts ● main.js  
1  
2 let a: number;  
3 let b: boolean;  
4 let c: string;  
5 let d: any;  
6 let e: number[] = [1, 2, 3];  
7 let f: any[] = [1, true, 'a', false];
```

enums



```
main.ts ● main.js  
1  
2 let a: number;  
3 let b: boolean;  
4 let c: string;  
5 let d: any;  
6 let e: number[] = [1, 2, 3];  
7 let f: any[] = [1, true, 'a', false];  
8  
9 const ColorRed = 0;  
10 const ColorGreen = 1;  
11 const ColorBlue = 2;
```

A screenshot of the Visual Studio Code interface. The left sidebar shows icons for file, search, and other tools. The main editor area has two tabs: "main.ts" and "main.js". The "main.ts" tab is active, displaying the following TypeScript code:

```
1 let a: number;
2 let b: boolean;
3 let c: string;
4 let d: any;
5 let e: number[] = [1, 2, 3];
6 let f: any[] = [1, true, 'a', false];
7
8 const ColorRed = 0;
9 const ColorGreen = 1;
10 const ColorBlue = 2;
11
12 enum Color { Red, Green, Blue };
13 let backgroundColor = Color.
```

The cursor is at the end of the line "let backgroundColor = Color.". A dropdown menu is open, showing three options: "Blue", "Green", and "Red". The "Blue" option is highlighted. The status bar at the bottom right indicates "(enum member) Color.Blue = 2".

A screenshot of the Visual Studio Code interface, identical to the one above but with the assignment completed. The code now looks like this:

```
1 let a: number;
2 let b: boolean;
3 let c: string;
4 let d: any;
5 let e: number[] = [1, 2, 3];
6 let f: any[] = [1, true, 'a', false];
7
8 const ColorRed = 0;
9 const ColorGreen = 1;
10 const ColorBlue = 2;
11
12 enum Color { Red, Green, Blue };
13 let backgroundColor = Color.Red;
```

```
1  let a: number;
2  let b: boolean;
3  let c: string;
4  let d: any;
5  let e: number[] = [1, 2, 3];
6  let f: any[] = [1, true, 'a', false];
7
8  const ColorRed = 0;
9  const ColorGreen = 1;
10 const ColorBlue = 2;
11
12 enum Color { Red = 0, Green = 1, Blue = 2 };
13 let backgroundColor = Color.Red;
```

```
ts-hello $  
ts-hello $tsc main.ts
```

main.ts

```
8  var ColorGreen = 1;
9  var ColorBlue = 2;
10 var Color;
11 (function (Color) {
12     Color[Color["Red"] = 0] = "Red";
13     Color[Color["Green"] = 1] = "Green";
14     Color[Color["Blue"] = 2] = "Blue";
15     Color[Color["Purple"] = 3] = "Purple";
16 })(Color || (Color = {}));
17 ;
18 var backgroundColor = Color.Red;
19
```

main.ts

```
1
2  let a: number;
3  let b: boolean;
4  let c: string;
5  let d: any;
6  let e: number[] = [1, 2, 3];
7  let f: any[] = [1, true, 'a', false];
8
9  const ColorRed = 0;
10 const ColorGreen = 1;
11 const ColorBlue = 2;
12
13 enum Color { Red = 0, Green = 1, Blue = 2, Purple = 3 };
14 let backgroundColor = Color.Red;
```

Type Assertions

A screenshot of a code editor interface. On the left is a dark sidebar with icons for file, search, cut/paste, and refresh. The main area shows a file named "main.ts". The code contains three lines:

```
1 let message = 'abc';
2
3 message.
```

The cursor is at the end of the third line, and a tooltip is displayed, listing several methods available on the string variable:

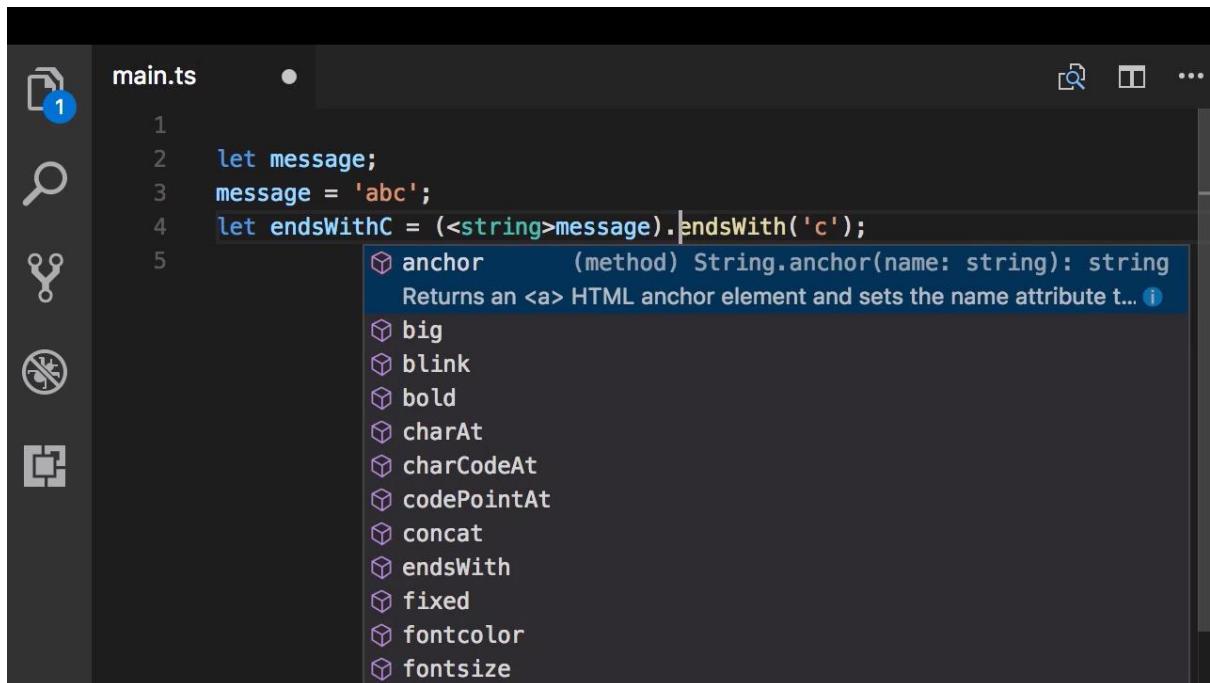
- anchor (method) String.anchor(name: string): string
Returns an <a> HTML anchor element and sets the name attribute to ... ⓘ
- big
- blink
- bold
- charAt
- charCodeAt
- codePointAt
- concat
- endsWith
- fixed
- fontcolor
- fontsize

A screenshot of a code editor interface. On the left is a dark sidebar with icons for file, search, cut/paste, and refresh. The main area shows a file named "main.ts". The code contains four lines:

```
1
2 let message = 'abc';
3 let endsWithC = message.endsWith('c');
4
```

The cursor is at the end of the fourth line. A small blue checkmark icon is visible in the status bar at the bottom of the editor window.

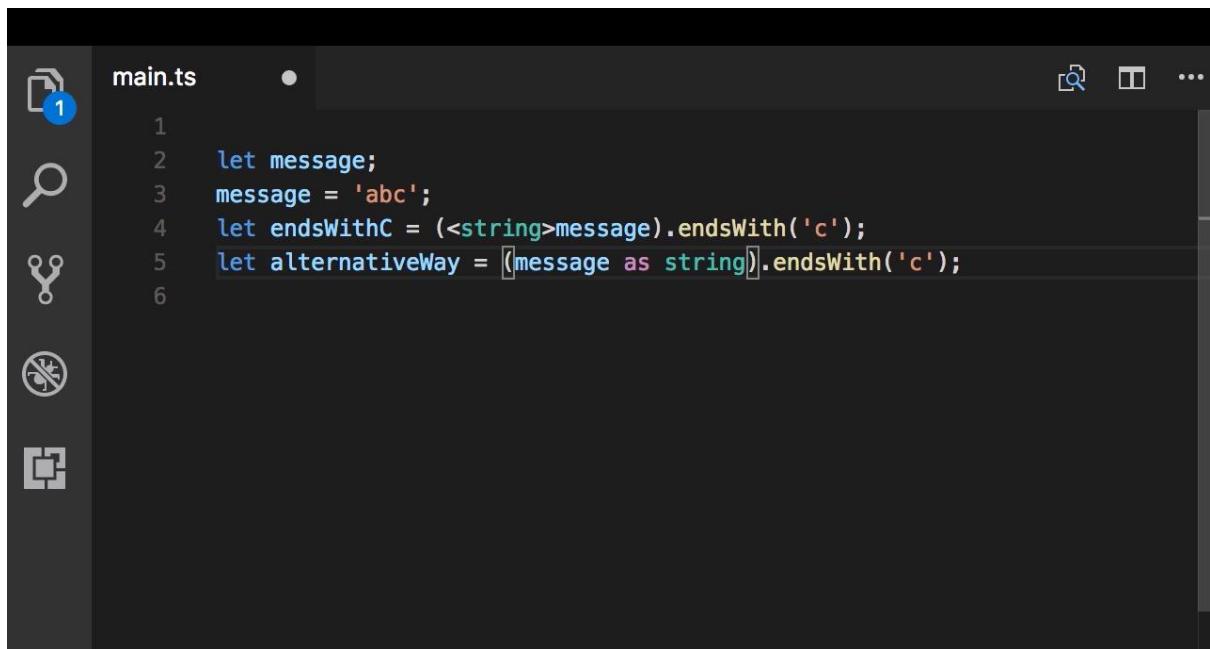
Type assertion in action:



The screenshot shows a tooltip for the `endsWith` method on a string variable. The code in the editor is:

```
1 let message;
2 message = 'abc';
3 let endsWithC = (<string>message).endsWith('c');
```

The tooltip for `endsWith` is displayed, showing its signature: `anchor (method) String.anchor(name: string): string`. It also includes a description: "Returns an <a> HTML anchor element and sets the name attribute t...". Other methods listed in the tooltip include `big`, `blink`, `bold`, `charAt`, `charCodeAt`, `concat`, `codePointAt`, `fixed`, `fontcolor`, and `fontsize`.

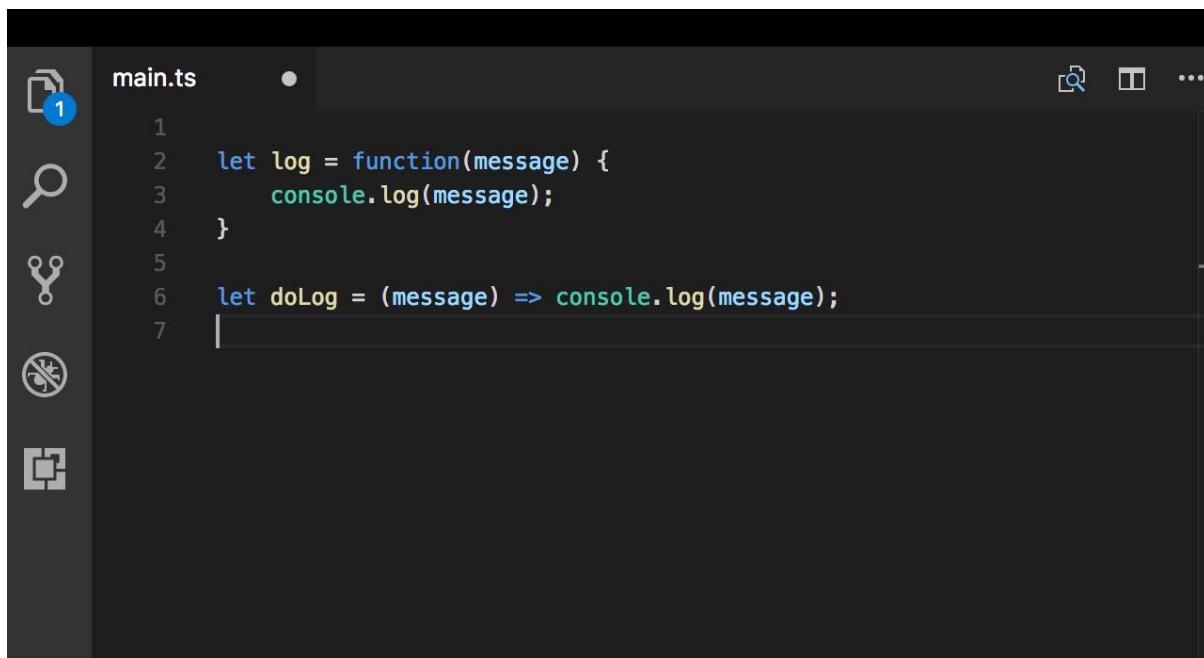


The screenshot shows two ways to perform type assertion on a string variable. The code in the editor is:

```
1
2 let message;
3 message = 'abc';
4 let endsWithC = (<string>message).endsWith('c');
5 let alternativeWay = [message as string].endsWith('c');
```

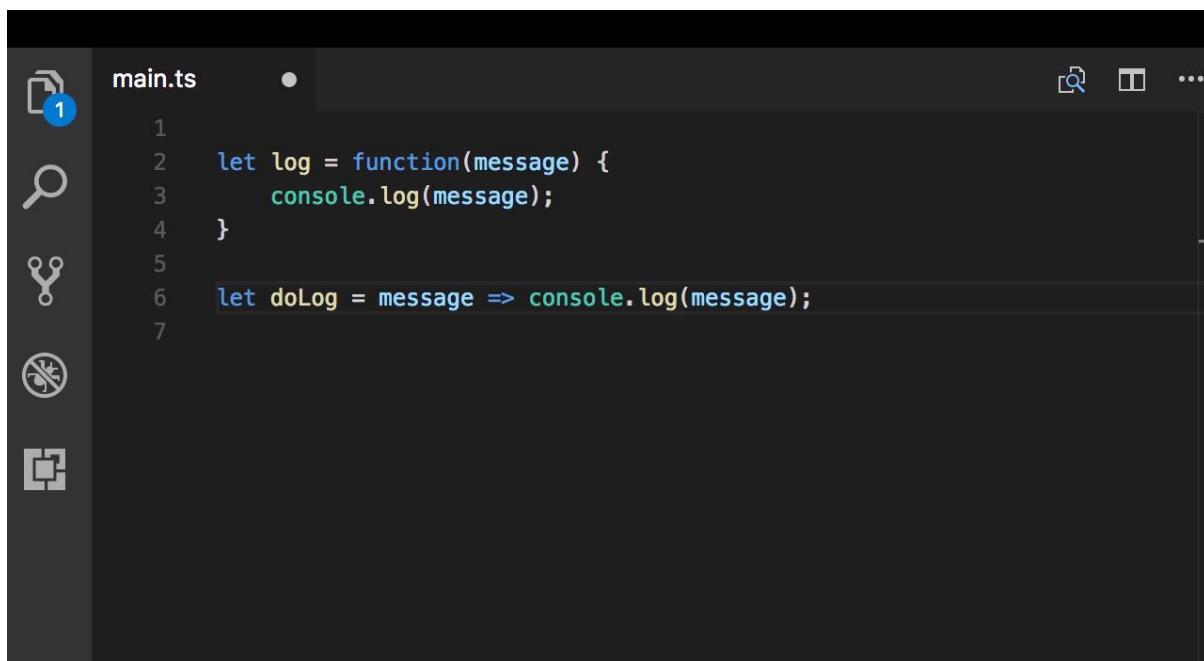
This demonstrates that both the standard type assertion syntax (`<string>message`) and the newer optional chaining syntax (`[message as string]`) can be used to assert the type of a variable.

Arrow Functions

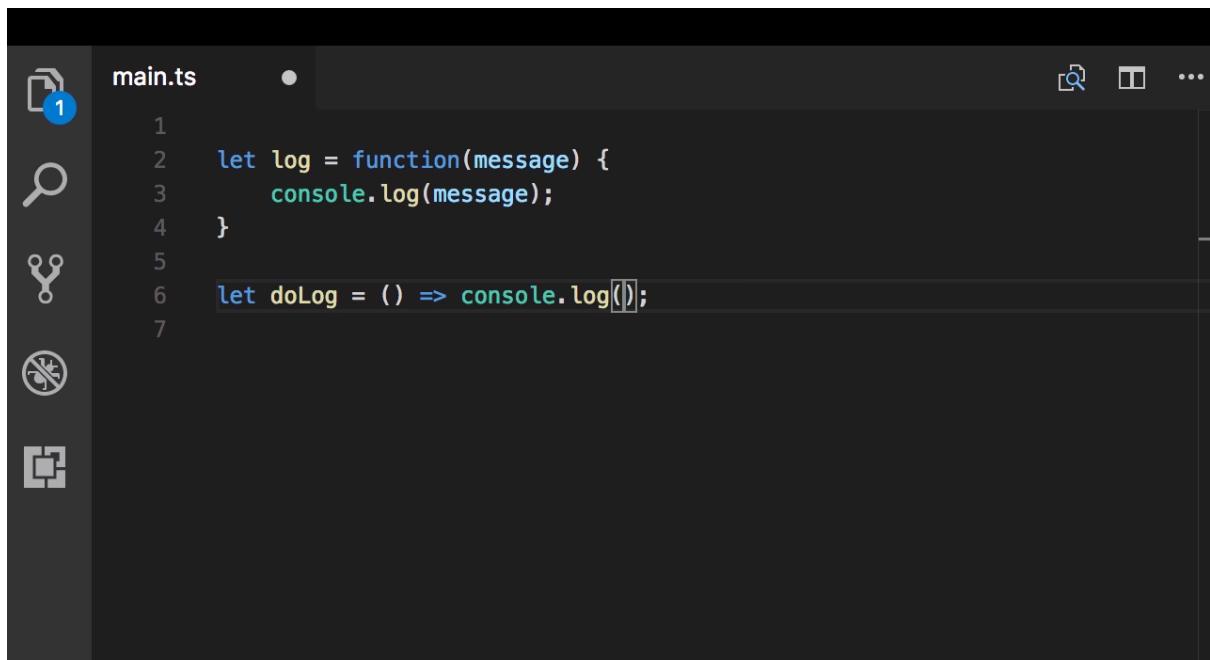


The screenshot shows a code editor window with a dark theme. On the left is a vertical toolbar with icons for file operations, search, and other tools. The main area displays a file named "main.ts". The code contains two examples of arrow functions:

```
1 let log = function(message) {  
2     console.log(message);  
3 }  
4  
5 let doLog = (message) => console.log(message);  
6  
7
```



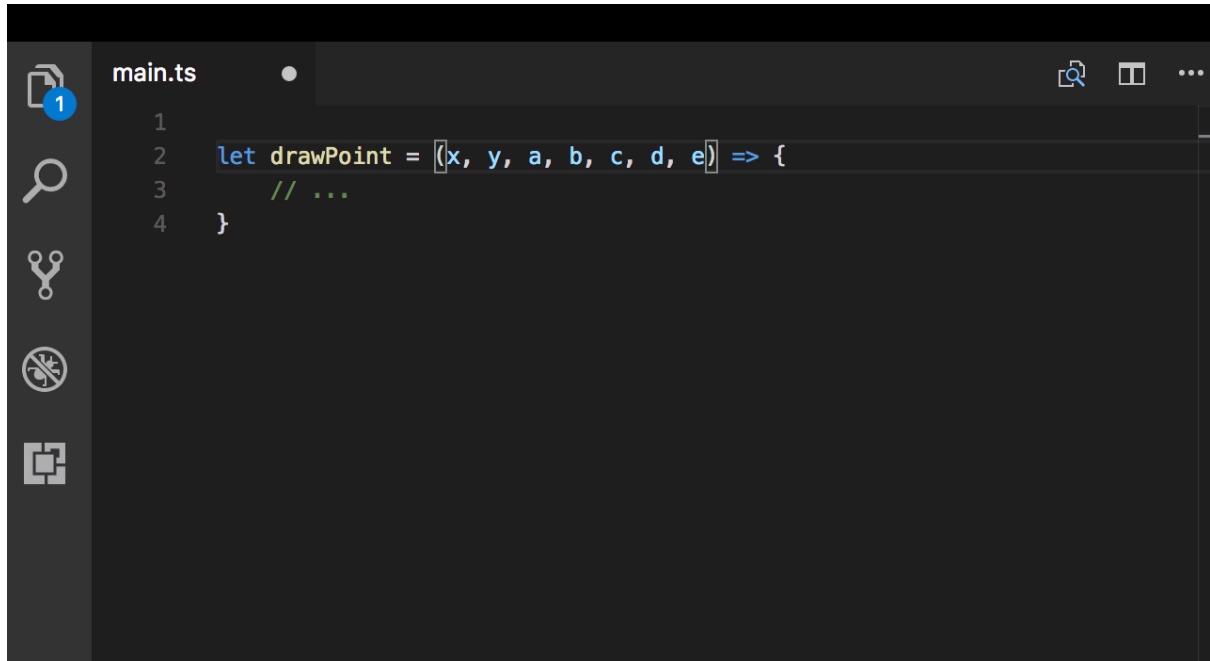
This screenshot is identical to the one above, but the last line of code ("let doLog = message => console.log(message);") is highlighted with a light gray background.



A screenshot of a code editor interface, likely Visual Studio Code, showing a single file named `main.ts`. The file contains the following TypeScript code:

```
1 let log = function(message) {
2     console.log(message);
3 }
4
5 let doLog = () => console.log();
```

Interfaces

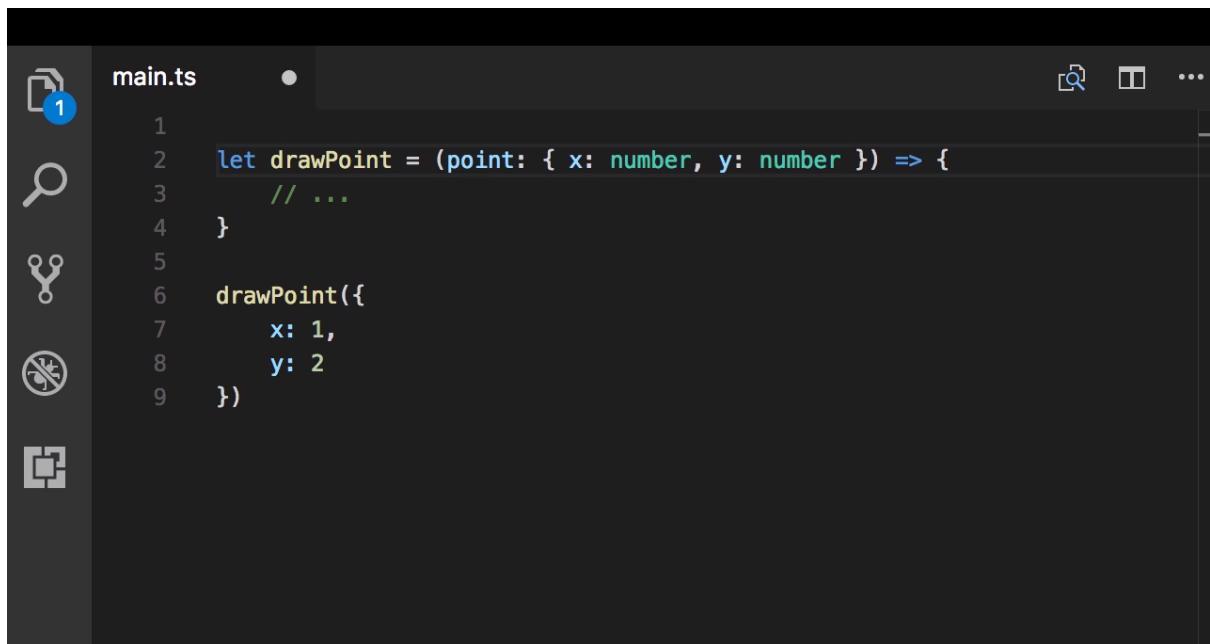


```
main.ts
1
2 let drawPoint = (x, y, a, b, c, d, e) => {
3     // ...
4 }
```



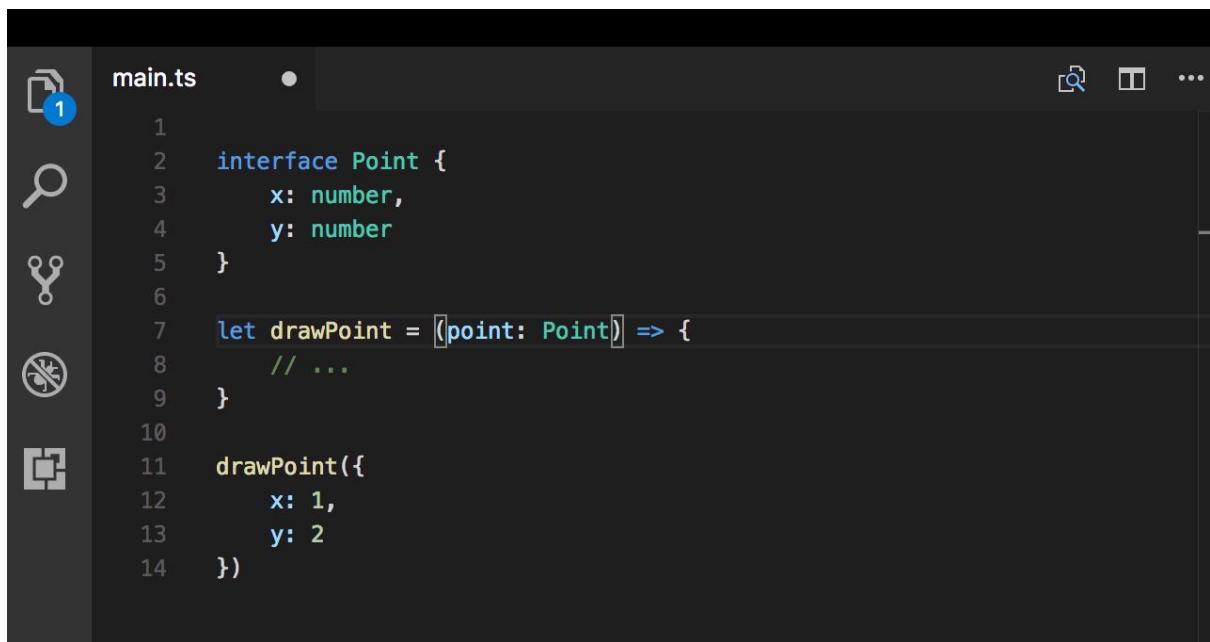
```
main.ts
1
2 let drawPoint = (point) => {
3     // ...
4 }
5
6 drawPoint({
7     x: 1,
8     y: 2
9 })
```

Inline annotation



main.ts

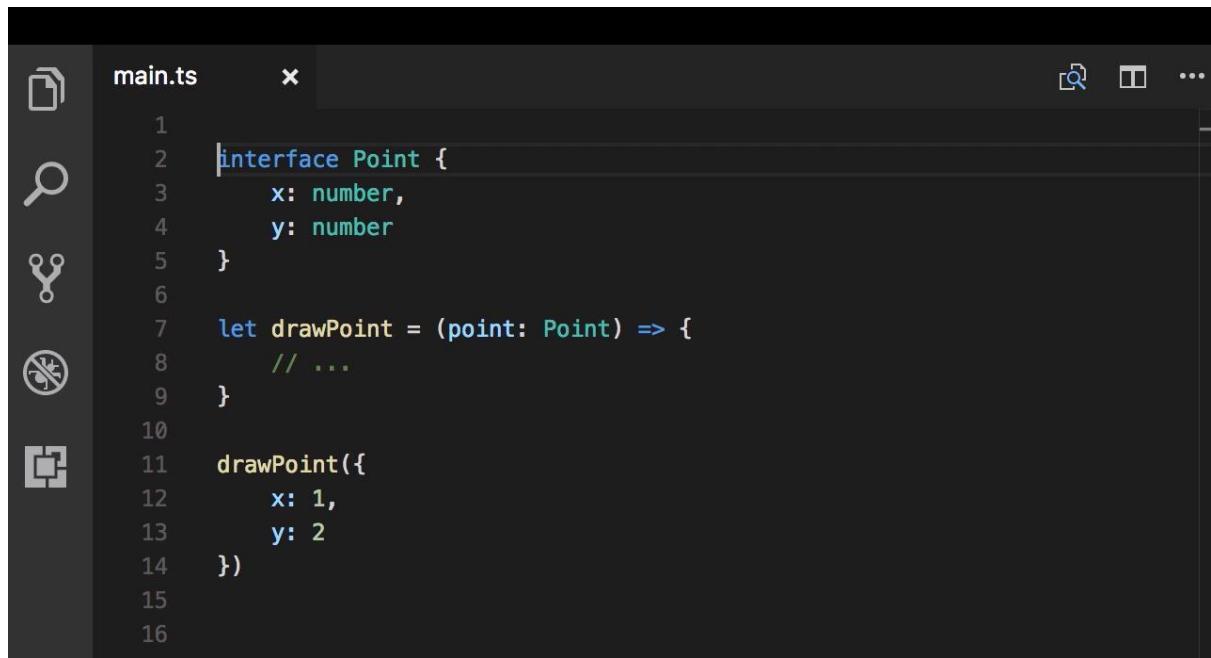
```
1
2 let drawPoint = (point: { x: number, y: number }) => {
3     // ...
4 }
5
6 drawPoint({
7     x: 1,
8     y: 2
9 })
```



main.ts

```
1
2 interface Point {
3     x: number,
4     y: number
5 }
6
7 let drawPoint = [point: Point] => {
8     // ...
9 }
10
11 drawPoint({
12     x: 1,
13     y: 2
14 })
```

Classes



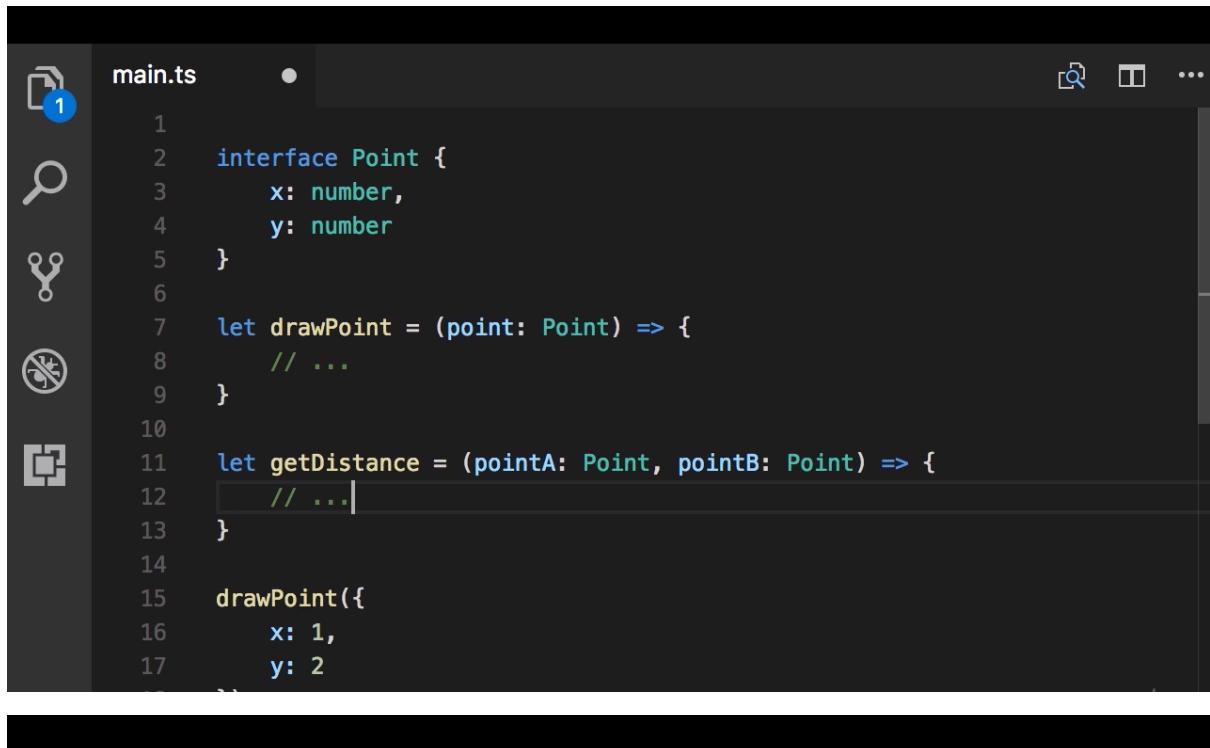
```
main.ts      x
1
2  interface Point {
3      x: number,
4      y: number
5  }
6
7  let drawPoint = (point: Point) => {
8      // ...
9  }
10
11 drawPoint({
12     x: 1,
13     y: 2
14 })
15
16
```

Cohesion



All the related things should be at one place. Data and Functions

This code violates the principle of cohesion.



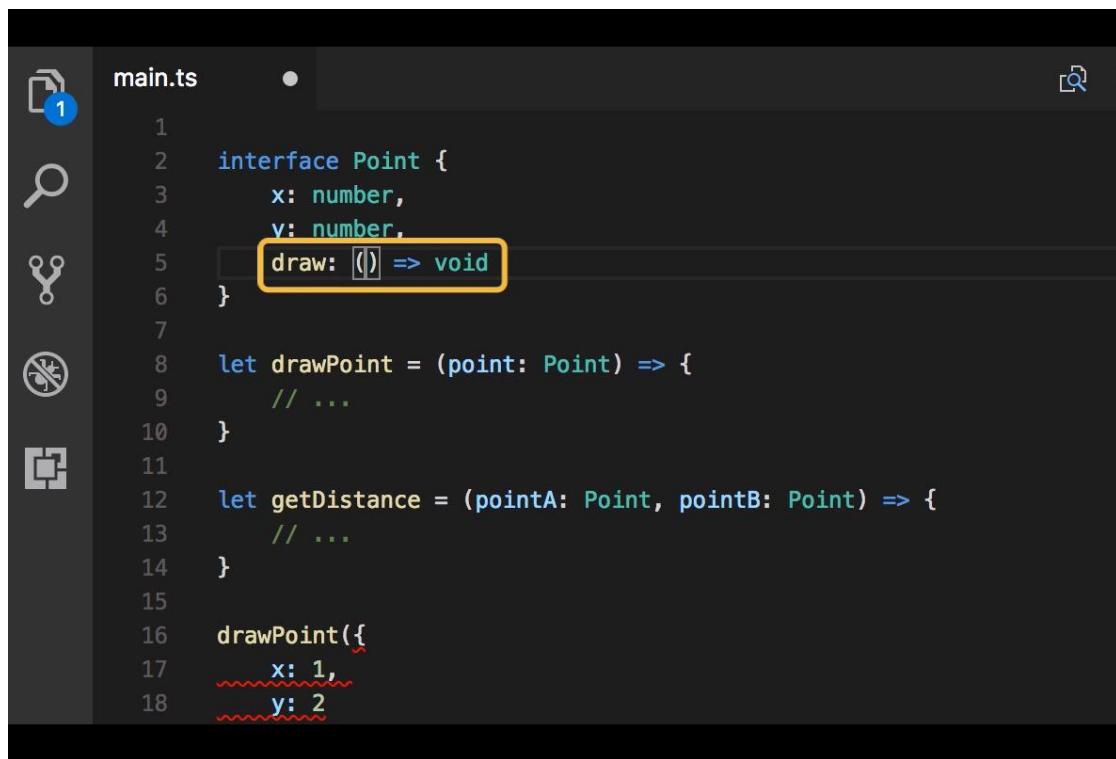
The screenshot shows a code editor window with a dark theme. On the left is a vertical toolbar with icons for file operations, search, and other tools. The main area displays the following TypeScript code:

```
main.ts
1
2  interface Point {
3      x: number,
4      y: number
5  }
6
7  let drawPoint = (point: Point) => {
8      // ...
9  }
10
11 let getDistance = (pointA: Point, pointB: Point) => {
12     // ...
13 }
14
15 drawPoint({
16     x: 1,
17     y: 2
18 })
```

Class

Groups variables (properties) and functions (methods) that are highly related.

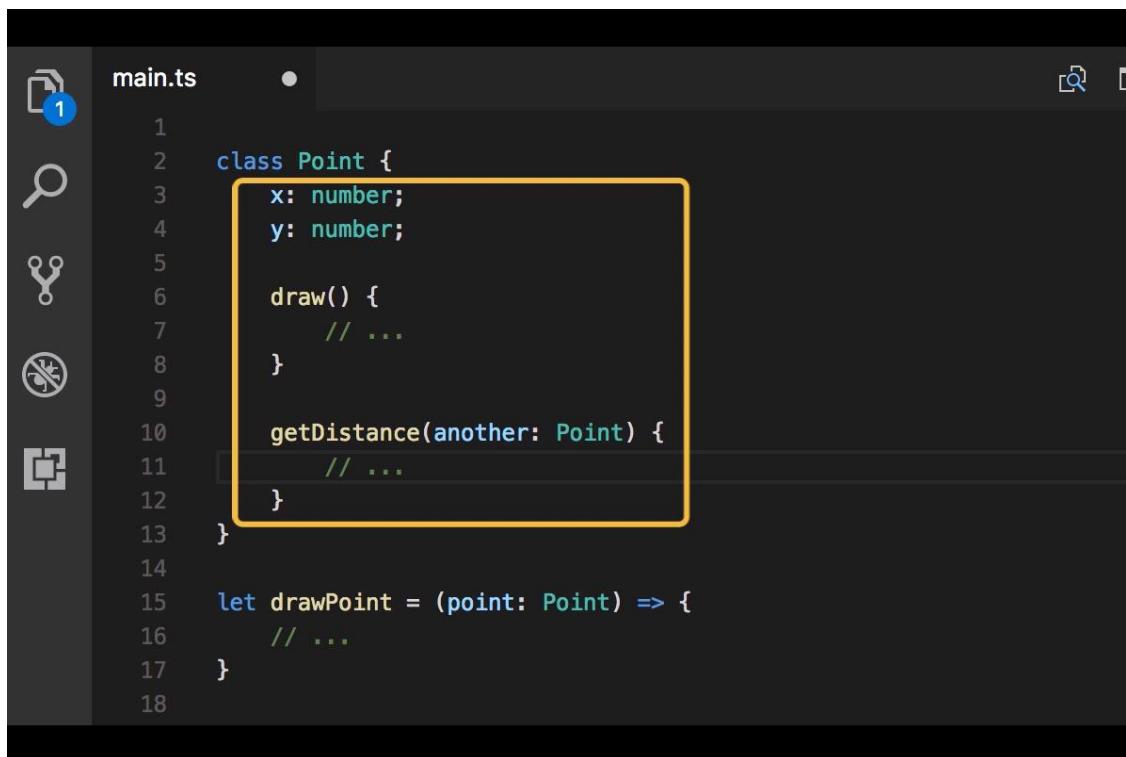
Interfaces have only declarations and not implementations



The screenshot shows a code editor window with a dark theme. On the left is a vertical toolbar with icons for file operations, search, and other tools. The main area displays a file named 'main.ts'.

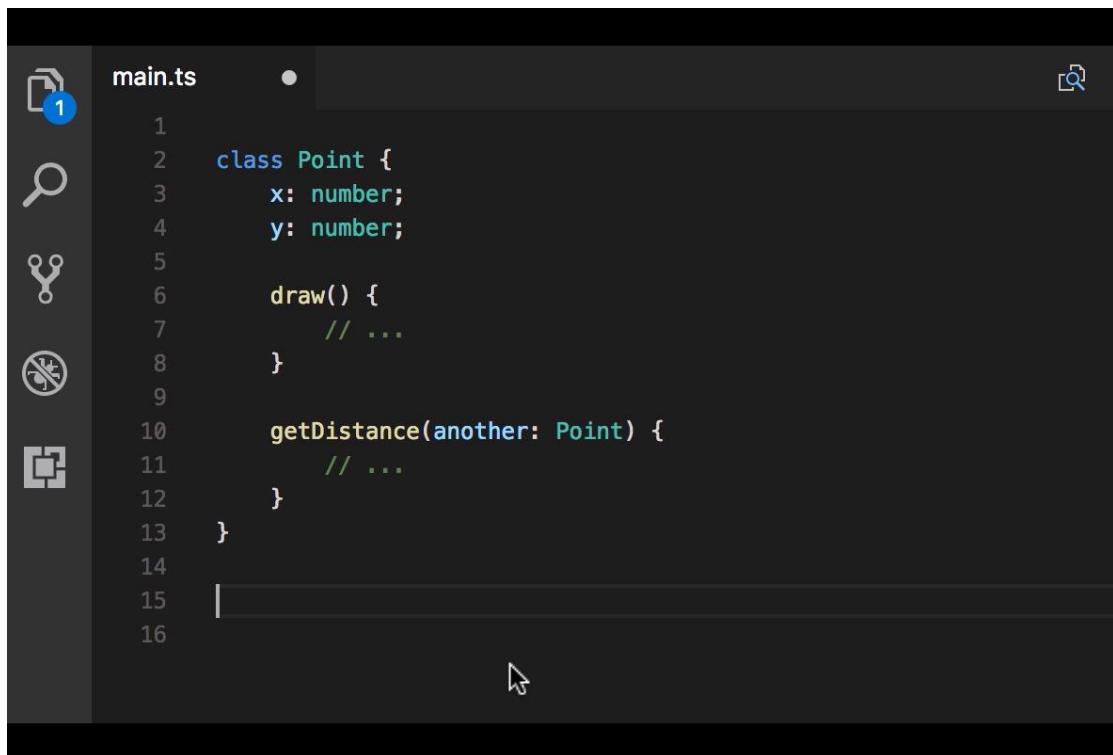
```
1  interface Point {  
2      x: number,  
3      y: number,  
4      draw: () => void  
5  }  
6  
7  let drawPoint = (point: Point) => {  
8      // ...  
9  }  
10  
11  let getDistance = (pointA: Point, pointB: Point) => {  
12      // ...  
13  }  
14  
15  drawPoint({  
16      x: 1,  
17      y: 2  
18  })
```

The 'draw' method in the 'Point' interface is highlighted with a yellow rectangular selection. In the code below, when 'drawPoint' is called with an argument, the 'x' and 'y' properties are underlined with red squiggly lines, indicating they are not defined in the interface declaration.



main.ts

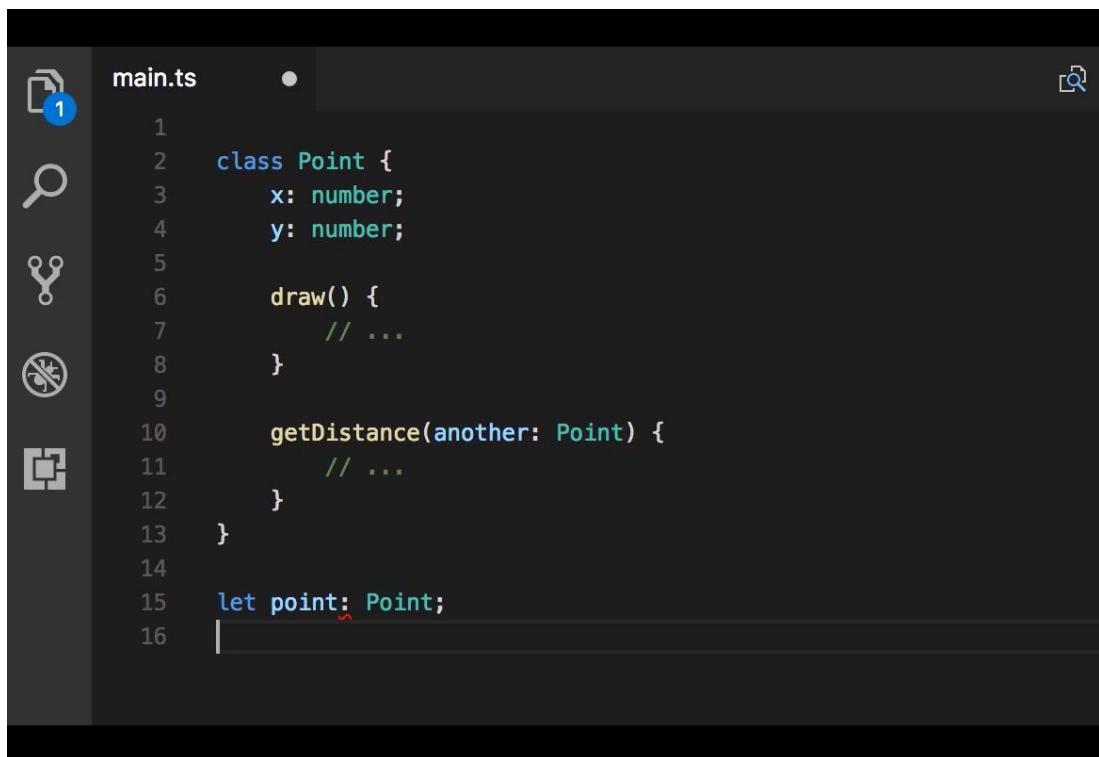
```
1
2  class Point {
3      x: number;
4      y: number;
5
6      draw() {
7          // ...
8      }
9
10     getDistance(another: Point) {
11         // ...
12     }
13 }
14
15 let drawPoint = (point: Point) => {
16     // ...
17 }
```



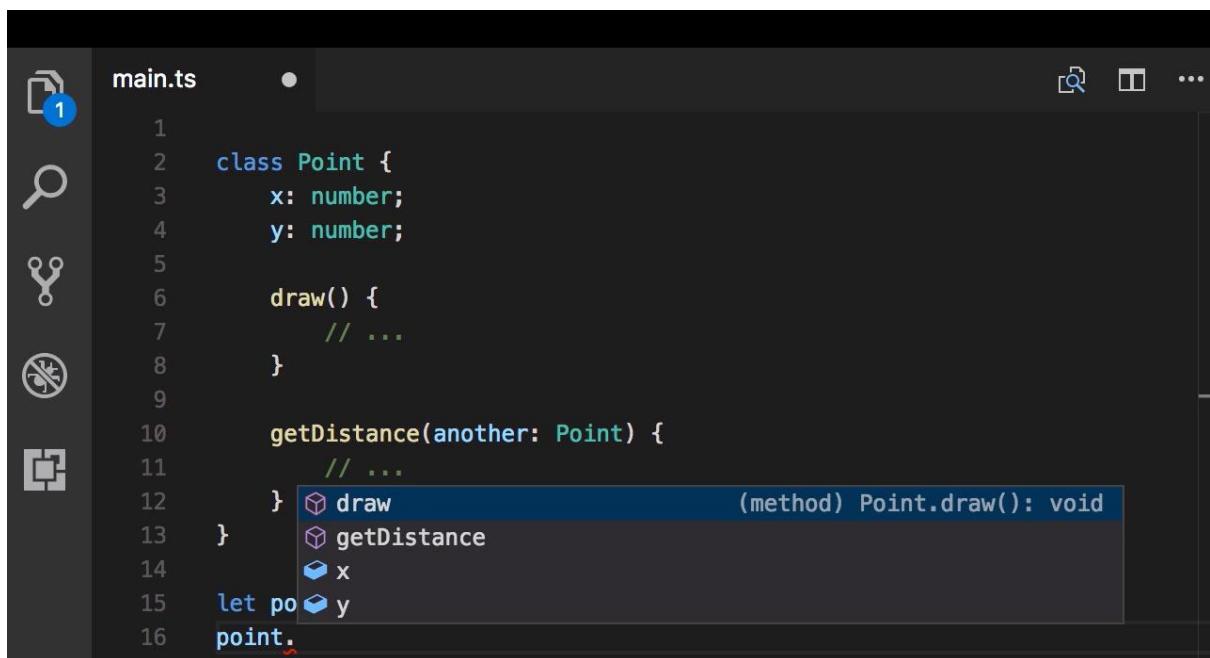
main.ts

```
1
2  class Point {
3      x: number;
4      y: number;
5
6      draw() {
7          // ...
8      }
9
10     getDistance(another: Point) {
11         // ...
12     }
13 }
14
15
16 
```

Objects



```
main.ts
1
2  class Point {
3      x: number;
4      y: number;
5
6      draw() {
7          // ...
8      }
9
10     getDistance(another: Point) {
11         // ...
12     }
13 }
14
15 let point: Point;
16
```



```
main.ts
1
2  class Point {
3      x: number;
4      y: number;
5
6      draw() {
7          // ...
8      }
9
10     getDistance(another: Point) {
11         // ...
12     }
13 }
14
15 let poynt;
16
```

The code completion dropdown shows:

- draw (method) Point.draw(): void
- getDistance
- x
- y

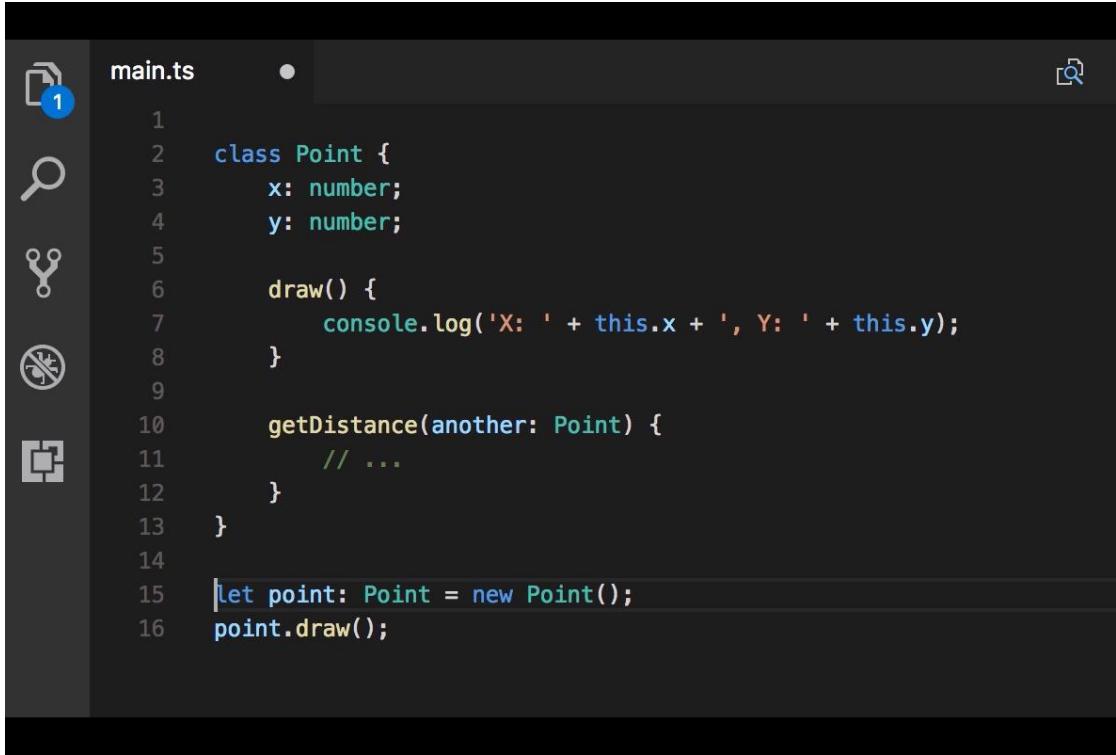
The screenshot shows the VS Code interface with the file 'main.ts' open. The code defines a class 'Point' with properties x and y, a draw method that logs the coordinates to the console, and a getDistance method. It also declares a variable 'point' and calls its draw method.

```
1  class Point {
2      x: number;
3      y: number;
4
5      draw() {
6          console.log('X: ' + this.x + ', Y: ' + this.y);
7      }
8
9      getDistance(another: Point) {
10         // ...
11     }
12 }
13
14 let point: Point;
15 point.draw();
```

The terminal window shows the command \$tsc main.ts being run, followed by the output of the compiled JavaScript code. When the script is executed with node main.js, it results in a TypeError because 'point' is undefined at the line point.draw();.

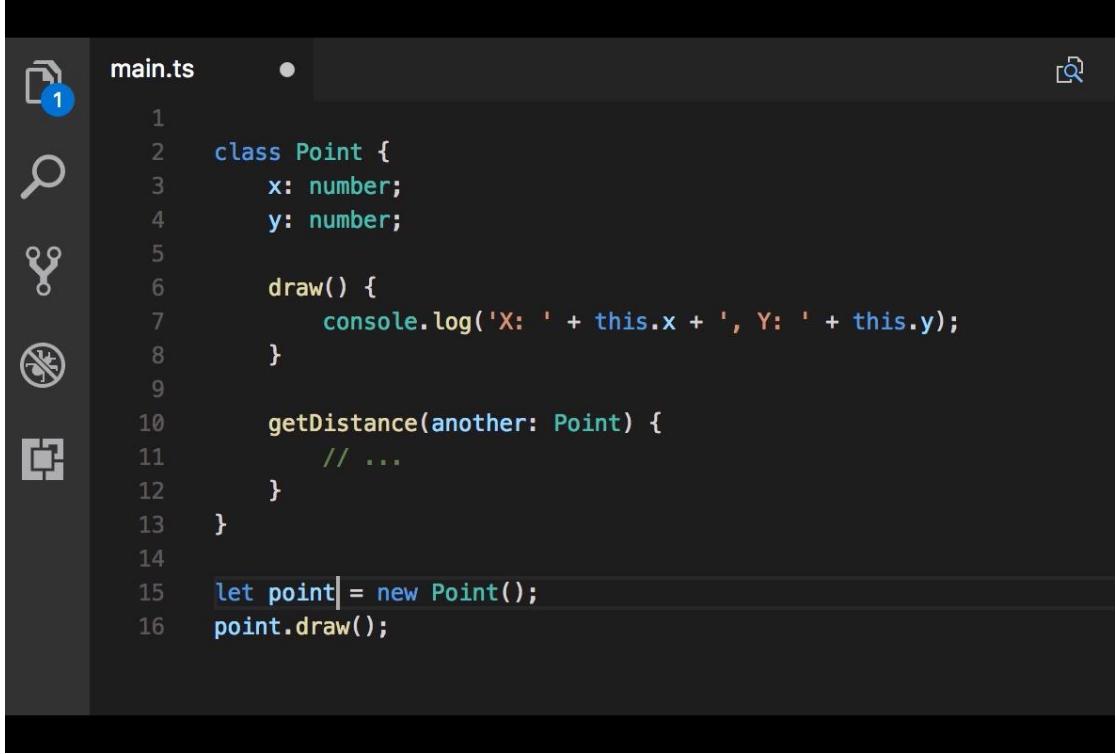
```
ts-hello $ tsc main.ts | node main.js
/Users/moshfeghhamedani/Desktop/Angular 4/code/ts-hello/main.js:13
point.draw();
^

TypeError: Cannot read property 'draw' of undefined
    at Object.<anonymous> (/Users/moshfeghhamedani/Desktop/Angular 4/code/ts-hello/main.js:13:6)
    at Module._compile (module.js:570:32)
    at Object.Module._extensions..js (module.js:579:10)
    at Module.load (module.js:487:32)
    at tryModuleLoad (module.js:446:12)
    at Function.Module._load (module.js:438:3)
    at Module.runMain (module.js:604:10)
    at run (bootstrap_node.js:390:7)
    at startup (bootstrap_node.js:150:9)
    at bootstrap_node.js:505:3
ts-hello $
```



```
main.ts ●

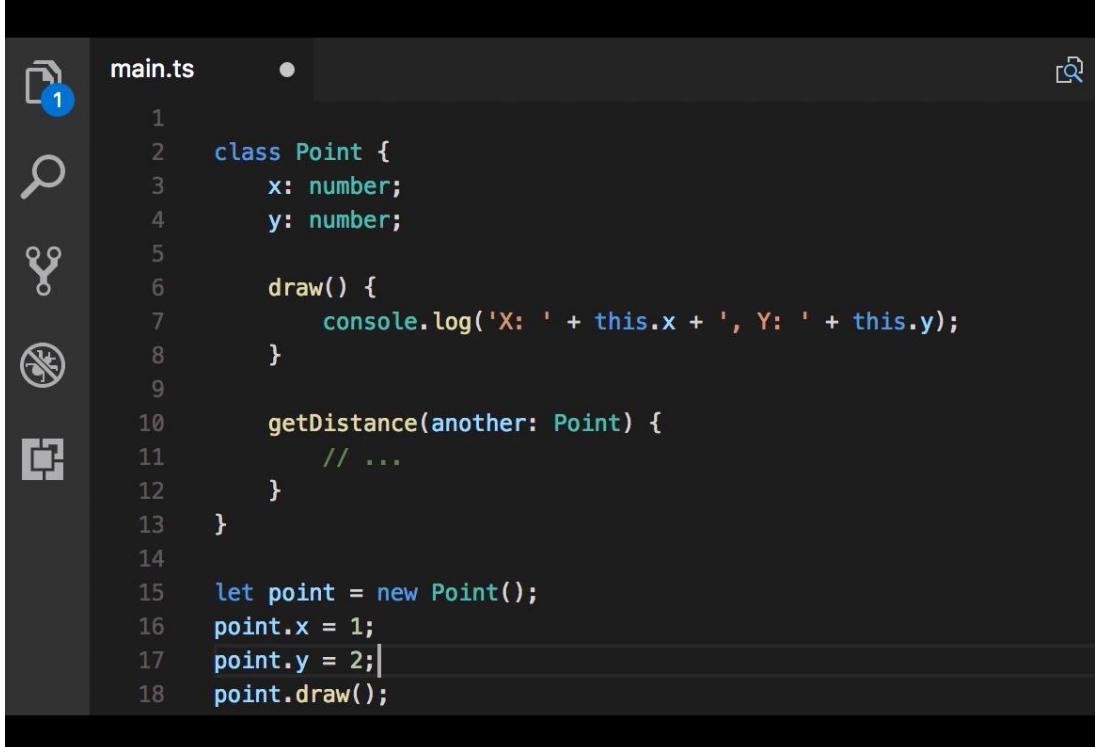
1
2  class Point {
3      x: number;
4      y: number;
5
6      draw() {
7          console.log('X: ' + this.x + ', Y: ' + this.y);
8      }
9
10     getDistance(another: Point) {
11         // ...
12     }
13 }
14
15 let point: Point = new Point();
16 point.draw();
```



```
main.ts ●

1
2  class Point {
3      x: number;
4      y: number;
5
6      draw() {
7          console.log('X: ' + this.x + ', Y: ' + this.y);
8      }
9
10     getDistance(another: Point) {
11         // ...
12     }
13 }
14
15 let point| = new Point();
16 point.draw();
```

```
ts-hello $  
ts-hello $tsc main.ts  
ts-hello $node main.js  
X: undefined, Y: undefined  
ts-hello $
```

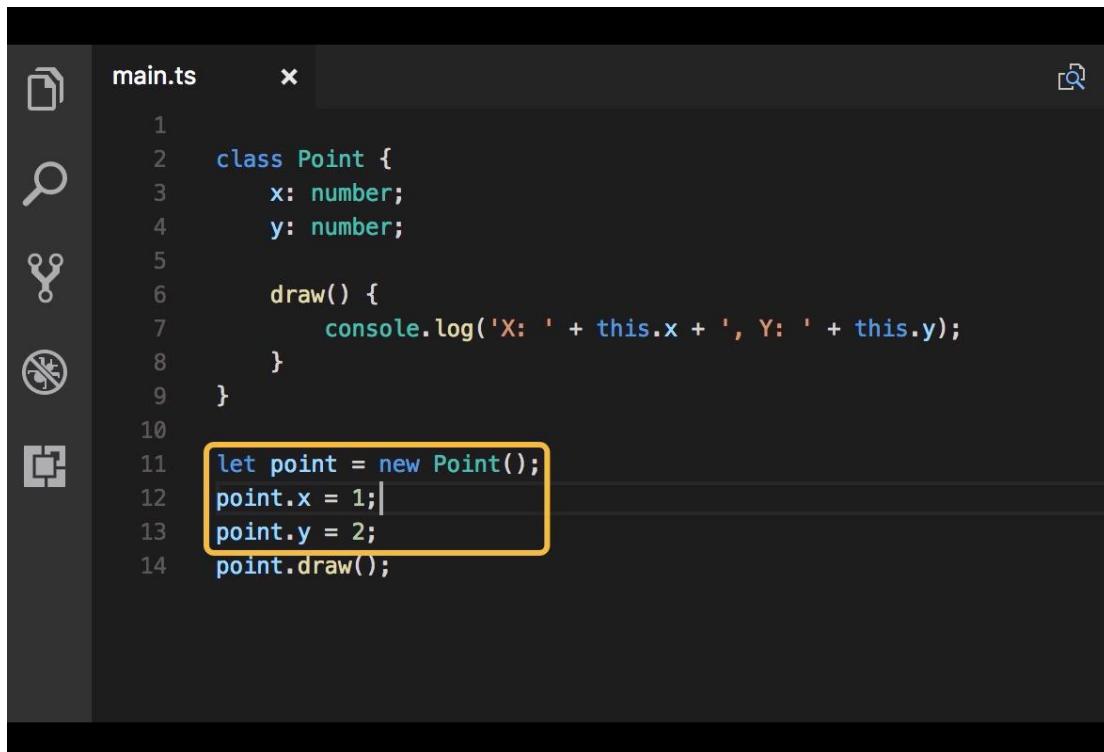


The screenshot shows a code editor interface with a dark theme. On the left is a sidebar containing icons for file operations (new file, save, search, find, copy, paste, etc.). The main area displays the contents of a file named `main.ts`. The code defines a `Point` class with properties `x` and `y`, a `draw` method that logs the coordinates to the console, and a `getDistance` method. It also creates a new `Point` object and calls its `draw` method.

```
1  
2  class Point {  
3      x: number;  
4      y: number;  
5  
6      draw() {  
7          console.log('X: ' + this.x + ', Y: ' + this.y);  
8      }  
9  
10     getDistance(another: Point) {  
11         // ...  
12     }  
13 }  
14  
15 let point = new Point();  
16 point.x = 1;  
17 point.y = 2;  
18 point.draw();
```

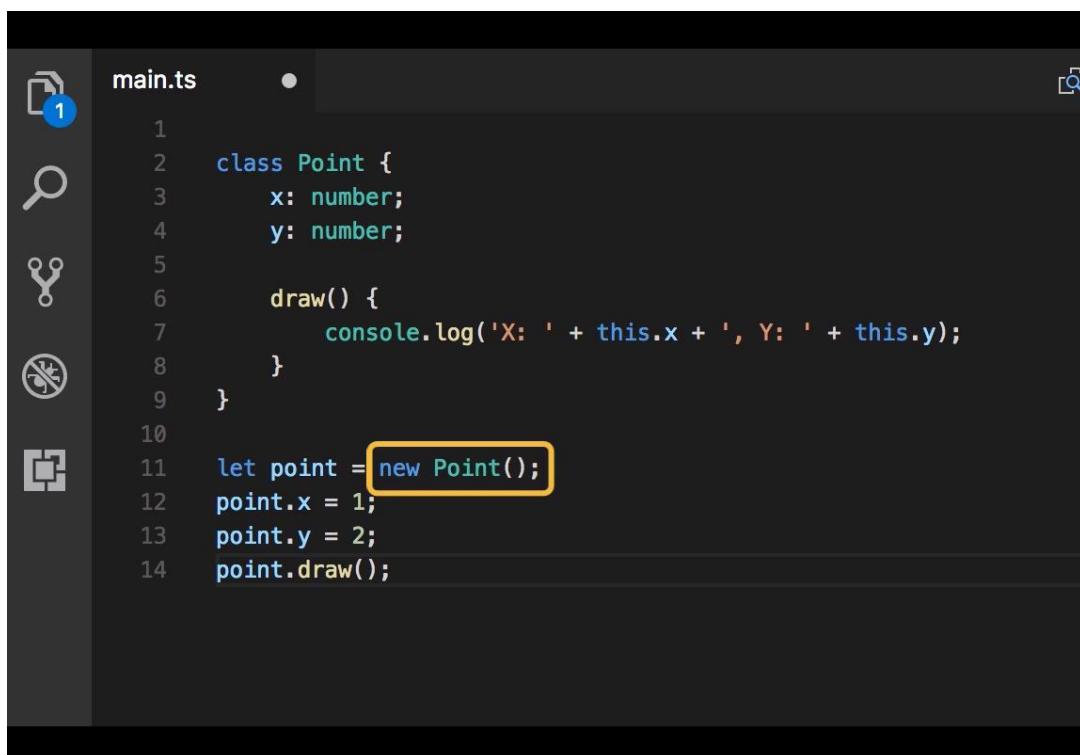
```
ts-hello $  
ts-hello $tsc main.ts && node main.js  
X: 1, Y: 2  
ts-hello $
```

Constructors



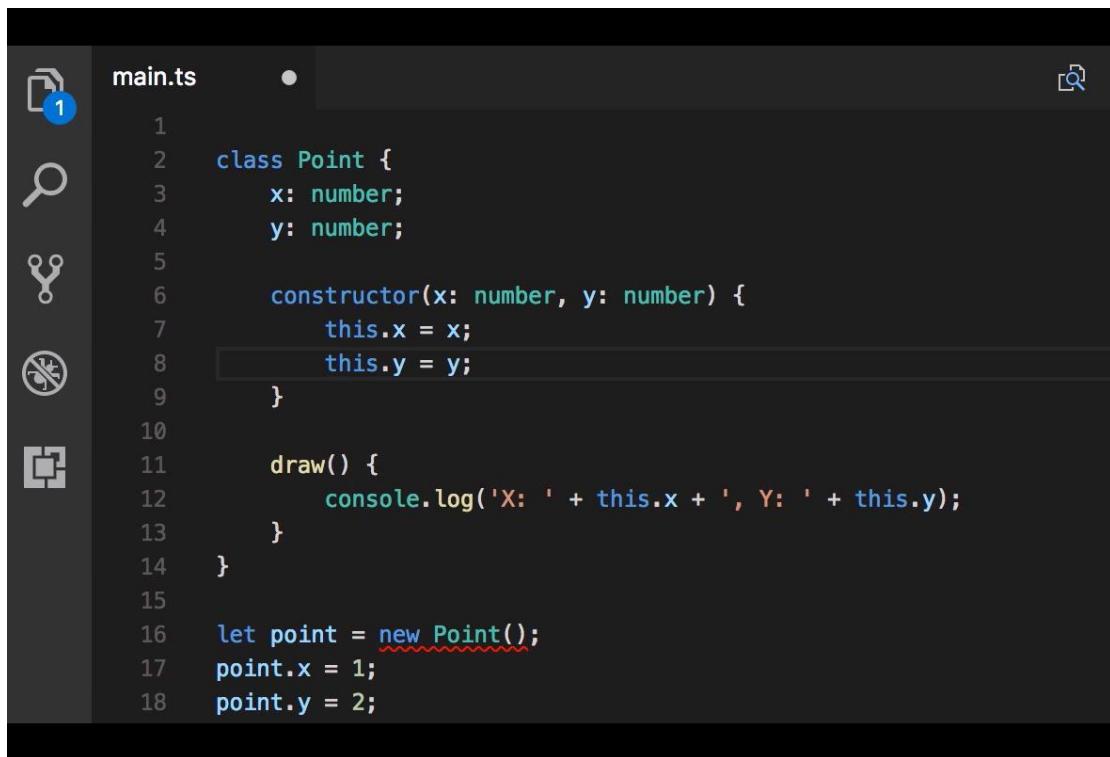
A screenshot of a code editor window titled "main.ts". The code defines a "Point" class with properties "x" and "y", and a "draw" method that logs the coordinates to the console. Below the class definition, there is a block of code that creates a new "Point" object, sets its "x" and "y" properties to 1 and 2 respectively, and then calls its "draw" method. The line "let point = new Point();" is highlighted with a yellow box.

```
1
2  class Point {
3      x: number;
4      y: number;
5
6      draw() {
7          console.log('X: ' + this.x + ', Y: ' + this.y);
8      }
9  }
10
11 let point = new Point();
12 point.x = 1;
13 point.y = 2;
14 point.draw();
```

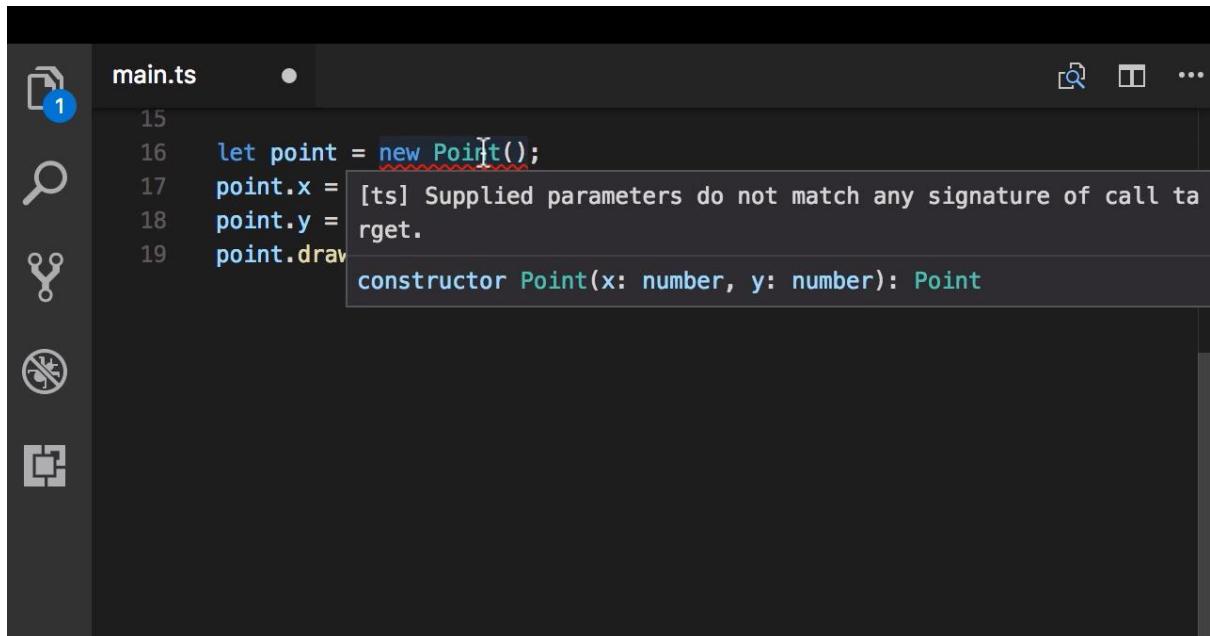


A screenshot of a code editor window titled "main.ts". The code is identical to the one in the previous screenshot, defining a "Point" class and creating a new instance with x=1 and y=2, then calling its "draw" method. The line "let point = new Point();" is highlighted with a yellow box. A blue circular icon with the number "1" is visible in the top-left corner of the editor area.

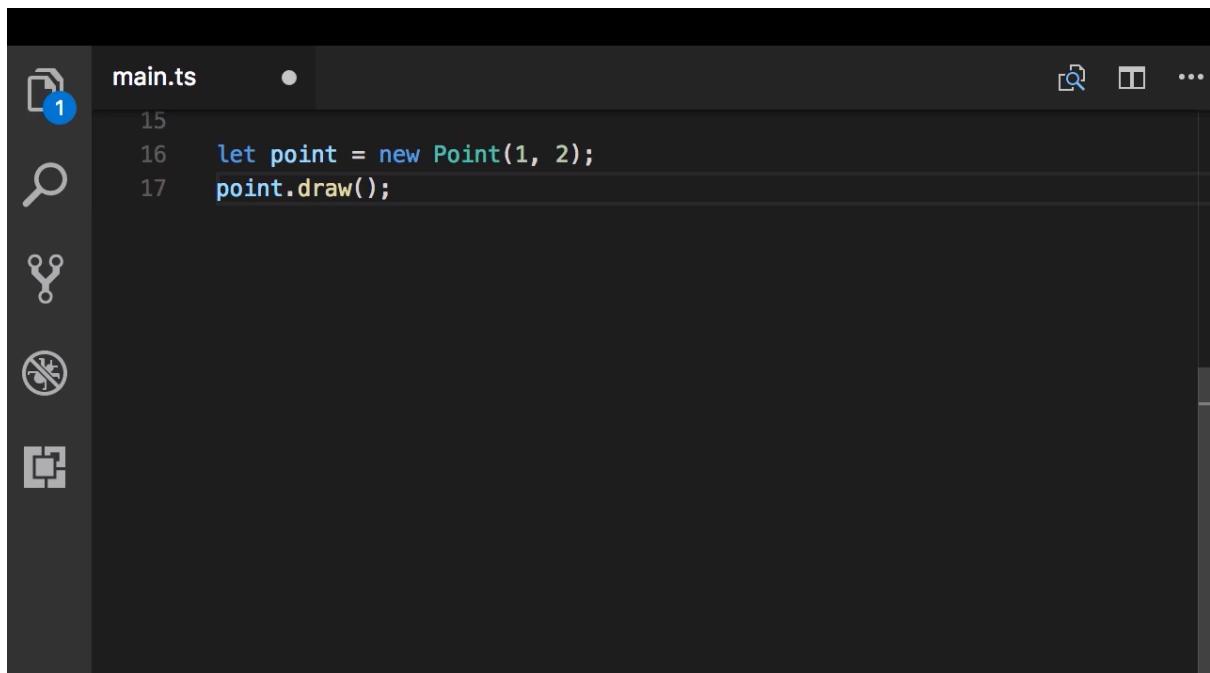
```
1
2  class Point {
3      x: number;
4      y: number;
5
6      draw() {
7          console.log('X: ' + this.x + ', Y: ' + this.y);
8      }
9  }
10
11 let point = new Point();
12 point.x = 1;
13 point.y = 2;
14 point.draw();
```



```
main.ts
1
2  class Point {
3      x: number;
4      y: number;
5
6      constructor(x: number, y: number) {
7          this.x = x;
8          this.y = y;
9      }
10
11     draw() {
12         console.log('X: ' + this.x + ', Y: ' + this.y);
13     }
14 }
15
16 let point = new Point();
17 point.x = 1;
18 point.y = 2;
```

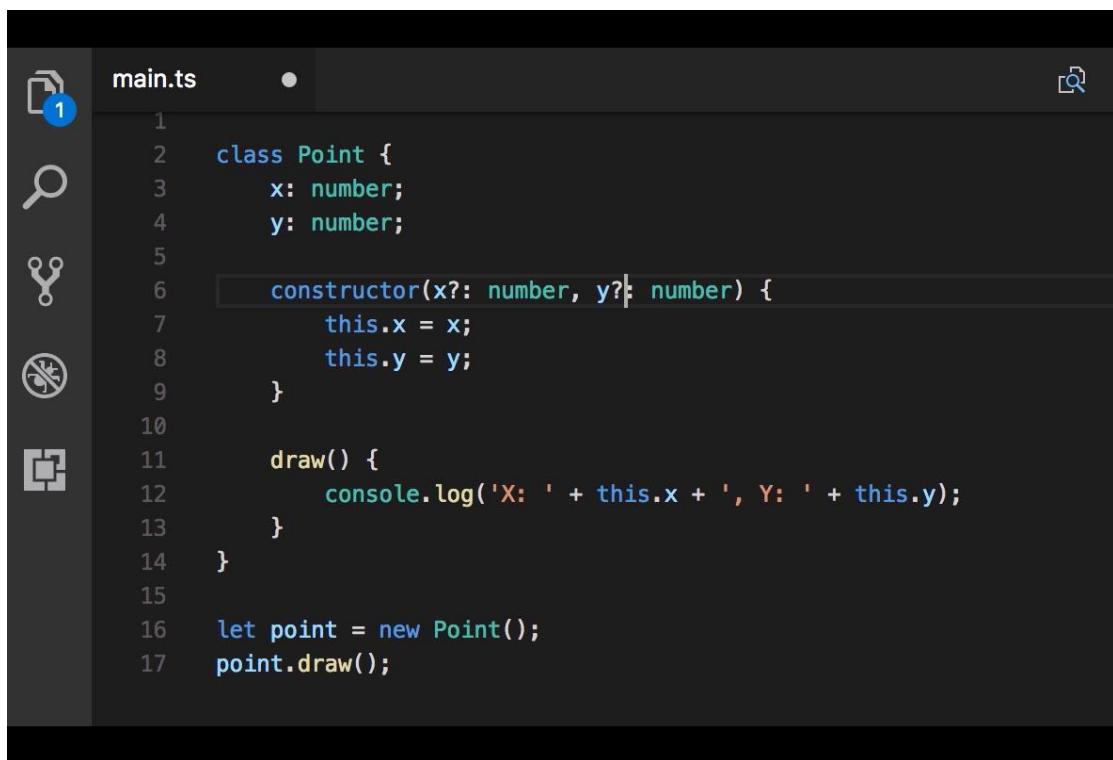


```
main.ts
15
16 let point = new Point();
17 point.x =
18 point.y =
19 point.draw
    constructor Point(x: number, y: number): Point
```



main.ts

```
15
16 let point = new Point(1, 2);
17 point.draw();
```



main.ts

```
1
2 class Point {
3     x: number;
4     y: number;
5
6     constructor(x?: number, y?: number) {
7         this.x = x;
8         this.y = y;
9     }
10
11    draw() {
12        console.log('X: ' + this.x + ', Y: ' + this.y);
13    }
14 }
15
16 let point = new Point();
17 point.draw();
```

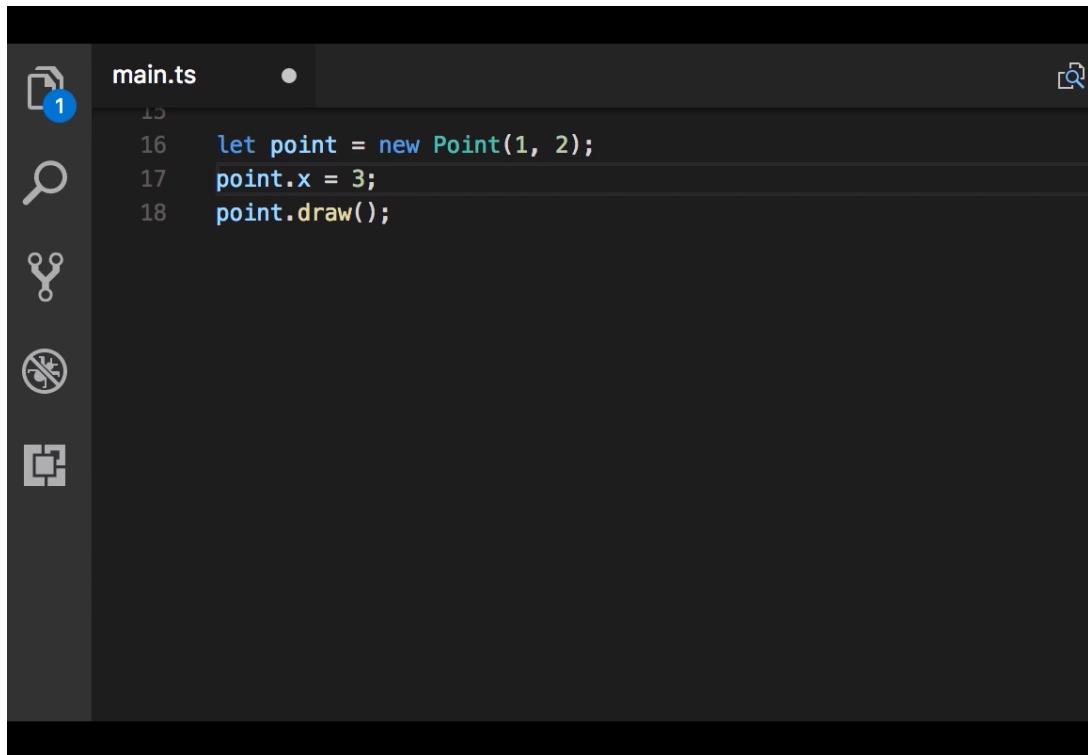
All the parameters on the right side of the first optional parameters should also be optional.

The screenshot shows a code editor interface with a dark theme. On the left, there is a vertical toolbar with several icons: a document icon with a '1' (highlighted), a magnifying glass, a gear, a refresh/circular arrow, and a copy/paste icon. The main area is titled 'main.ts'. The code is as follows:

```
1
2  class Point {
3      x: number;
4      y: number;
5
6      constructor(x?: number, y?: number) {
7          this.x = x;
8          this.y = y;
9      }
10
11     draw() {
12         console.log('X: ' + this.x + ', Y: ' + this.y);
13     }
14 }
15
16 let point = new Point();
17 point.draw();
```

The line 'let point = new Point();' is highlighted with a yellow rectangular selection.

Access Modifiers

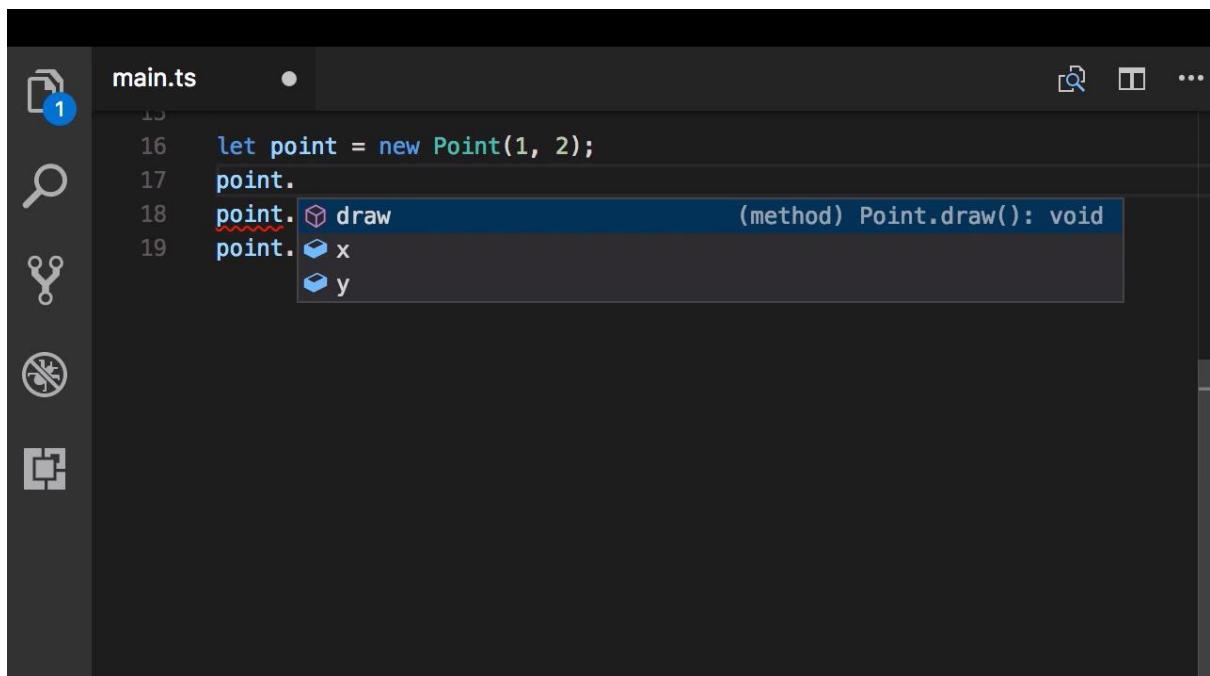


```
main.ts
16 let point = new Point(1, 2);
17 point.x = 3;
18 point.draw();
```

Access Modifiers

- public
- private
- protected

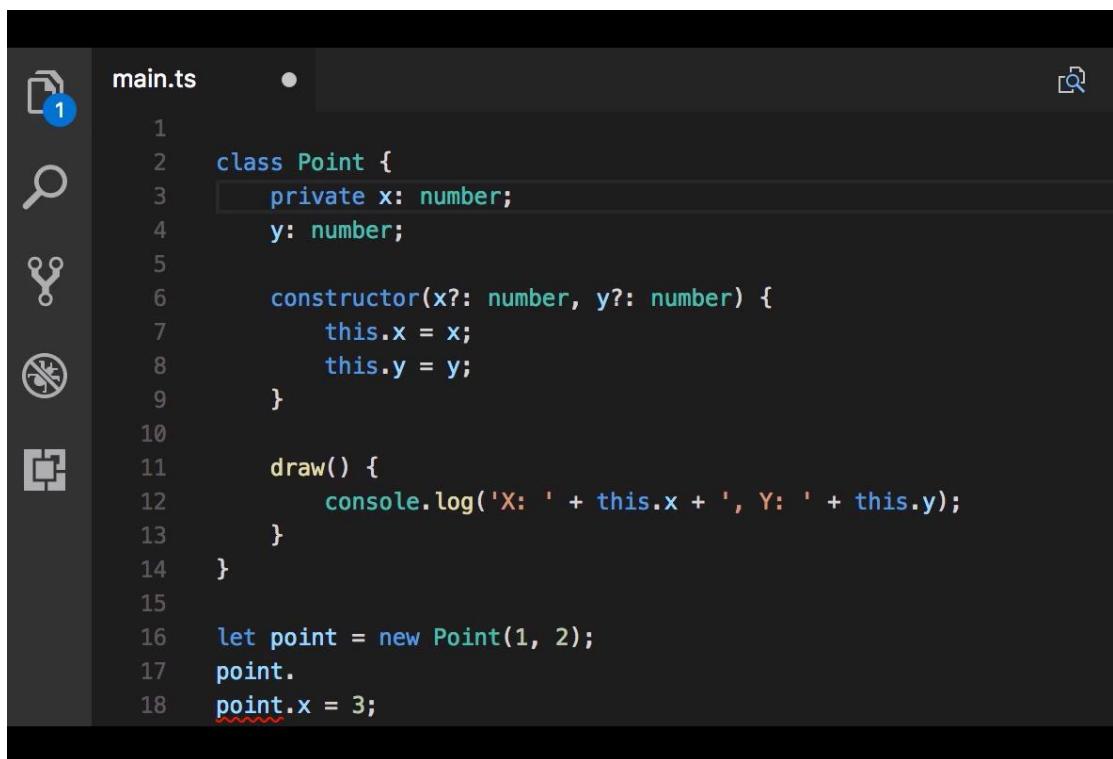
By default all public



A screenshot of a dark-themed code editor window titled "main.ts". The code editor shows the following TypeScript code:

```
15
16 let point = new Point(1, 2);
17 point.
18 point.draw          (method) Point.draw(): void
19 point.x
20 point.y
```

A tooltip is displayed over the line "point.draw", showing the method signature "(method) Point.draw(): void". The tooltip also includes small icons representing the method and its parameters.



A screenshot of a dark-themed code editor window titled "main.ts". The code editor shows the following TypeScript code:

```
1
2 class Point {
3     private x: number;
4     y: number;
5
6     constructor(x?: number, y?: number) {
7         this.x = x;
8         this.y = y;
9     }
10
11    draw() {
12        console.log('X: ' + this.x + ', Y: ' + this.y);
13    }
14 }
15
16 let point = new Point(1, 2);
17 point.
18 point.x = 3;
```

A tooltip is displayed over the line "point.x", showing the assignment operation "point.x = 3;".

A screenshot of the Visual Studio Code interface. The left sidebar shows icons for file, search, and other tools. The main editor window is titled "main.ts". The code is as follows:

```
15  
16 let point = new Point(1, 2);  
17 point.  
18 point. draw (property) Point.y: number
```

The word "draw" is underlined with a red squiggle, indicating it is a misspelling or undefined. The tooltip for "y" shows it is a property of type "number".

A screenshot of the Visual Studio Code interface. The left sidebar shows icons for file, search, and other tools. The main editor window is titled "main.ts". The code is as follows:

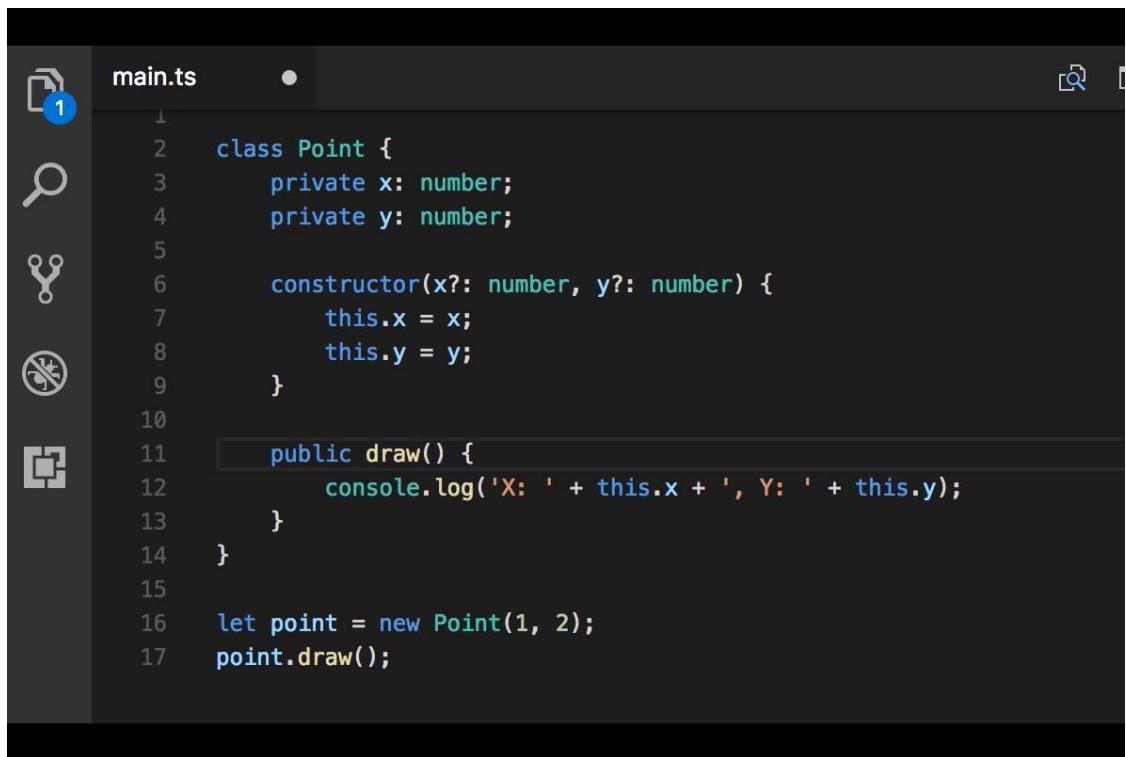
```
15  
16 let point = new Point(1, 2);  
17 point.x = 3;  
18 point. x [ts] Property 'x' is private and only accessible within class 'Point'.  
(property) Point.x: number
```

The line "point.x = 3;" is highlighted with a red underline, indicating a TypeScript error. A tooltip above the cursor explains that the property "x" is private and can only be accessed within its class.

```
main.ts
1
2  class Point {
3      private x: number;
4      private y: number;
5
6      constructor(x?: number, y?: number) {
7          this.x = x;
8          this.y = y;
9      }
10
11     draw() {
12         console.log('X: ' + this.x + ', Y: ' + this.y);
13     }
14 }
15
16 let point = new Point(1, 2);
17 point.x = 3;
18 point.draw();
```

```
main.ts
15
16 let point = new Point(1, 2);
17 point.|
```

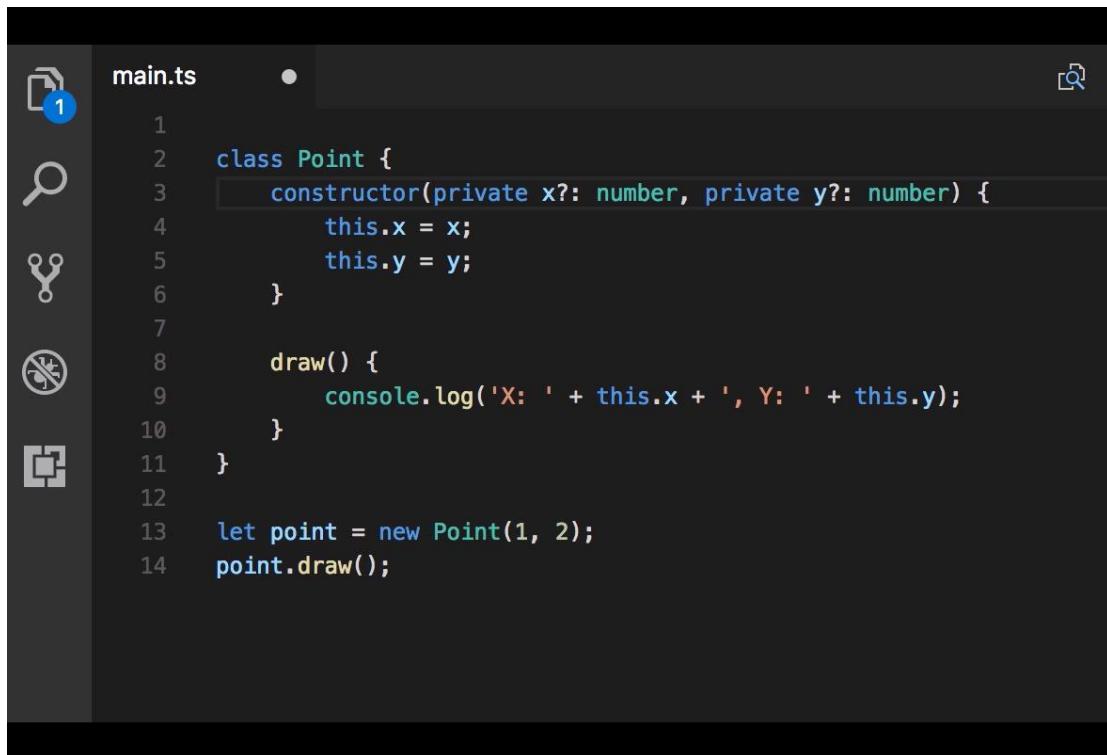
point. **draw** (method) Point.draw(): void



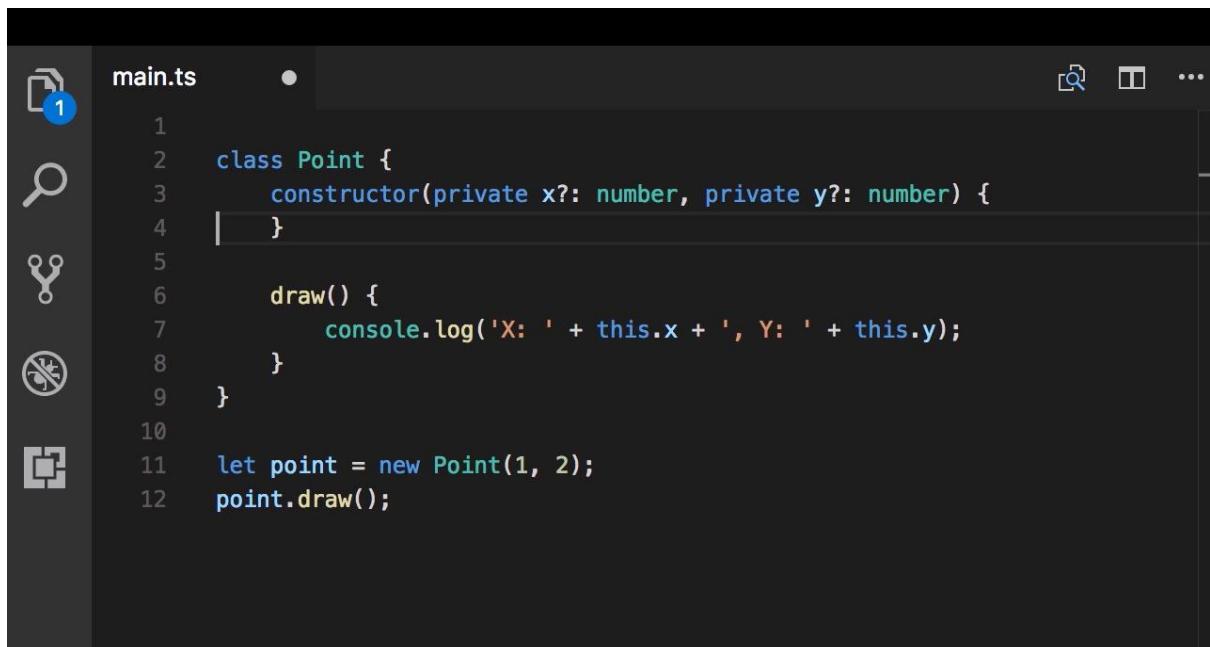
The screenshot shows a code editor interface with a dark theme. On the left is a vertical toolbar containing icons for file operations (new, open, save, close), search, and other developer tools. The main area displays a file named "main.ts". The code is as follows:

```
1
2  class Point {
3      private x: number;
4      private y: number;
5
6      constructor(x?: number, y?: number) {
7          this.x = x;
8          this.y = y;
9      }
10
11     public draw() {
12         console.log('X: ' + this.x + ', Y: ' + this.y);
13     }
14 }
15
16 let point = new Point(1, 2);
17 point.draw();
```

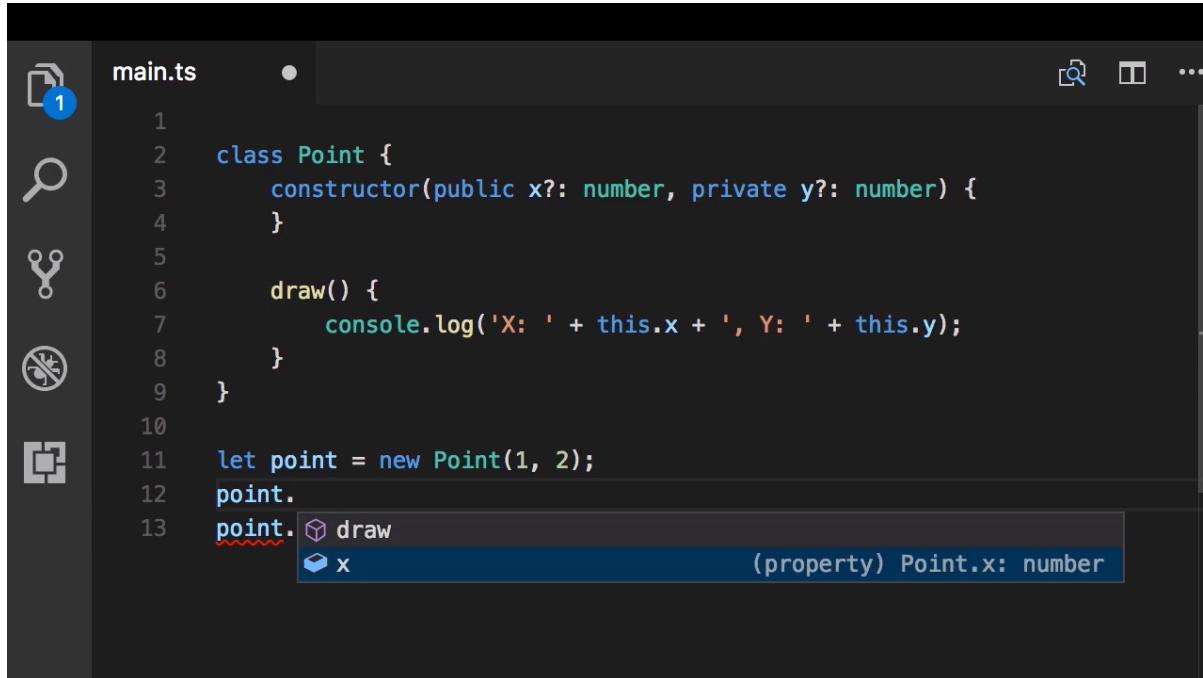
Access Modifiers in Constructor Parameters



```
main.ts
1
2 class Point {
3     constructor(private x?: number, private y?: number) {
4         this.x = x;
5         this.y = y;
6     }
7
8     draw() {
9         console.log('X: ' + this.x + ', Y: ' + this.y);
10    }
11 }
12
13 let point = new Point(1, 2);
14 point.draw();
```



```
main.ts
1
2 class Point {
3     constructor(private x?: number, private y?: number) {
4
5     draw() {
6         console.log('X: ' + this.x + ', Y: ' + this.y);
7     }
8 }
9
10 let point = new Point(1, 2);
11 point.draw();
```

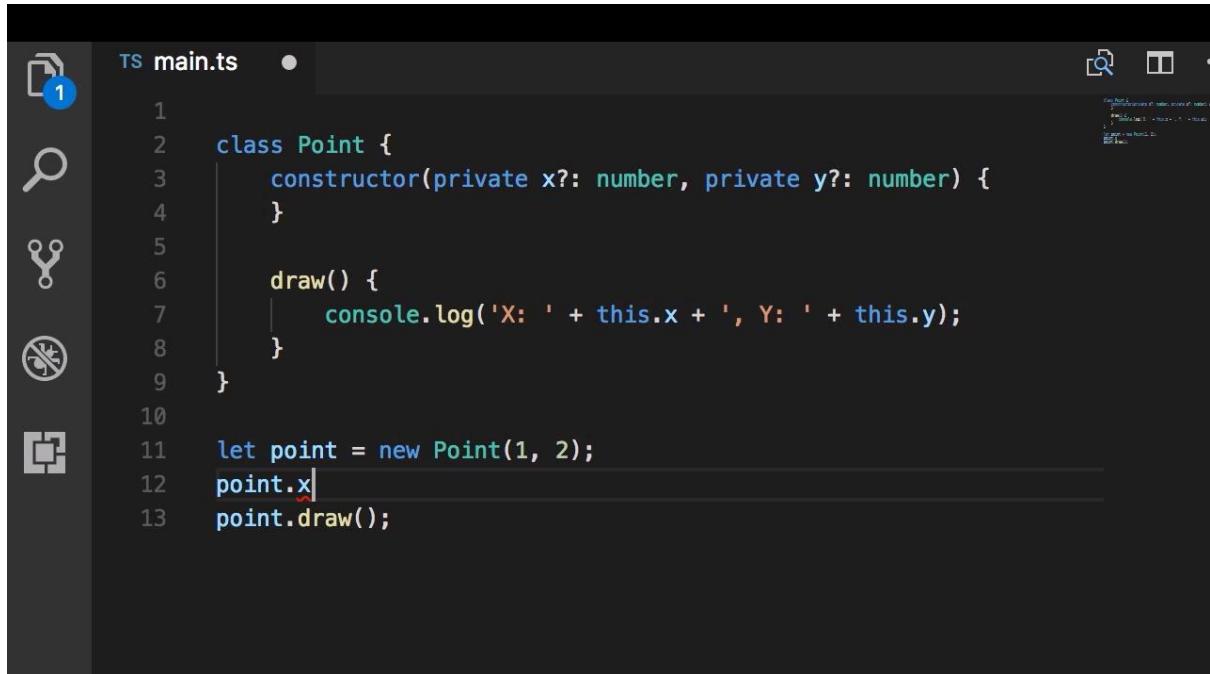


The screenshot shows a dark-themed interface of the Visual Studio Code (VS Code) code editor. On the left, there is a vertical toolbar with several icons: a document with a '1' (File), a magnifying glass (Search), a gear (Tools), a speaker with a slash (Notifications), and a square (New Tab). The main area is titled "main.ts". The code editor displays the following TypeScript code:

```
1
2 class Point {
3     constructor(public x?: number, private y?: number) {
4         }
5
6     draw() {
7         console.log('X: ' + this.x + ', Y: ' + this.y);
8     }
9 }
10
11 let point = new Point(1, 2);
12 point.
13 point. ⚡ draw
          (property) Point.x: number
```

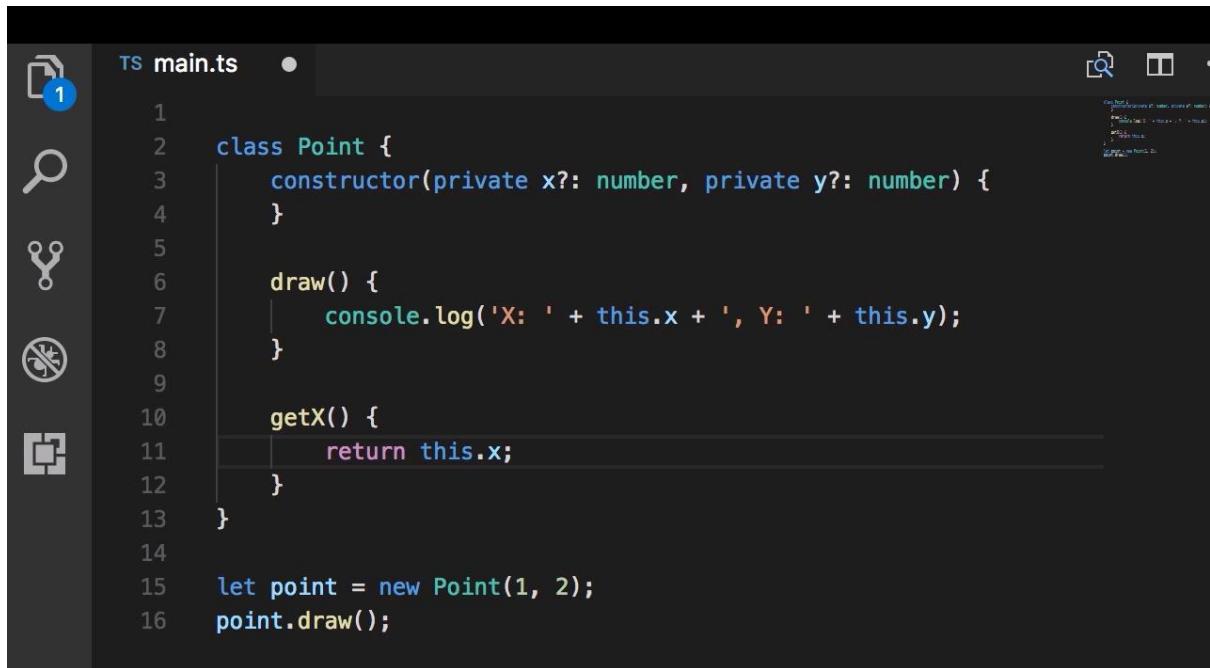
A tooltip is visible over the "draw" method, showing its signature: "(property) Point.x: number".

Properties



TS main.ts

```
1
2 class Point {
3     constructor(private x?: number, private y?: number) {
4         }
5
6     draw() {
7         console.log('X: ' + this.x + ', Y: ' + this.y);
8     }
9 }
10 let point = new Point(1, 2);
11 point.x
12 point.draw();
```



TS main.ts

```
1
2 class Point {
3     constructor(private x?: number, private y?: number) {
4         }
5
6     draw() {
7         console.log('X: ' + this.x + ', Y: ' + this.y);
8     }
9
10    getX() {
11        return this.x;
12    }
13 }
14
15 let point = new Point(1, 2);
16 point.draw();
```

```
TS main.ts ●
```

```
1
2  class Point {
3      constructor(private x?: number, private y?: number) {
4          }
5
6      draw() {
7          console.log('X: ' + this.x + ', Y: ' + this.y);
8      }
9
10     getX() {
11         return this.x;
12     }
13 }
14
15 let point = new Point(1, 2);
16 let x = point.getX();
17 point.draw();
```

```
TS main.ts ●
```

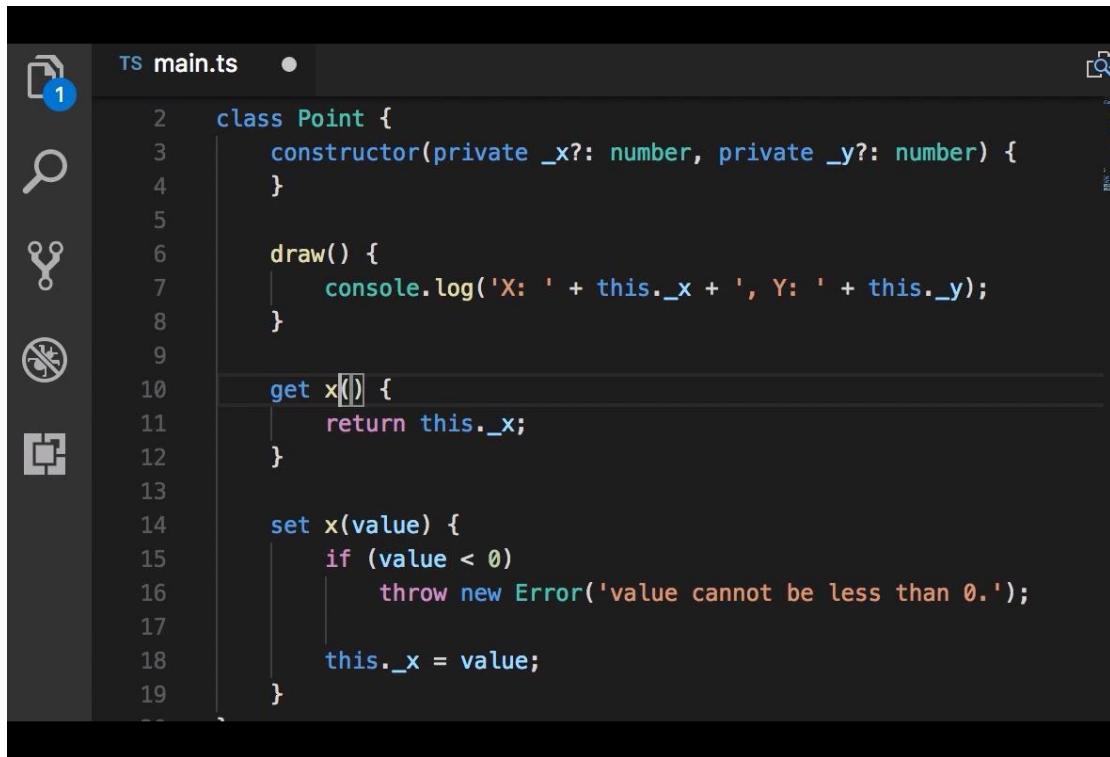
```
10     getX() {
11         return this.x;
12     }
13
14     setX(value) {
15         if (value < 0)
16             throw new Error('value cannot be less than 0.');
17         this.x = value;
18     }
19 }
20
21 let point = new Point(1, 2);
22 let x = point.getX();
23 point.setX(10);
24 point.draw();
```

TS main.ts

```
10     get X() {
11         return this.x;
12     }
13
14     set X(value) {
15         if (value < 0)
16             throw new Error('value cannot be less than 0.');
17         this.x = value;
18     }
19 }
20
22 let point = new Point(1, 2);
23 let x = point.getX();
24 point.setX(10);
25 point.draw();
```

TS main.ts

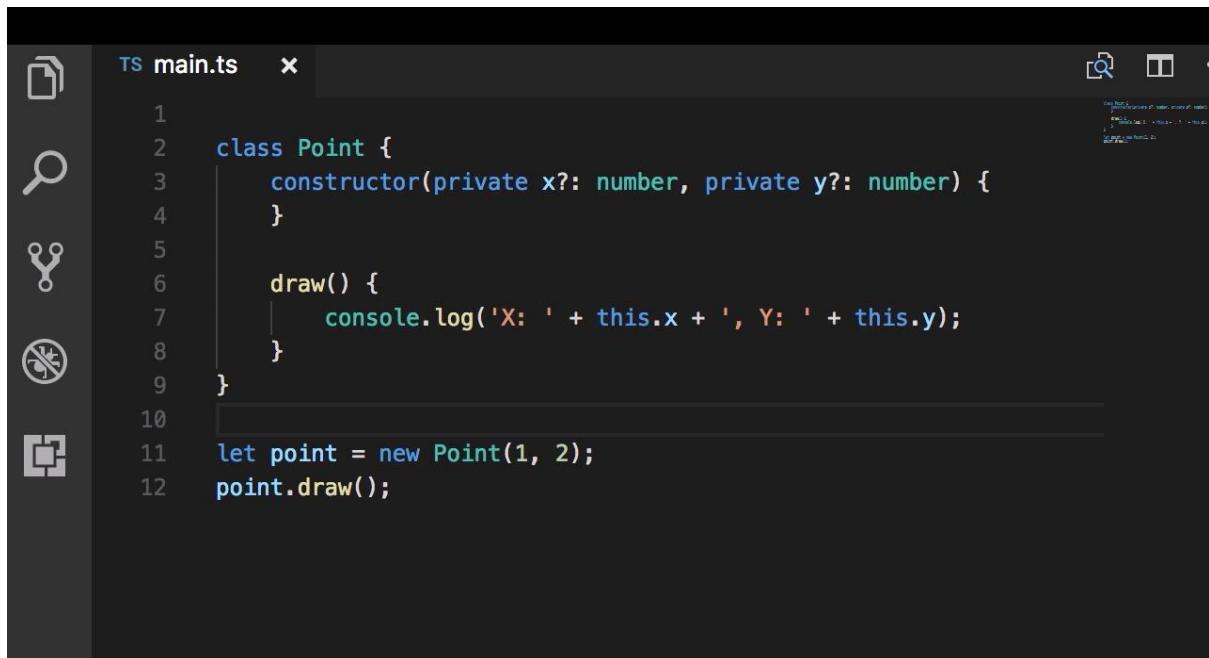
```
19 }
20 }
21
22 let point = new Point(1, 2);
23 let x = point.X;
24 point.setX(10);
25
26 point.draw();
```



```
TS main.ts ●

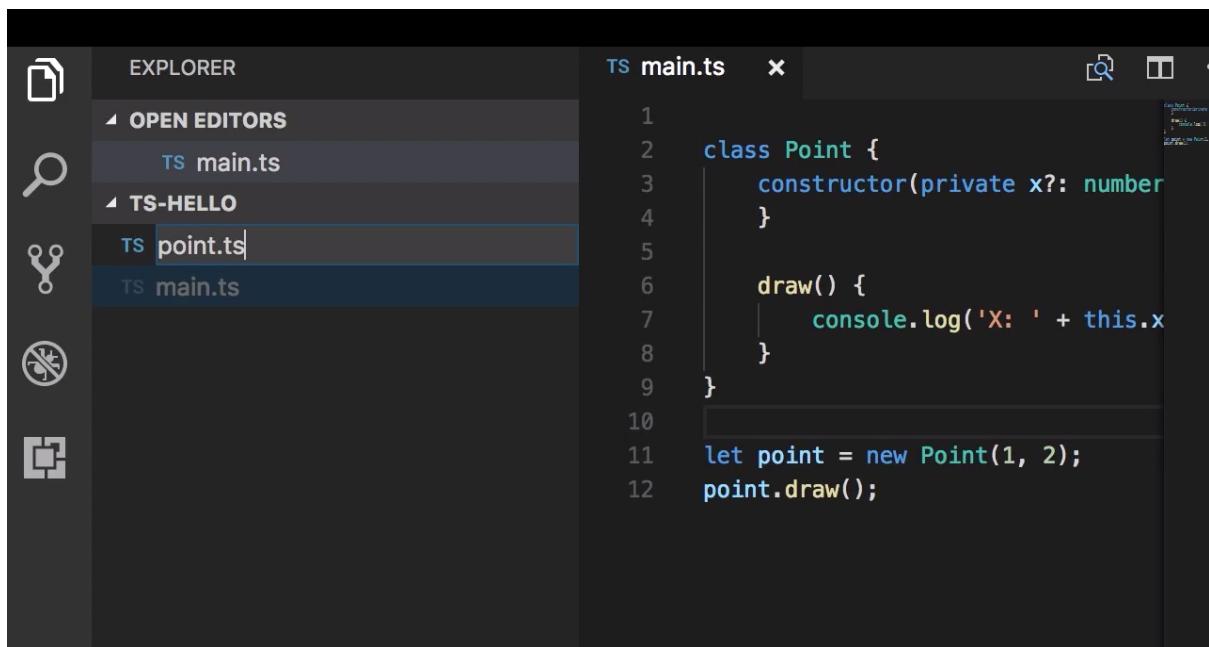
2  class Point {
3      constructor(private _x?: number, private _y?: number) {
4          }
5
6      draw() {
7          console.log('X: ' + this._x + ', Y: ' + this._y);
8      }
9
10     get x(): number {
11         return this._x;
12     }
13
14     set x(value) {
15         if (value < 0)
16             throw new Error('value cannot be less than 0.');
17         this._x = value;
18     }
19 }
```

Modules



A screenshot of the Visual Studio Code interface showing a single editor tab titled "main.ts". The code in the editor is:

```
1
2 class Point {
3     constructor(private x?: number, private y?: number) {
4         }
5
6     draw() {
7         console.log('X: ' + this.x + ', Y: ' + this.y);
8     }
9 }
10
11 let point = new Point(1, 2);
12 point.draw();
```



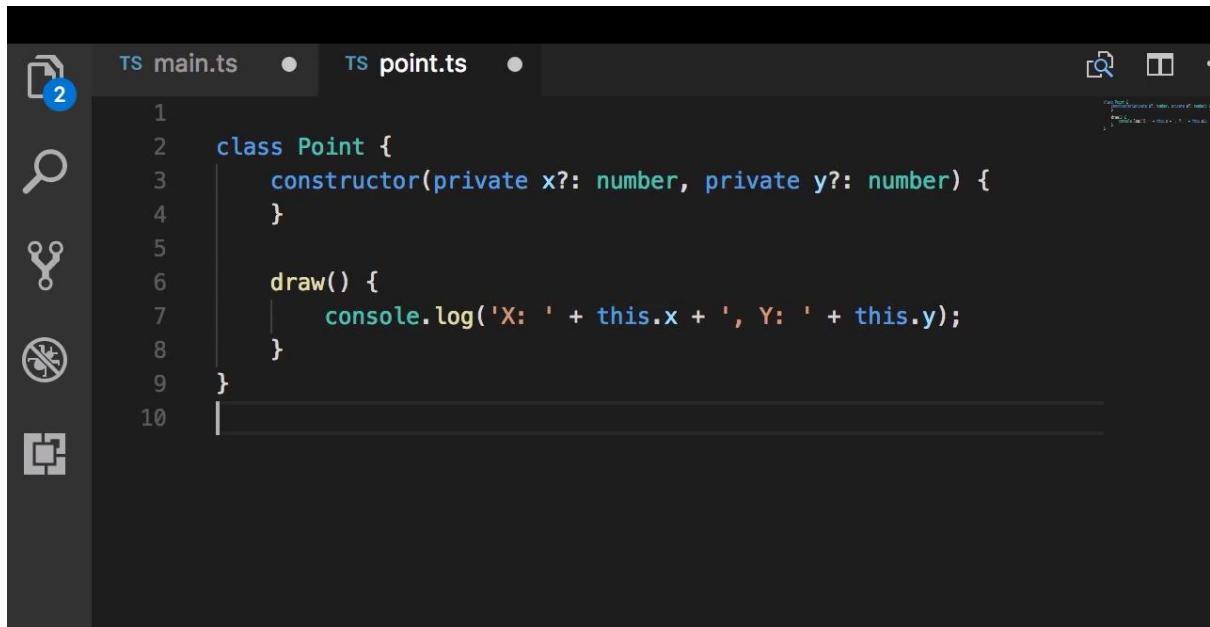
A screenshot of the Visual Studio Code interface showing multiple editor tabs and the Explorer sidebar.

The Explorer sidebar on the left shows the following structure:

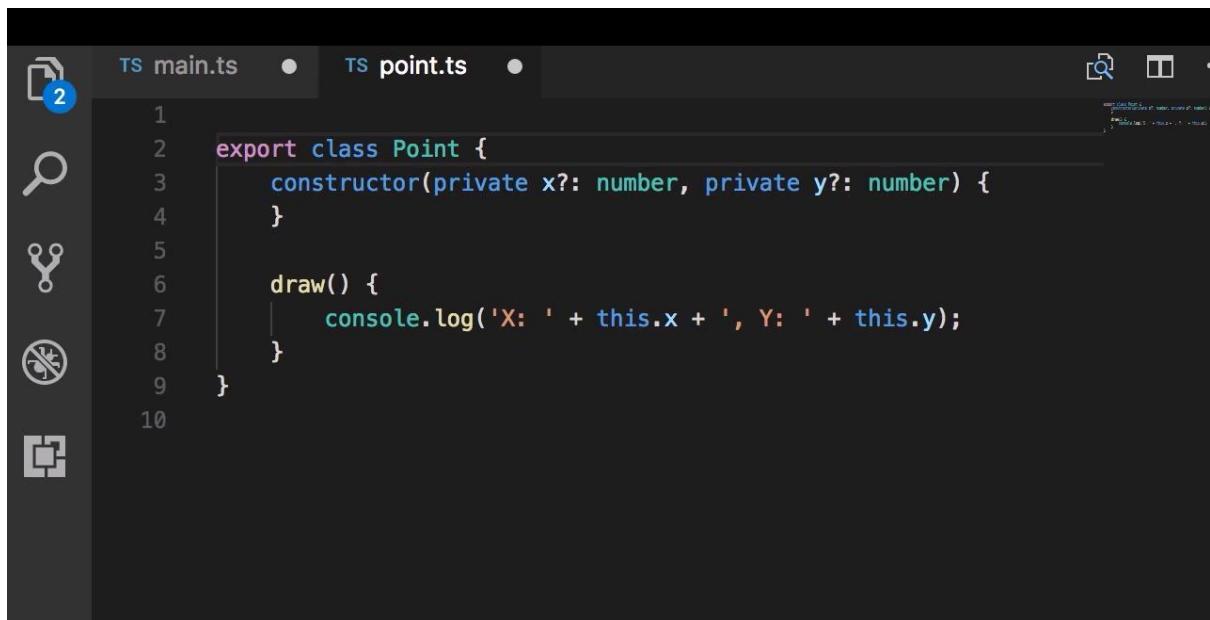
- OPEN EDITORS:
 - main.ts
- TS-HELLO:
 - point.ts
 - main.ts

The main editor area has two tabs open: "main.ts" and "point.ts". The "main.ts" tab contains the same code as the first screenshot, while the "point.ts" tab contains the following code:

```
1
2 class Point {
3     constructor(private x?: number
4         )
5
6     draw() {
7         console.log('X: ' + this.x
8     )
9 }
10
11 let point = new Point(1, 2);
12 point.draw();
```



```
1
2  class Point {
3      constructor(private x?: number, private y?: number) {
4          }
5
6      draw() {
7          console.log('X: ' + this.x + ', Y: ' + this.y);
8      }
9  }
10
```



```
1
2  export class Point {
3      constructor(private x?: number, private y?: number) {
4          }
5
6      draw() {
7          console.log('X: ' + this.x + ', Y: ' + this.y);
8      }
9  }
10
```

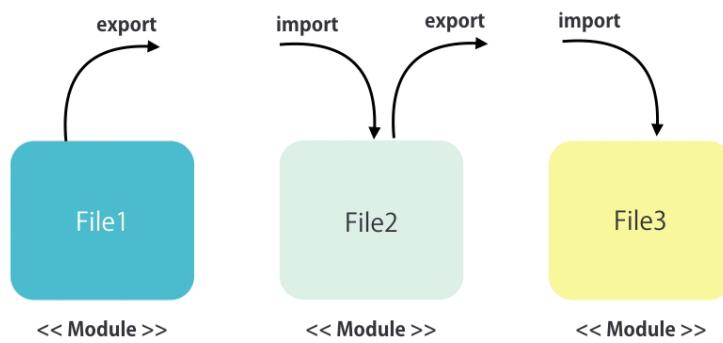
A screenshot of the Visual Studio Code interface. On the left is a dark sidebar with icons for file, search, and other tools. The main area shows two tabs: 'TS main.ts' and 'TS point.ts'. The 'main.ts' tab is active, displaying the following code:

```
1
2
3 let point = new Point(1, 2);
4 point.draw();
```

A tooltip is displayed over the word 'Point' in the third line, showing the error message: '[ts] Cannot find name 'Point''. Below the tooltip, the word 'any' is visible.

A screenshot of the Visual Studio Code interface, identical to the first one but with a fix applied. The 'main.ts' tab is active, and the code now includes an import statement:

```
1
2 import { Point } from './point';
3
4 let point = new Point(1, 2);
5 point.draw();
```



Types

Classes

Simple variables

Objects