Introduction to JavaScript, Json, TypeScript, TSX

Cours Esilv 2024

arnaud.nauwynck@gmail.com

Outline

JavaScript

JSON

TypeScript

TSX

JavaScript

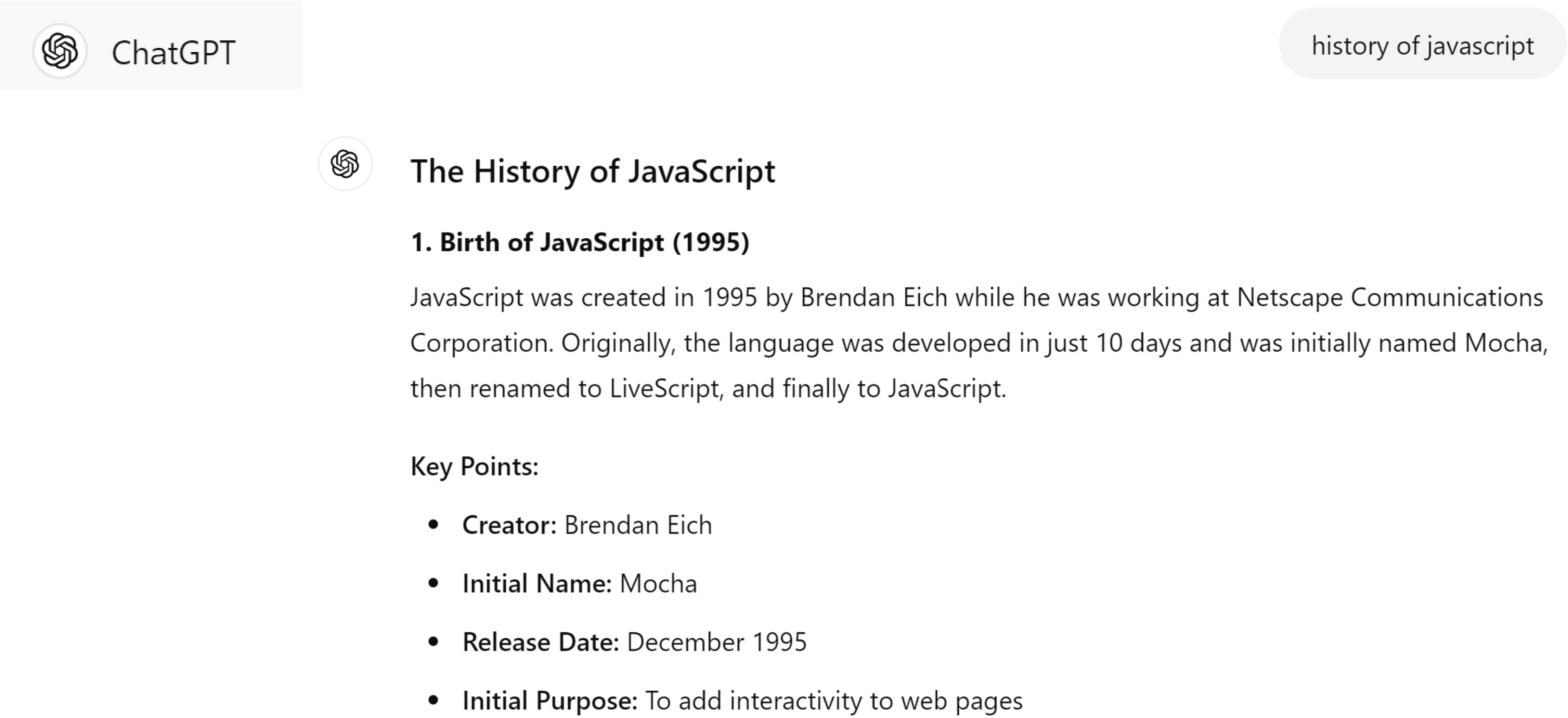
Java Script

Syntax inspired from "Java"
(itself inspired from "C", "C++")

untyped at compile-time,
interpreted at runtime
like "script" shells.

only "string", "numeric", boolean".
no "class" or type declarations !

Birth of JavaScript : 1995



history of javascript

The History of JavaScript

1. Birth of JavaScript (1995)

JavaScript was created in 1995 by Brendan Eich while he was working at Netscape Communications Corporation. Originally, the language was developed in just 10 days and was initially named Mocha, then renamed to LiveScript, and finally to JavaScript.

Key Points:

- **Creator:** Brendan Eich
- **Initial Name:** Mocha
- **Release Date:** December 1995
- **Initial Purpose:** To add interactivity to web pages

JavaScript Evolution

1995

Birth of JS

used in
Netscape
web browser

1997

Adopted by browsers,
but not standard

submitted to ECMA
international
for standardisation

spec : "**ECMAScript**"

2005

AJAX and Web 2.0

standardisation of
browsers

2009

ECMAScript 5
strict mode, JSON, ...

birth of many frameworks
(jQuery, AngularJS, etc..)

birth of **Node.JS**
server-side JS
development ecosystem
(npm, grunt, gulp, ..)

2012

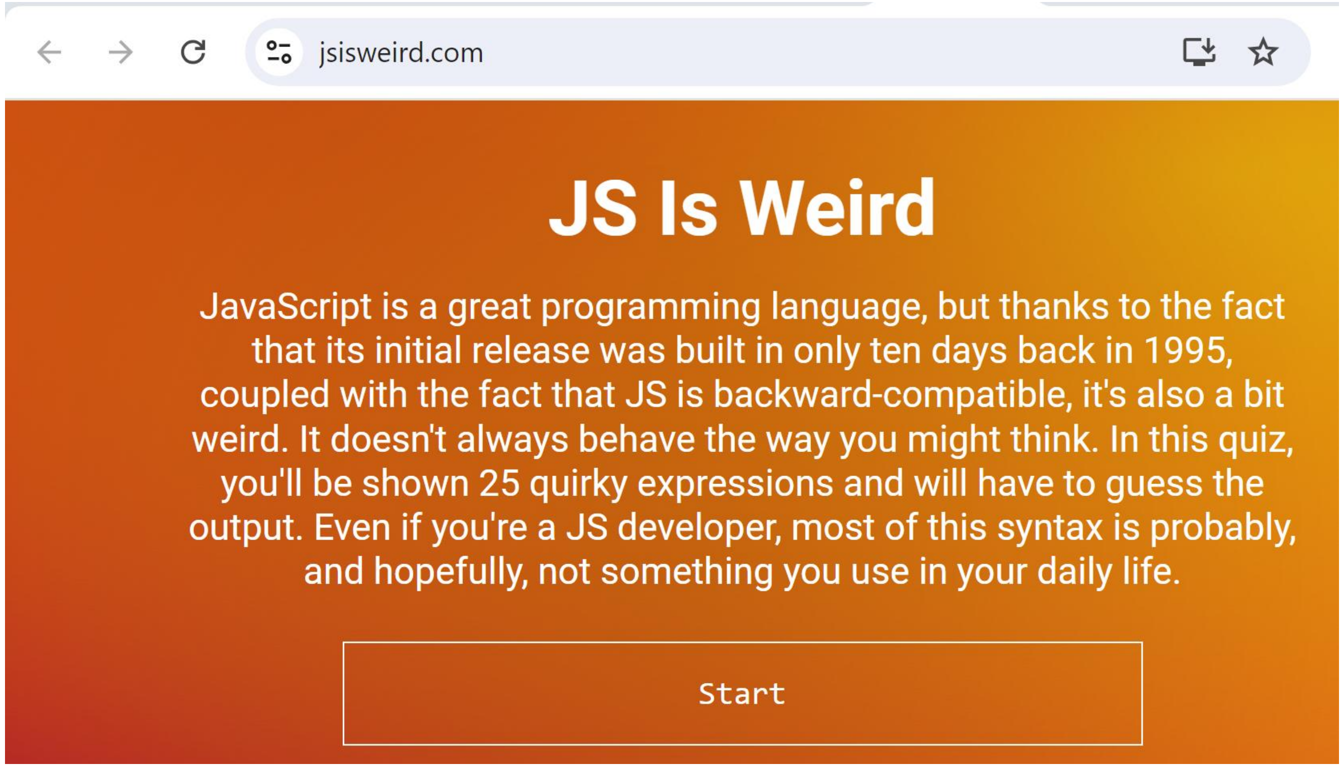
TypeScript

2015

ECMAScript 6 (=2015)
class, arrow, modules, ...

developed in 10 days by 1 person
... => explain many weird parts of the langage

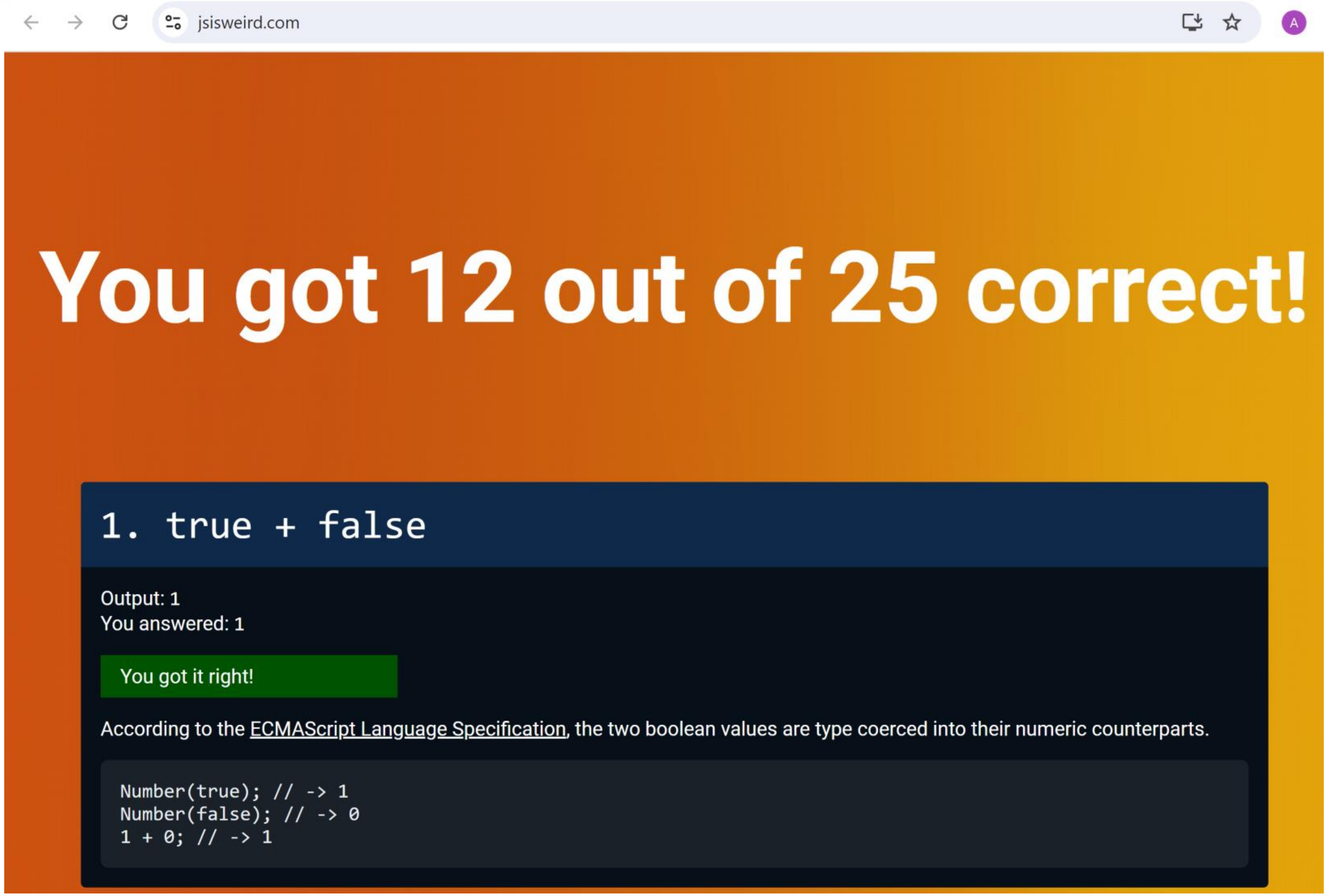
https://jsisweird.com/



JS Is Weird

JavaScript is a great programming language, but thanks to the fact that its initial release was built in only ten days back in 1995, coupled with the fact that JS is backward-compatible, it's also a bit weird. It doesn't always behave the way you might think. In this quiz, you'll be shown 25 quirky expressions and will have to guess the output. Even if you're a JS developer, most of this syntax is probably, and hopefully, not something you use in your daily life.

even for me



You got 12 out of 25 correct!

1. true + false

Output: 1
You answered: 1

You got it right!

According to the ECMAScript Language Specification, the two boolean values are type coerced into their numeric counterparts.

```
Number(true); // -> 1
Number(false); // -> 0
1 + 0; // -> 1
```

There are (strange) implicit coercions between primitive types...
For exact comparison, use "===" ("==" may introduce conversions)

JavaScript = un-maintanable code !!

NO Type !!

.. alternative like Google Dart since 2011 (ex in Flutter)

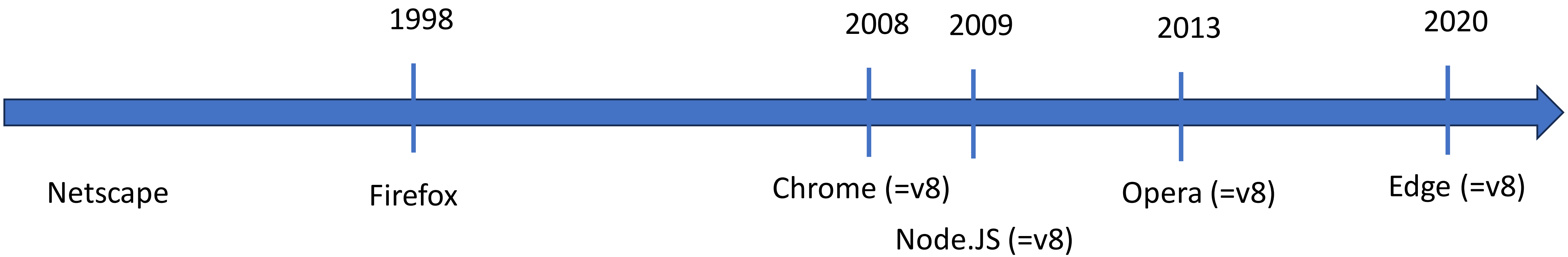
.. work-around since 2012 via TypeScript

NO Class !

.. work-around since 2012 via TypeScript

.. ok since 2015 EcmaScript6

Runtime Implementations



1998

2008

2009

2013

2020

Netscape

Firefox

Chrome (=v8)

Opera (=v8)

Edge (=v8)

Node.JS (=v8)

Mozilla
Open-Source
organisation
created from
Netscape Company

Firefox from
Nescape Navigator

Internet Explorer

IE had is own runtime.
his own non-standard specs
=> lot of behavior differences

jQuery was a hack for compatibility

Node.JS re-use the same runtime
as Chrome : "V8"

many browsers followed
(Opera in 2013, Edge in 2020, ...)

JavaScript Primitive Value Objects

null

undefined

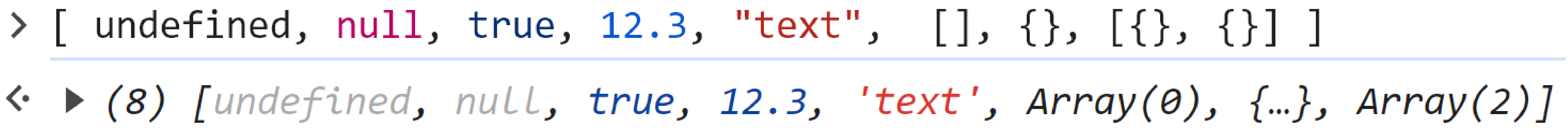
String

Numeric

Boolean

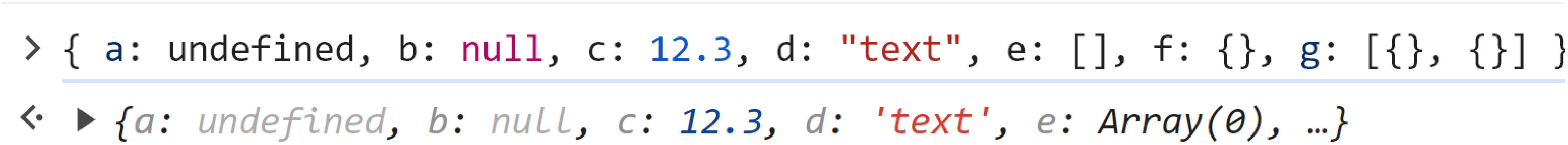
Object Compositions: [Array], {Object}

Array



```
> [ undefined, null, true, 12.3, "text",  [], {}, [{}, {}] ]
<· ▶ (8) [undefined, null, true, 12.3, 'text', Array(0), {…}, Array(2)]
```

Object



```
> { a: undefined, b: null, c: 12.3, d: "text", e: [], f: {}, g: [{}, {}] }
<· ▶ {a: undefined, b: null, c: 12.3, d: 'text', e: Array(0), …}
```

JSON = JavaScript Object Notation

NO code, only Value

Can be integrated directly in JavaScript (or TypeScript)

=> SUPER Easy for Http Rest using JSON as data encoding protocol

in pure JSON, identifier need to have double-quotes
... NOT needed in Js/Ts

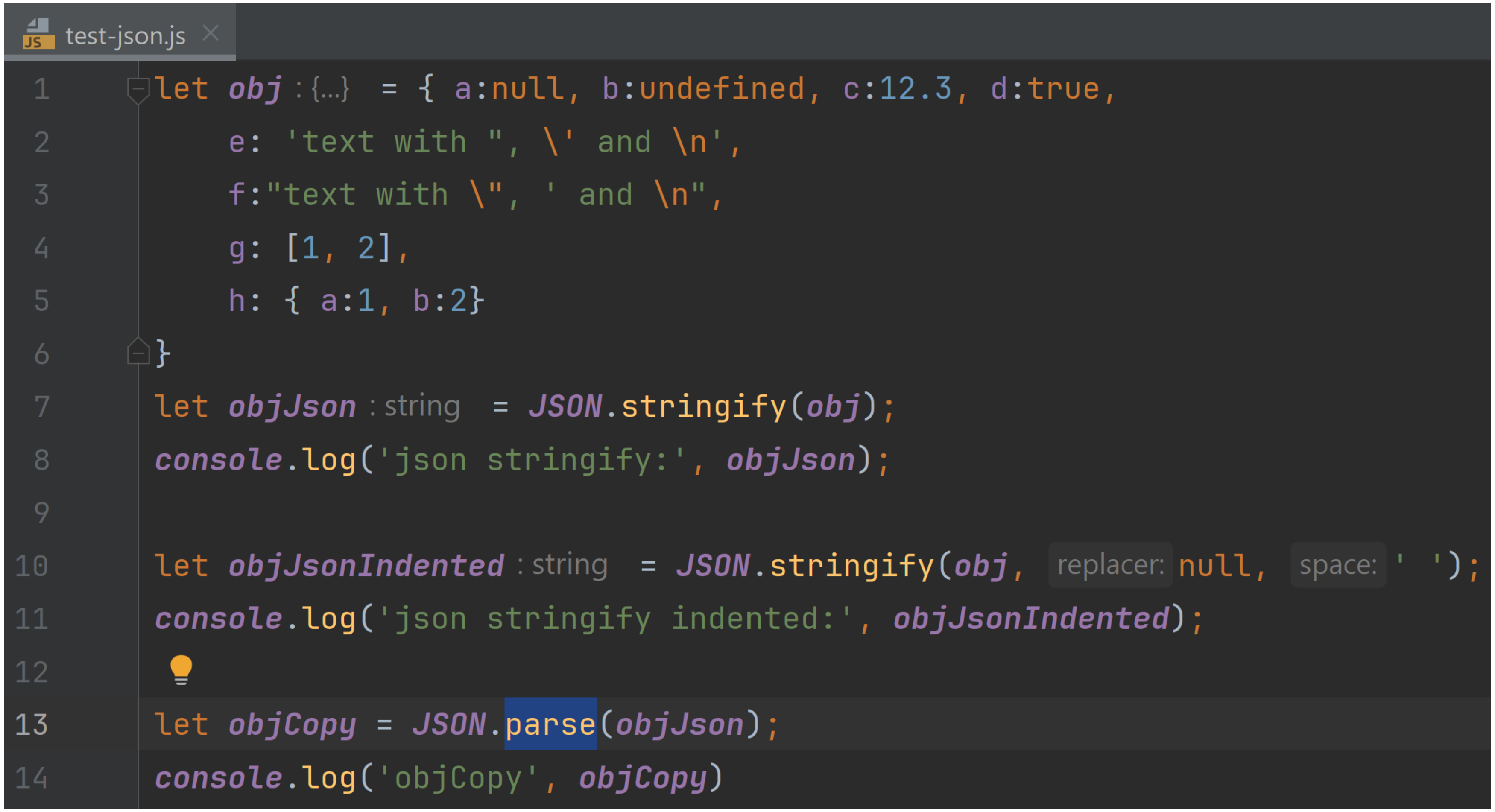
JSON Example

```
{
   "a": null,
   "b": undefined,
   "c": 123,
   "d":  true,
   "e":  "text with escaped \" and \n chars",
   "f":  [  null, undefined, 123, true, [], {} ],
   "g": {  "subField": 1, "b": true }
}
```

Example Unquoted JSON directly in JavaScript

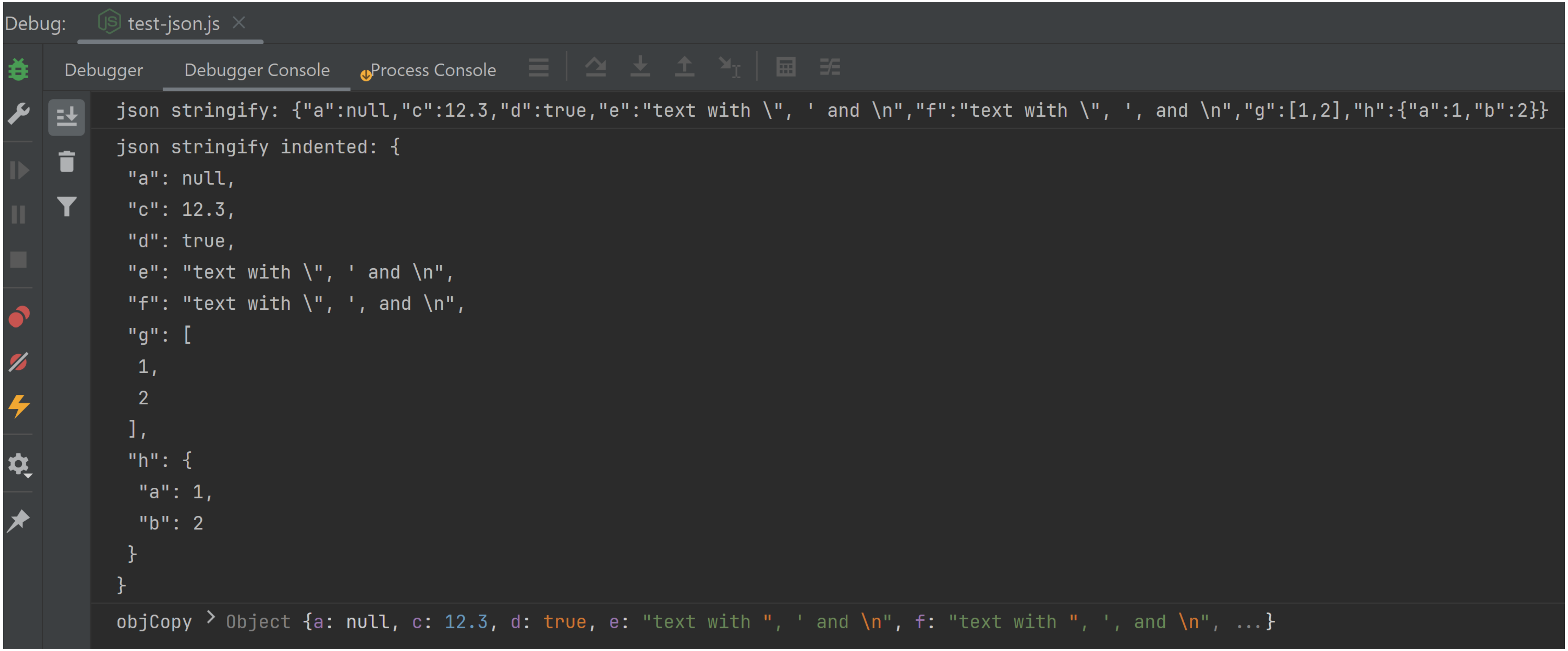
```
let obj = {
   a: null,
   b: undefined,
   c: 123,
   d:  true,
   e:  "text with escaped \" and \n chars",
   f:  [  null, undefined, 123, true, [], {} ],
   g: {  subField: 1, b: true }
};
```

JSON stringify() / parse()... builtin in JavaScript



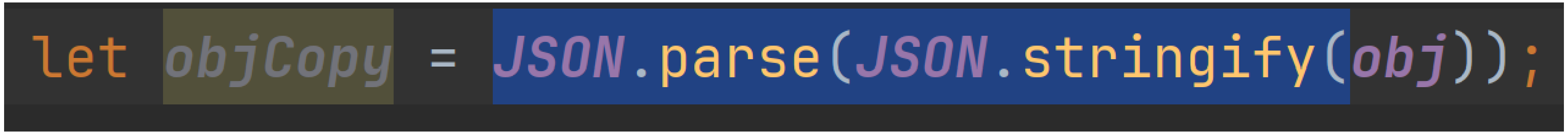
```
let obj : {...}  = { a:null, b:undefined, c:12.3, d:true,
    e: 'text with ", \' and \n',
    f:"text with \", ' and \n",
    g: [1, 2],
    h: { a:1, b:2}
}
let objJson : string  = JSON.stringify(obj);
console.log('json stringify:', objJson);

let objJsonIndented : string  = JSON.stringify(obj, replacer: null, space: ' ');
console.log('json stringify indented:', objJsonIndented);

let objCopy = JSON.parse(objJson);
console.log('objCopy', objCopy)
```

stringify, parse



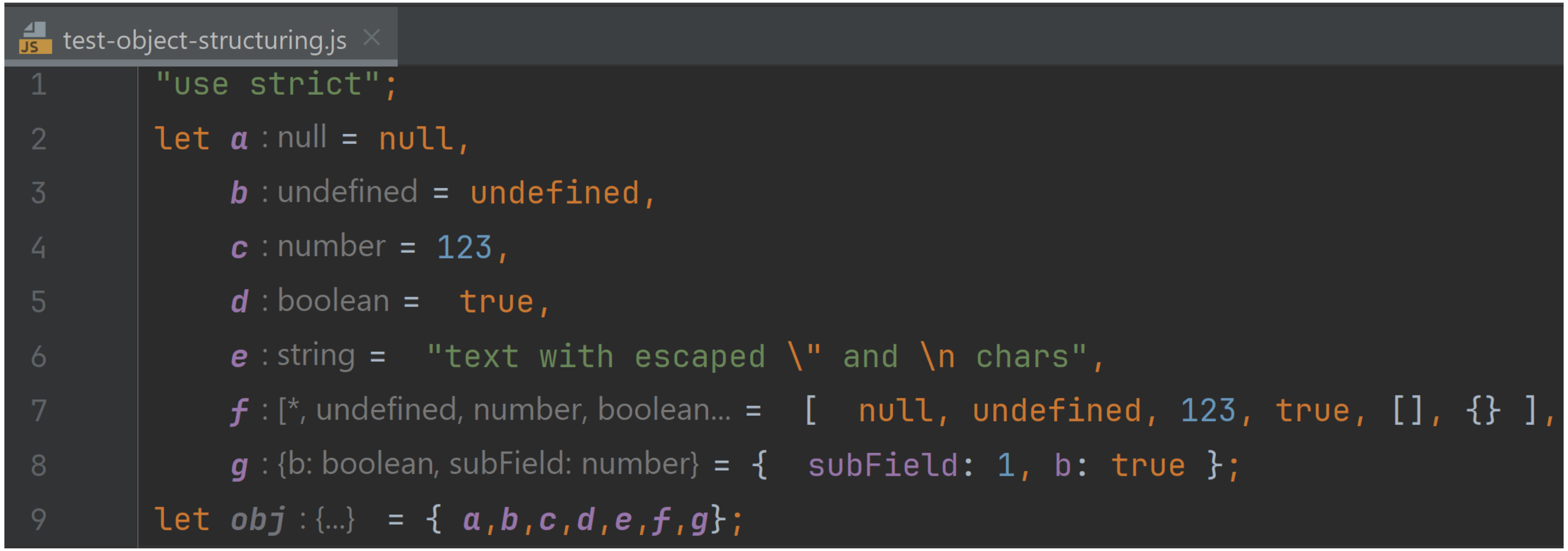
```
json stringify: {"a":null,"c":12.3,"d":true,"e":"text with \", ' and \n","f":"text with \", ', and \n","g":[1,2],"h":{"a":1,"b":2}}
json stringify indented: {
 "a": null,
 "c": 12.3,
 "d": true,
 "e": "text with \", ' and \n",
 "f": "text with \", ', and \n",
 "g": [
  1,
  2
 ],
 "h": {
  "a": 1,
  "b": 2
 }
}
objCopy > Object {a: null, c: 12.3, d: true, e: "text with ", ' and \n", f: "text with ", ', and \n", ...}
```

JavaScript object deep clone



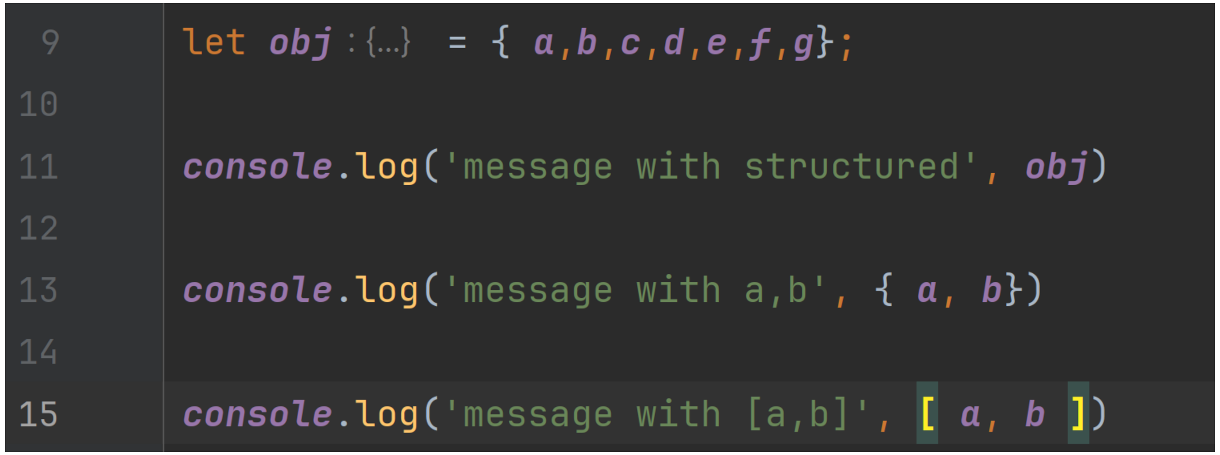
```javascript
let objCopy = JSON.parse(JSON.stringify(obj));
```

Object Creation from variables without reapeating "a":a, "b":b, simply {a,b}

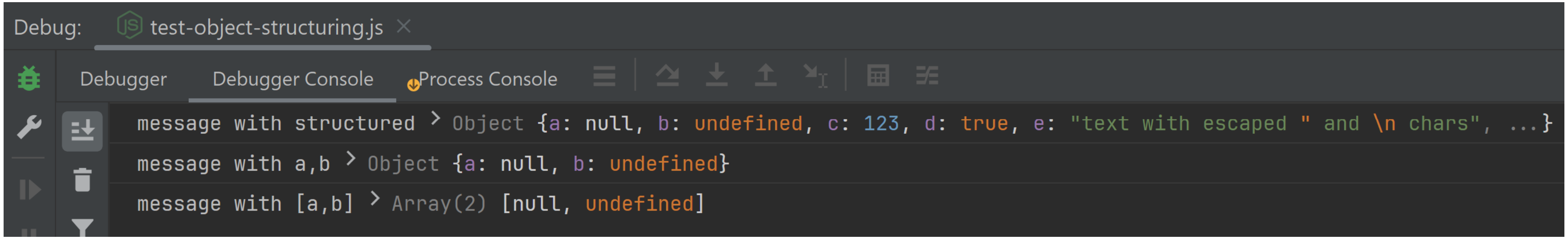


```
"use strict";
let a : null = null,
    b : undefined = undefined,
    c : number = 123,
    d : boolean =  true,
    e : string =  "text with escaped \" and \n chars",
    f : [*, undefined, number, boolean... =  [  null, undefined, 123, true, [], {} ],
    g : {b: boolean, subField: number} = {  subField: 1, b: true };
let obj : {...}  = { a,b,c,d,e,f,g};
```

Structured Logging

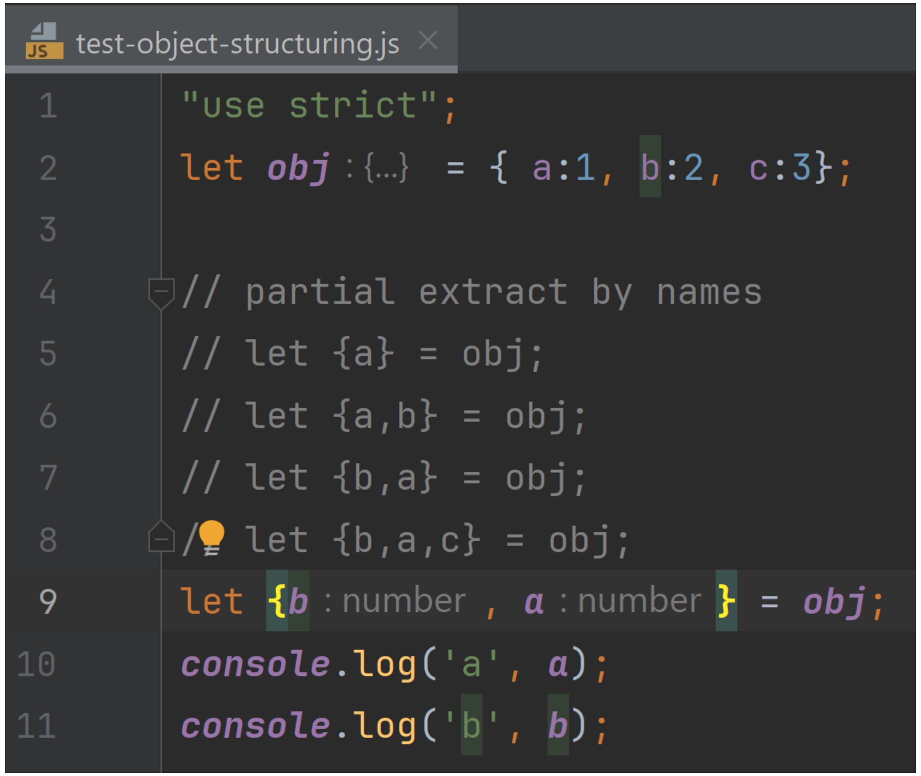


```
let obj : {…}  = { a,b,c,d,e,f,g};

console.log('message with structured', obj)

console.log('message with a,b', { a, b})

console.log('message with [a,b]', [ a, b ])
```

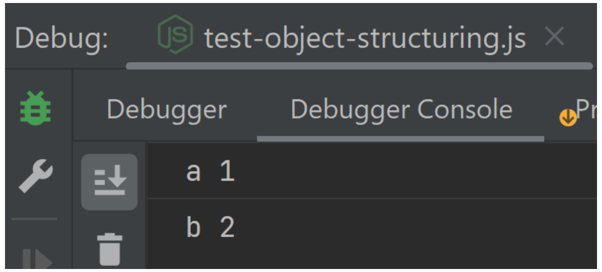


```
message with structured > Object {a: null, b: undefined, c: 123, d: true, e: "text with escaped " and \n chars", ...}
message with a,b > Object {a: null, b: undefined}
message with [a,b] > Array(2) [null, undefined]
```

Object De-Structuring to variables

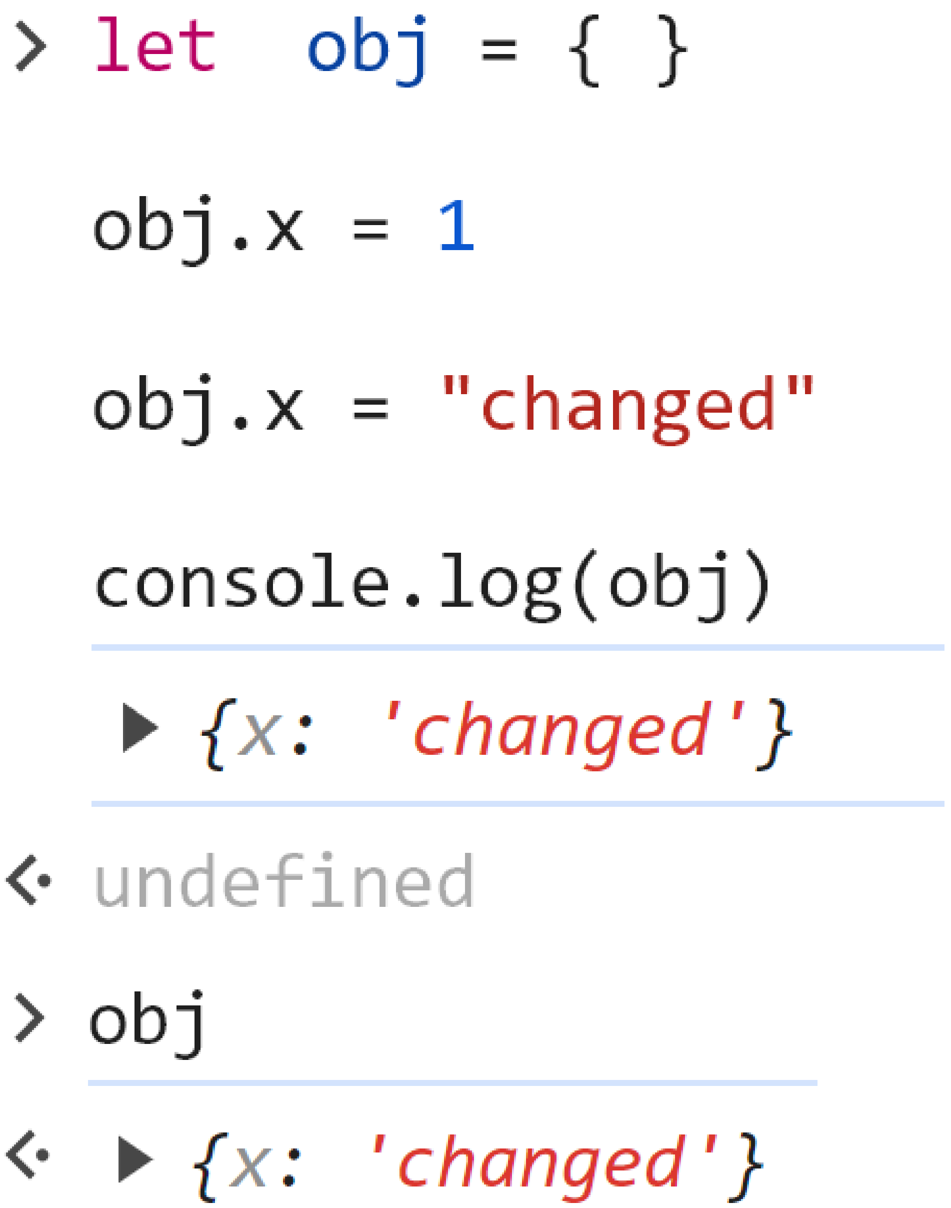


```
"use strict";
let obj : {…}  = { a:1, b:2, c:3};

// partial extract by names
// let {a} = obj;
// let {a,b} = obj;
// let {b,a} = obj;
// let {b,a,c} = obj;
let {b : number , a : number } = obj;
console.log('a', a);
console.log('b', b);
```



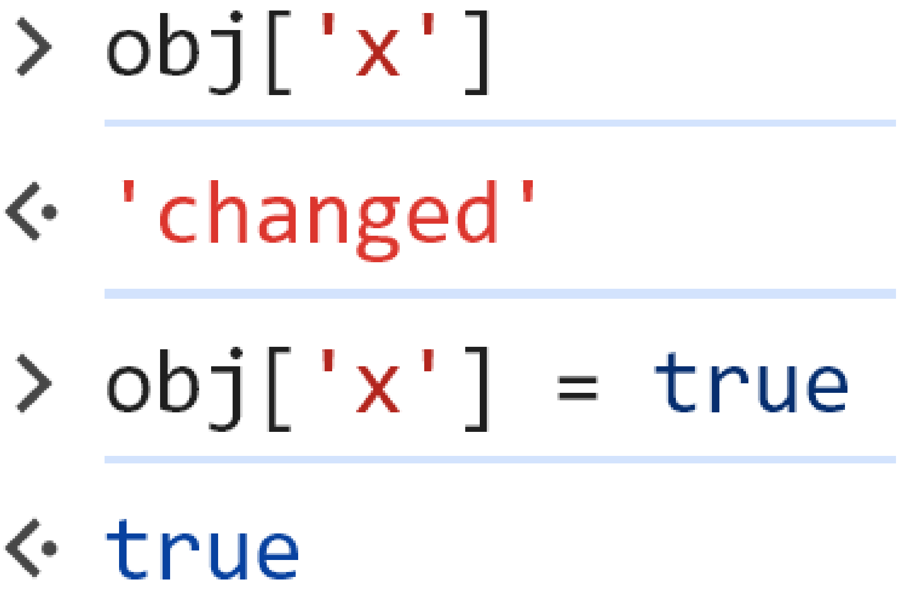
```
a 1
b 2
```

Objects (~= HashMap<String,*>)



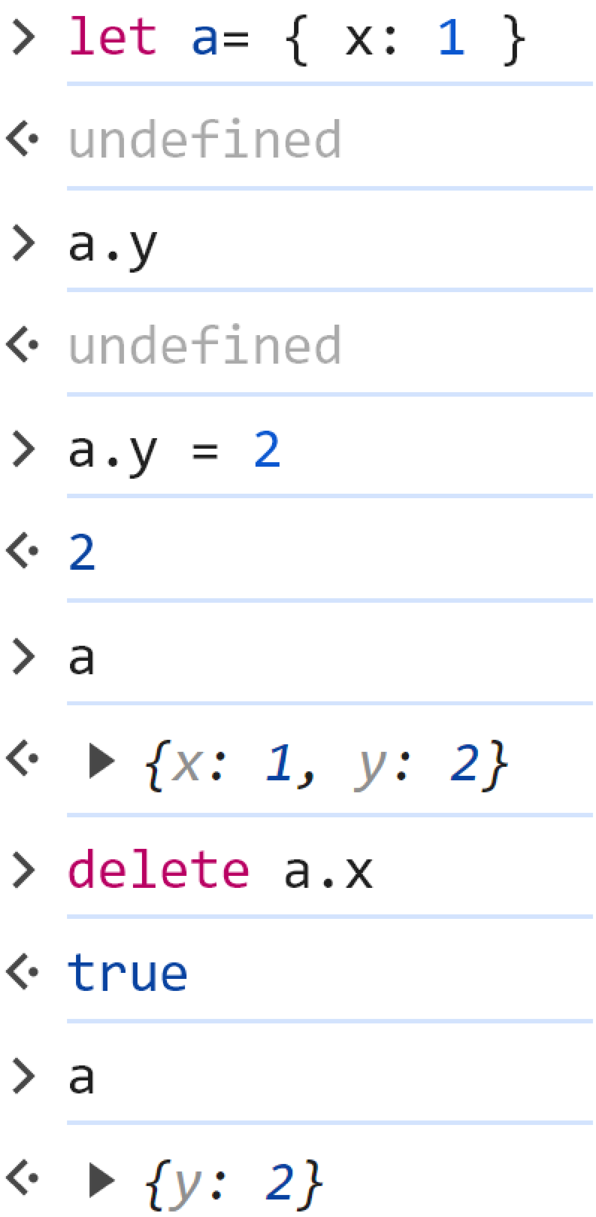
```
> let  obj = { }

  obj.x = 1

  obj.x = "changed"

  console.log(obj)
  ▶ {x: 'changed'}
<· undefined
> obj
<· ▶ {x: 'changed'}
```

using field Array notation :



```
> obj['x']
<· 'changed'
> obj['x'] = true
<· true
```

Undefined Field



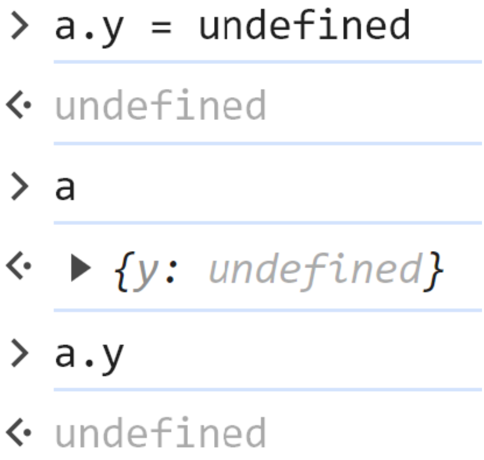
```
> let a= { x: 1 }
<· undefined
> a.y
<· undefined
> a.y = 2
<· 2
> a
<· ▶ {x: 1, y: 2}
> delete a.x
<· true
> a
<· ▶ {y: 2}
```

get unknown field
=> NO error, return undefined

add new field on first set

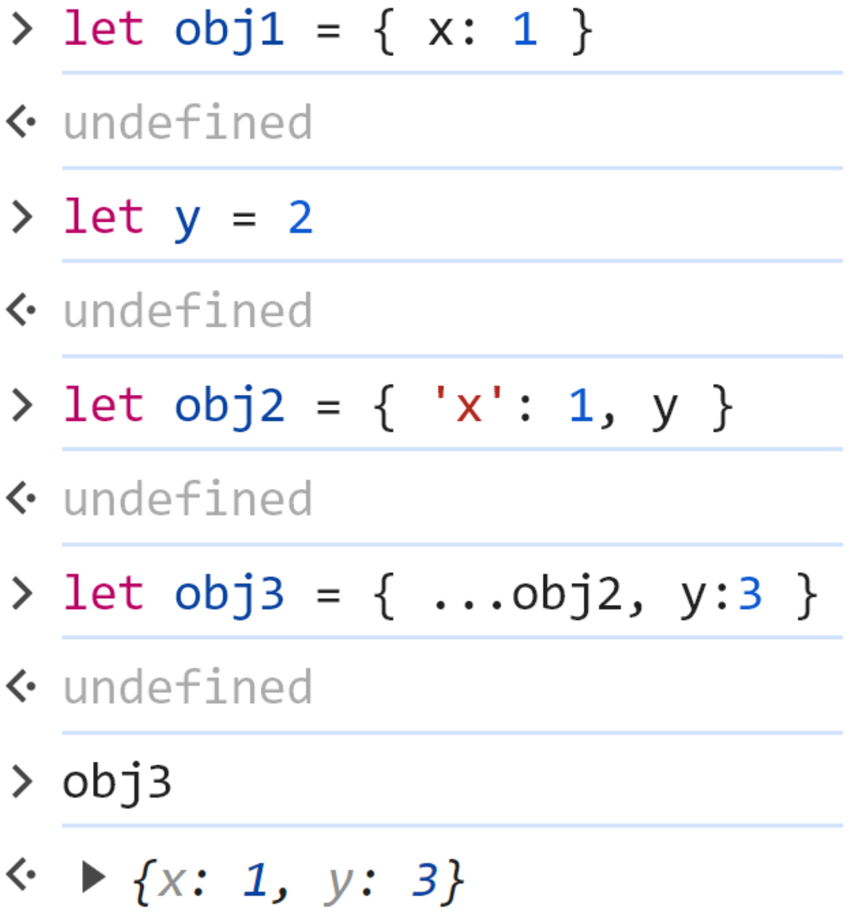
delete field

set field undefined



```
> a.y = undefined
<· undefined
> a
<· ▶ {y: undefined}
> a.y
<· undefined
```

Syntaxic Sugar for Object Creations

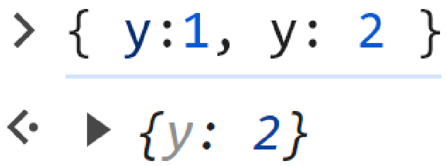


```
> let obj1 = { x: 1 }
<· undefined
> let y = 2
<· undefined
> let obj2 = { 'x': 1, y }
<· undefined
> let obj3 = { ...obj2, y:3 }
<· undefined
> obj3
<· ▶ {x: 1, y: 3}
```

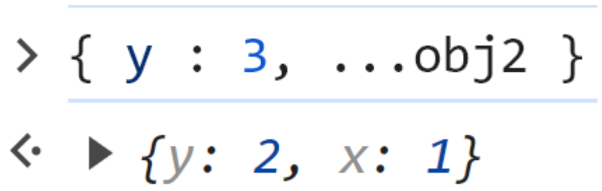
can ommit   quotes, double-quotes, and even variable name as in 'y': y

object spreading operator

(last value override previous)

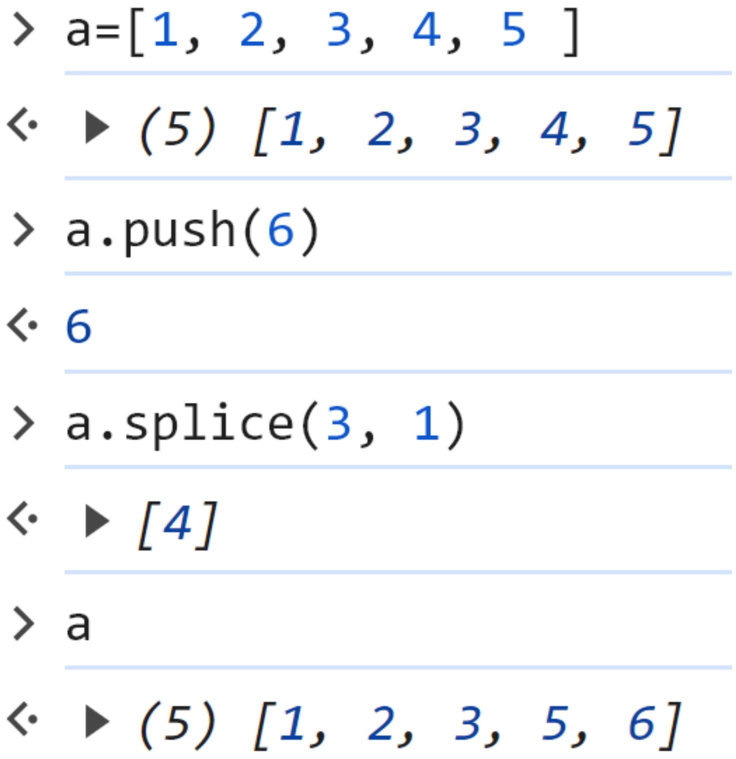


```
> { y:1, y: 2 }
<· ▶ {y: 2}
```



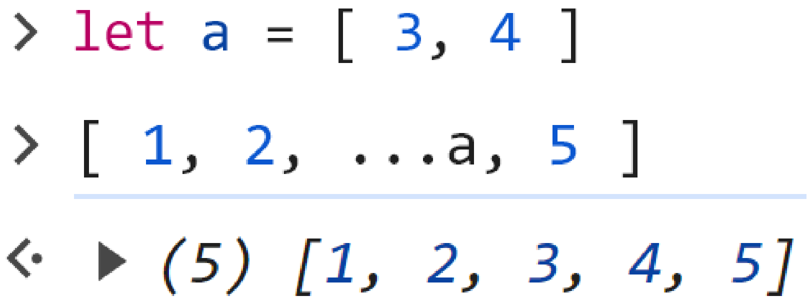
```
> { y : 3, ...obj2 }
<· ▶ {y: 2, x: 1}
```

Array



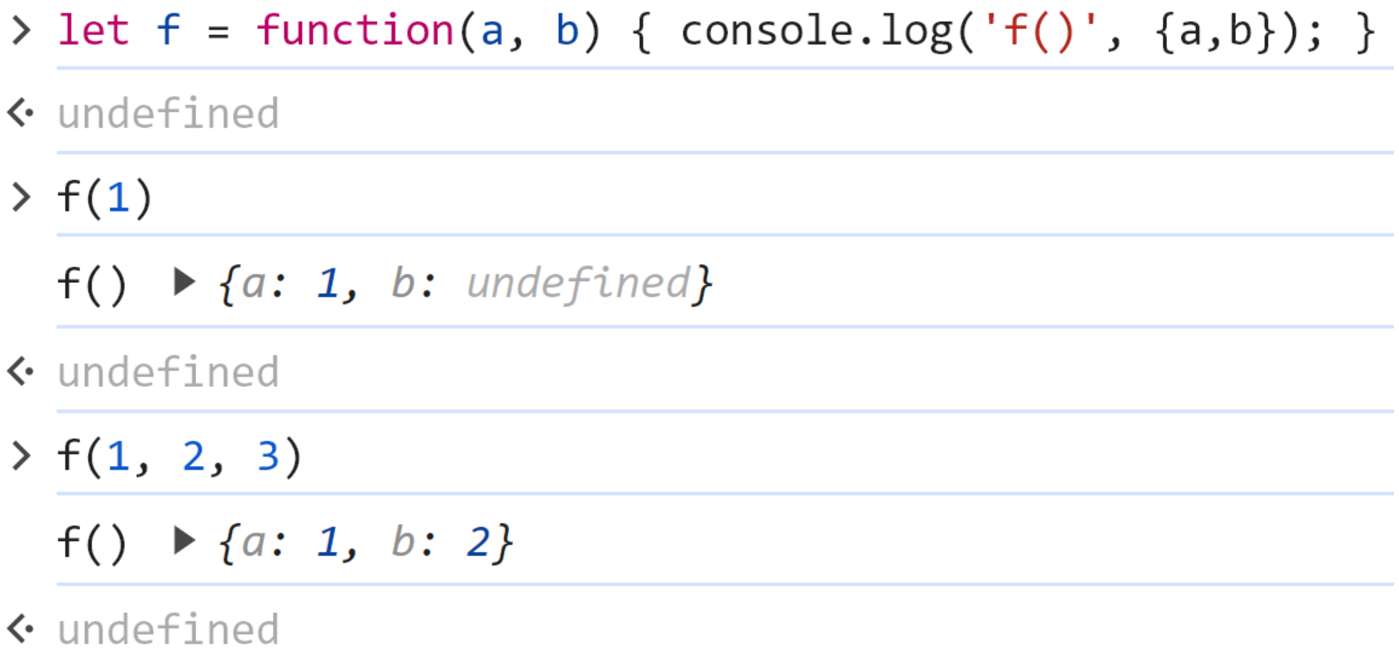
```
> a=[1, 2, 3, 4, 5 ]
<· ▶ (5) [1, 2, 3, 4, 5]
> a.push(6)
<· 6
> a.splice(3, 1)
<· ▶ [4]
> a
<· ▶ (5) [1, 2, 3, 5, 6]
```

spreading operator



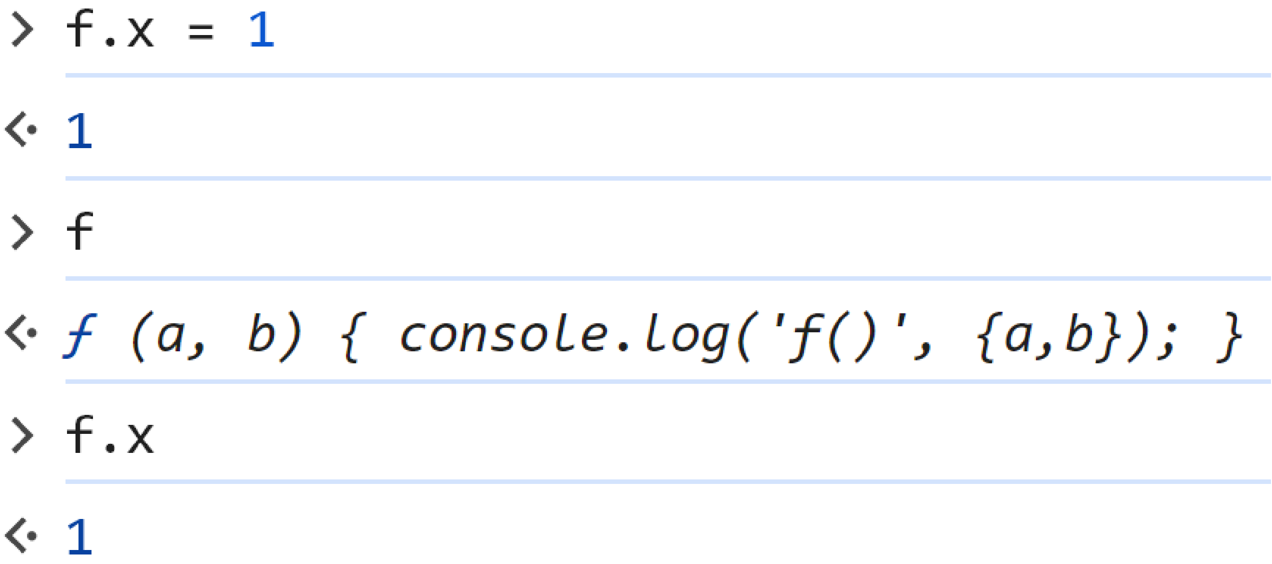
```
> let a = [ 3, 4 ]
> [ 1, 2, ...a, 5 ]
<· ▶ (5) [1, 2, 3, 4, 5]
```

Functions



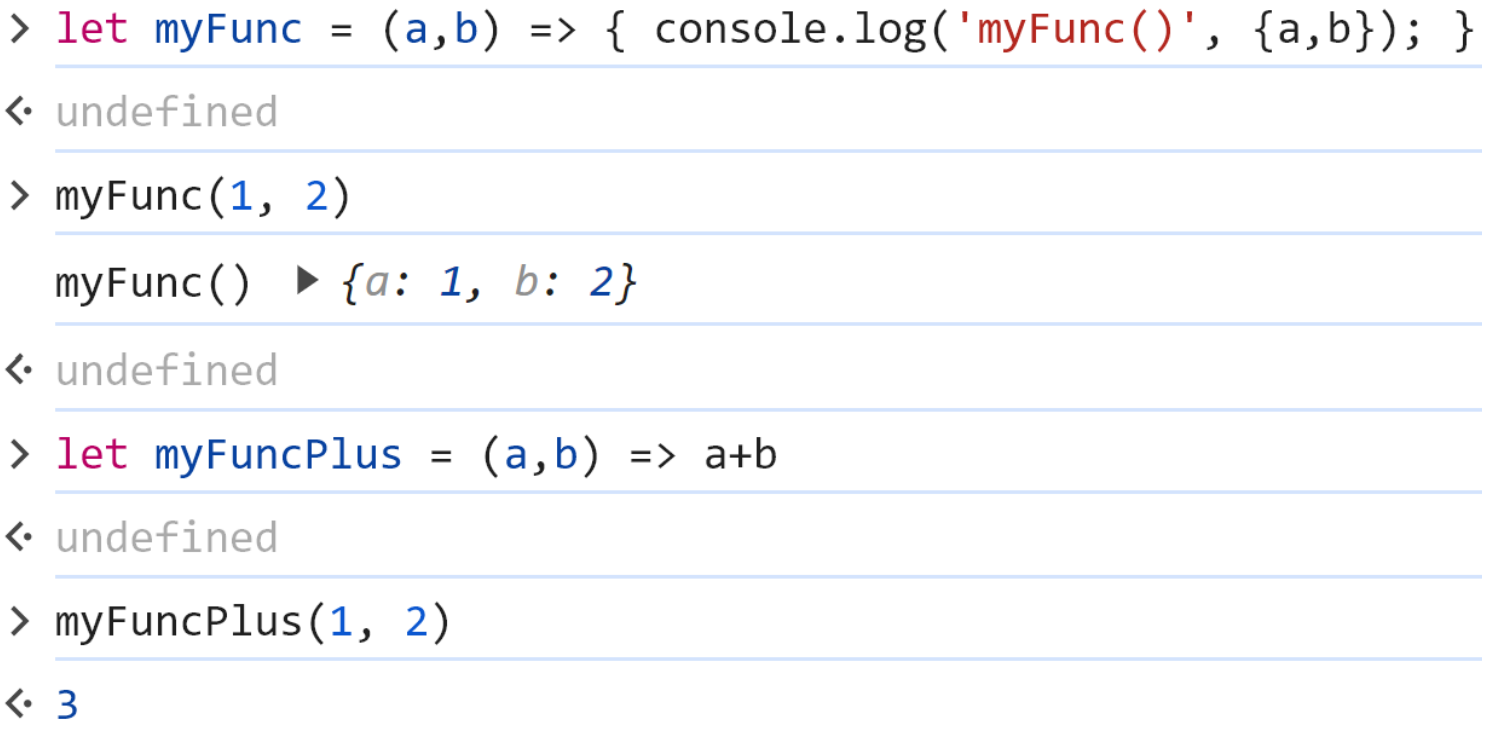
```
> let f = function(a, b) { console.log('f()', {a,b}); }
<· undefined
> f(1)
  f() ▶ {a: 1, b: undefined}
<· undefined
> f(1, 2, 3)
  f() ▶ {a: 1, b: 2}
<· undefined
```

Functions are Object



```
> f.x = 1
<· 1
> f
<· ƒ (a, b) { console.log('f()', {a,b}); }
> f.x
<· 1
```

Arrow (Lambda) Function

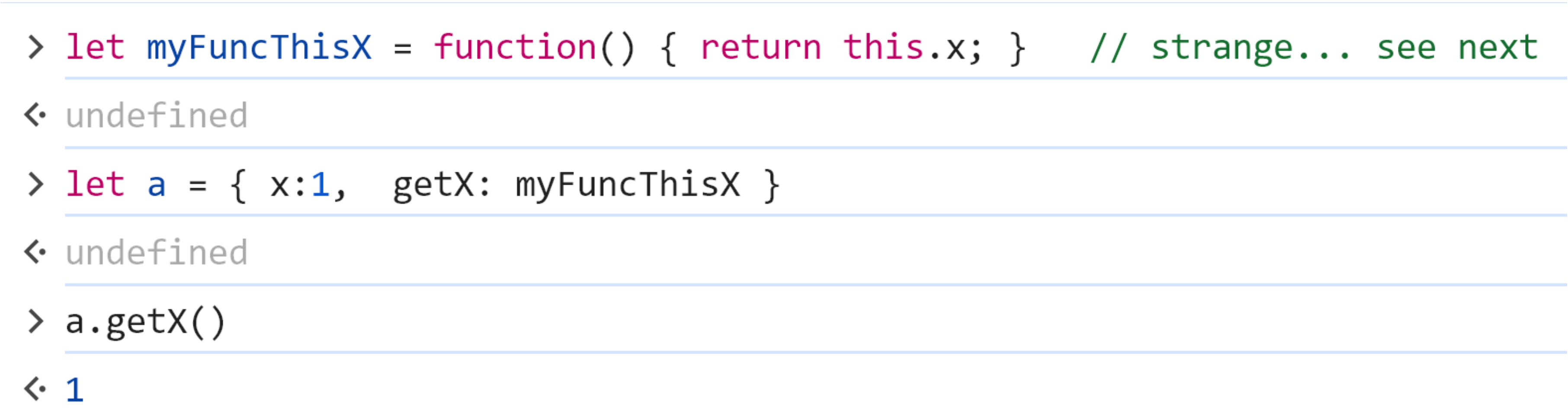


```
> let myFunc = (a,b) => { console.log('myFunc()', {a,b}); }
<· undefined
> myFunc(1, 2)
  myFunc()  ▶ {a: 1, b: 2}
<· undefined
> let myFuncPlus = (a,b) => a+b
<· undefined
> myFuncPlus(1, 2)
<· 3
```

arrow function with code block

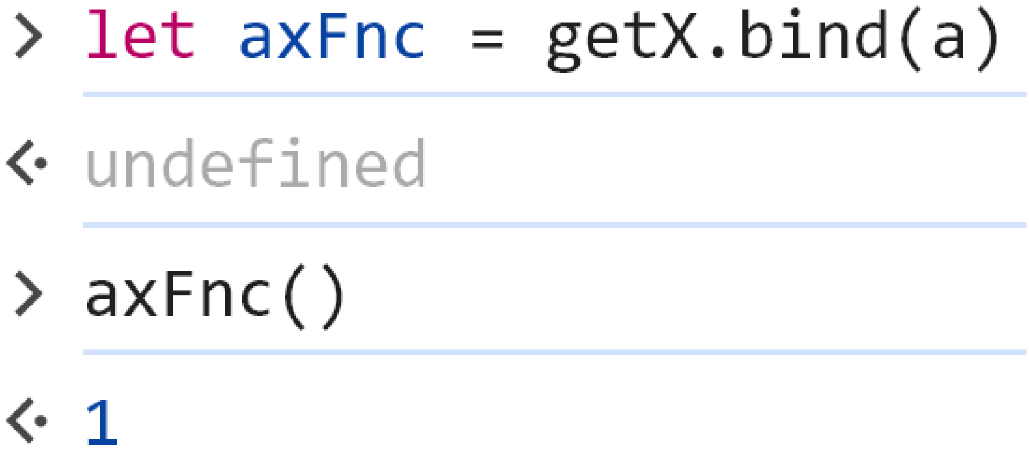
arrow function with expression

Function used as field ~ method ?
this = ??



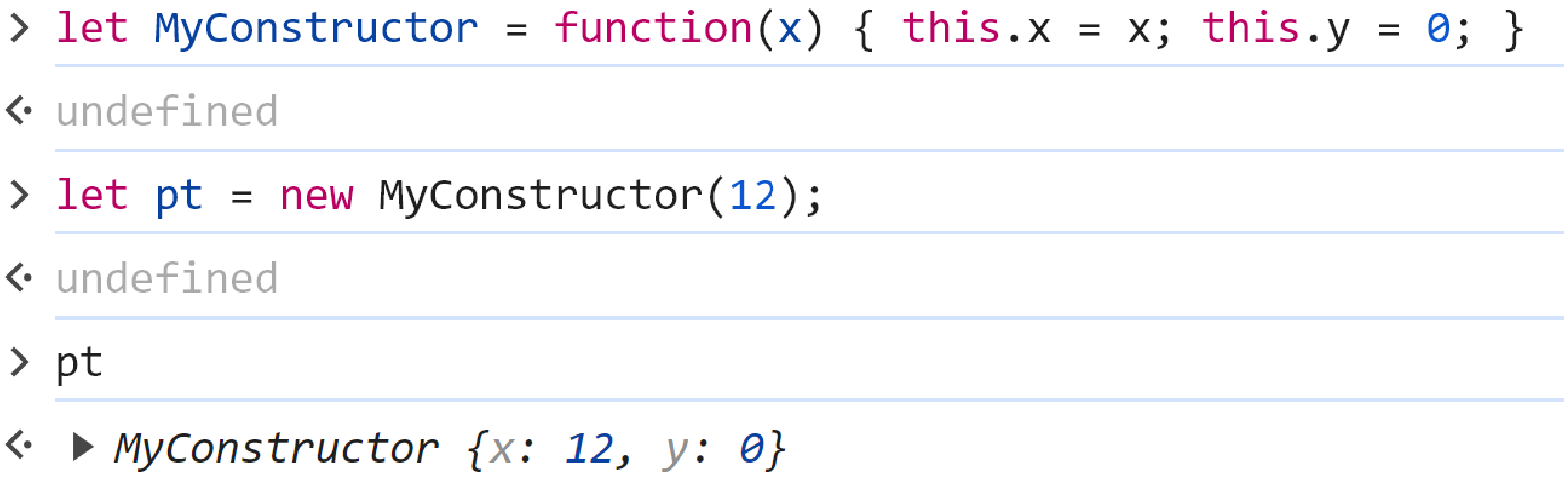
```
> let myFuncThisX = function() { return this.x; }   // strange... see next
<· undefined
> let a = { x:1,  getX: myFuncThisX }
<· undefined
> a.getX()
<· 1
```

"this" bounded to function



```
> let axFnc = getX.bind(a)
<· undefined
> axFnc()
<· 1
```

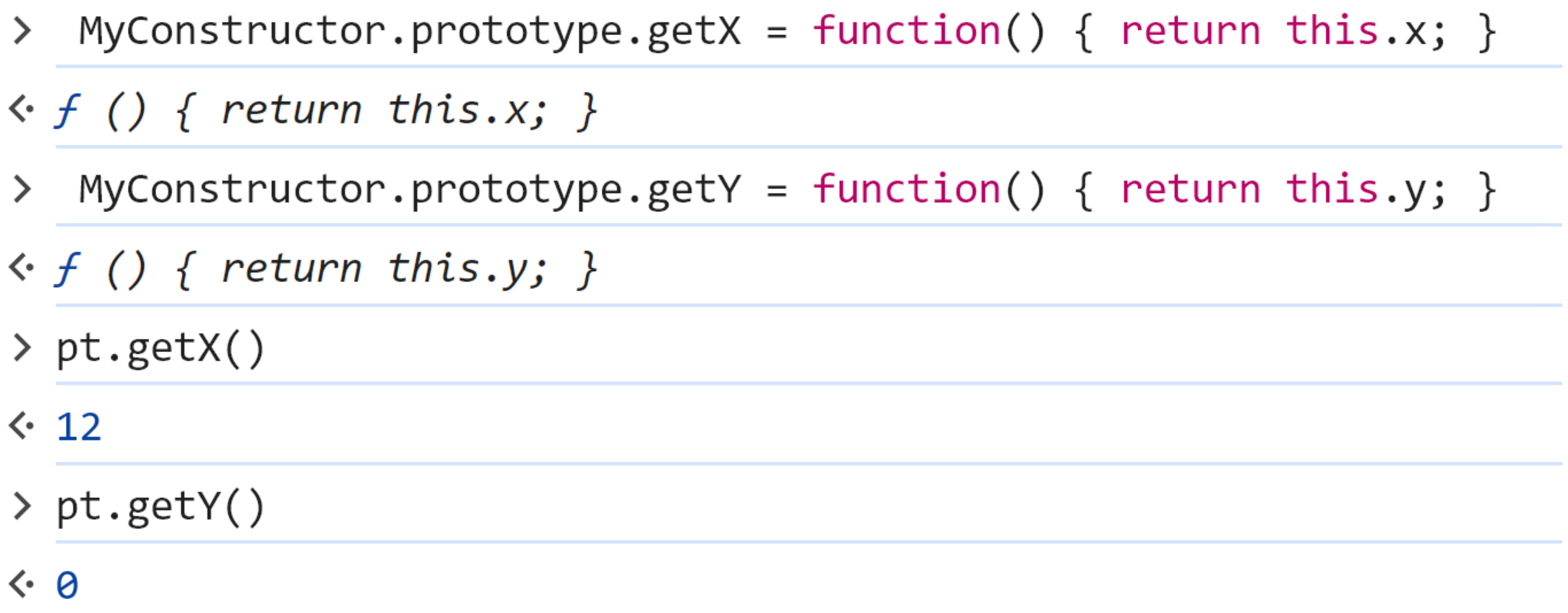
Function as "Class" Constructor



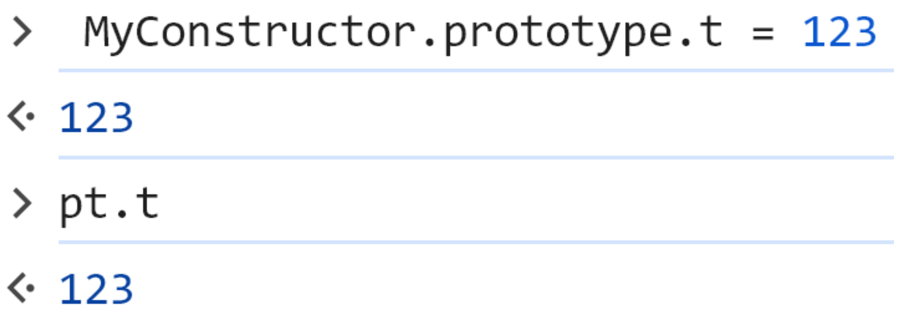
```
> let MyConstructor = function(x) { this.x = x; this.y = 0; }
<· undefined
> let pt = new MyConstructor(12);
<· undefined
> pt
<· ▶ MyConstructor {x: 12, y: 0}
```

prototype

adding field (property / method) to object prototype => adding to all impacted instances



```
>  MyConstructor.prototype.getX = function() { return this.x; }
<· ƒ () { return this.x; }
>  MyConstructor.prototype.getY = function() { return this.y; }
<· ƒ () { return this.y; }
> pt.getX()
<· 12
> pt.getY()
<· 0
```



```
>  MyConstructor.prototype.t = 123
<· 123
> pt.t
<· 123
```

emulated class with Prototype (before EcmaScript2015)

using Typescript (transpiler to JavaScript)

class are emulated with prototype before EcmaScript2015

looks like class after (but internally, still mutable prototype obects)

Type checking ?
at "Compile Time" ??

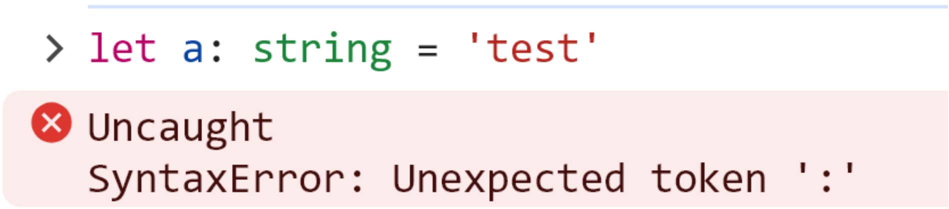
add optional " : <typeDeclaration>" to variables

TypeScript = superset of JavaScript langage

a JavaScript file is a TypeScript file
(valid grammar, but maybe invalid type check)

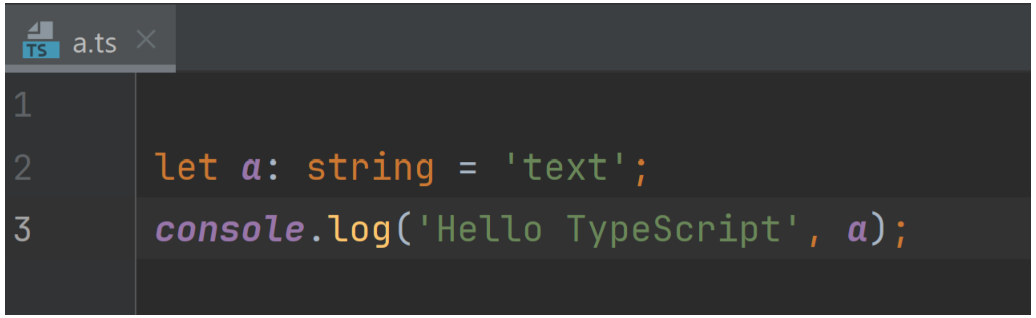
Edit Js to add ": <type>" to variables
Rename file ".js" to ".ts" TypeScript

": <type>"   is not valid JavaScript

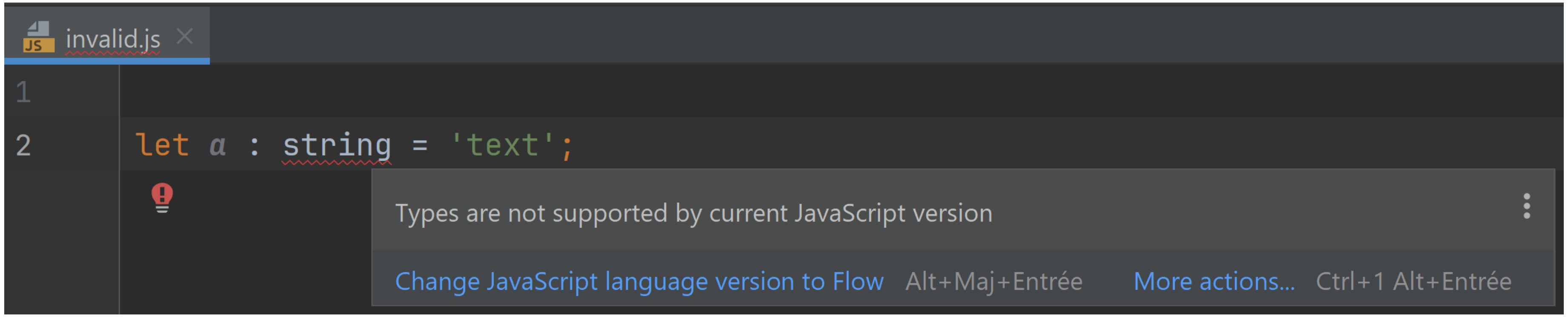


```
> let a: string = 'test'
✖ Uncaught
  SyntaxError: Unexpected token ':'
```

valid in TypeScript



```

let a: string = 'text';
console.log('Hello TypeScript', a);
```

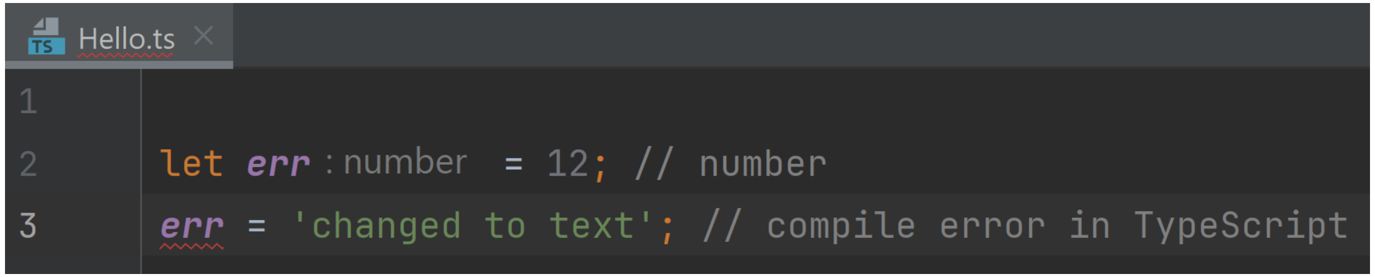


```

let a : string = 'text';
```

Types are not supported by current JavaScript version

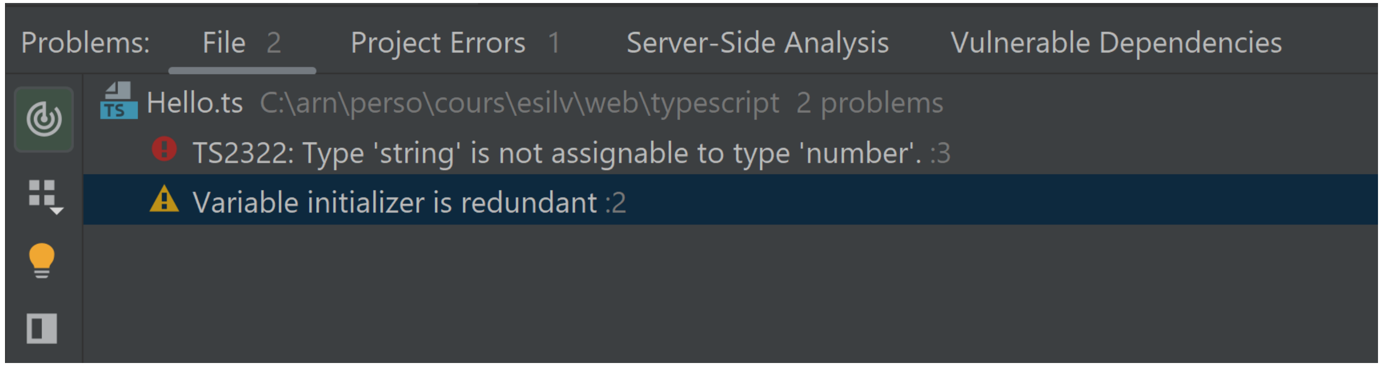
Change JavaScript language version to Flow Alt+Maj+Entrée
More actions... Ctrl+1 Alt+Entrée

Compile Time checking ...



```
let err :number  = 12; // number
err = 'changed to text'; // compile error in TypeScript
```

valid in untyped JavaScript
but not in TypeScript

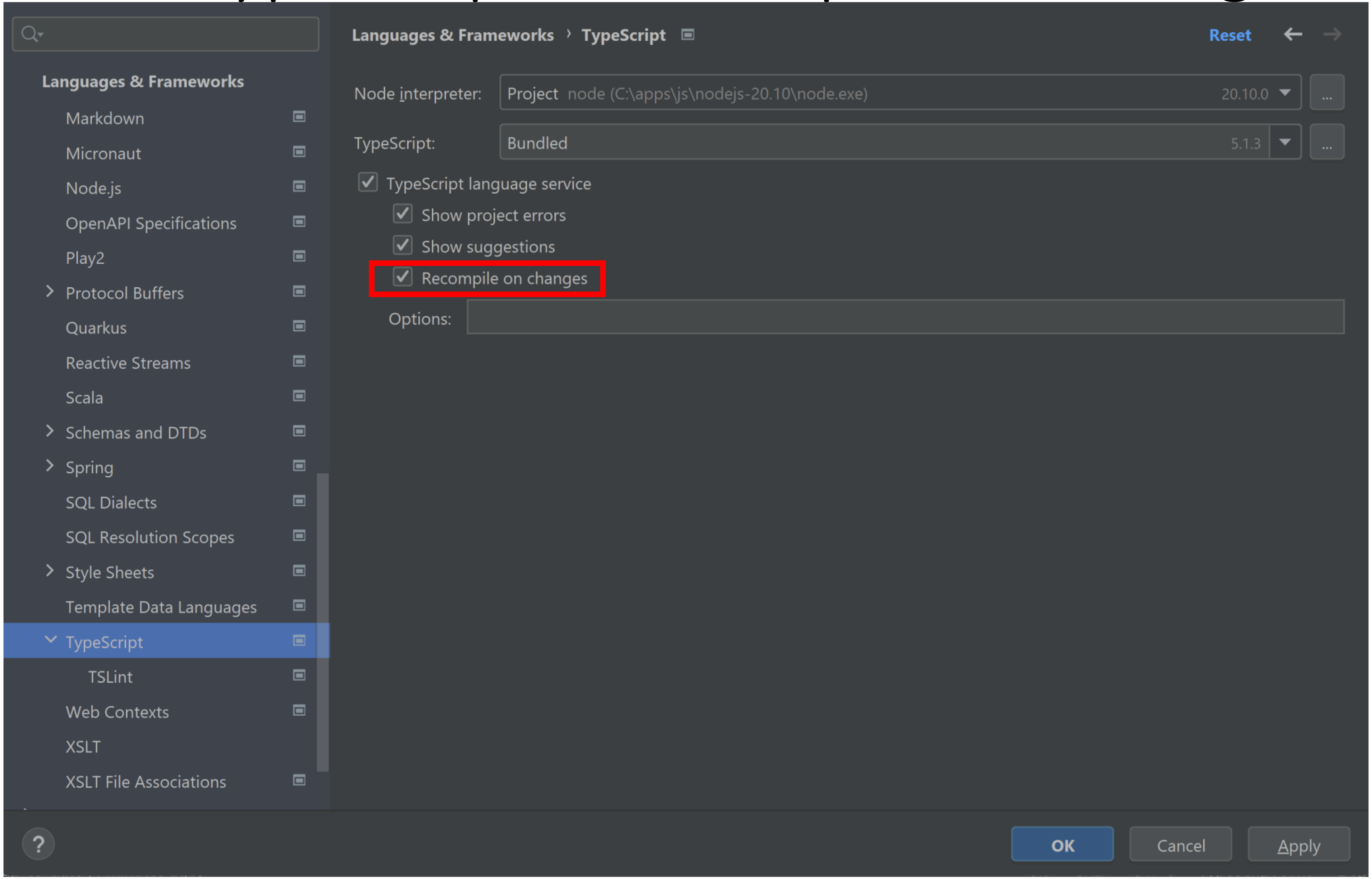


Hello.ts C:\arn\perso\cours\esilv\web\typescript 2 problems

TS2322: Type 'string' is not assignable to type 'number'. :3

Variable initializer is redundant :2

IntelliJ TypeScript : Recompile on changes



Languages & Frameworks › TypeScript

Node interpreter: Project node (C:\apps\js\nodejs-20.10\node.exe) 20.10.0

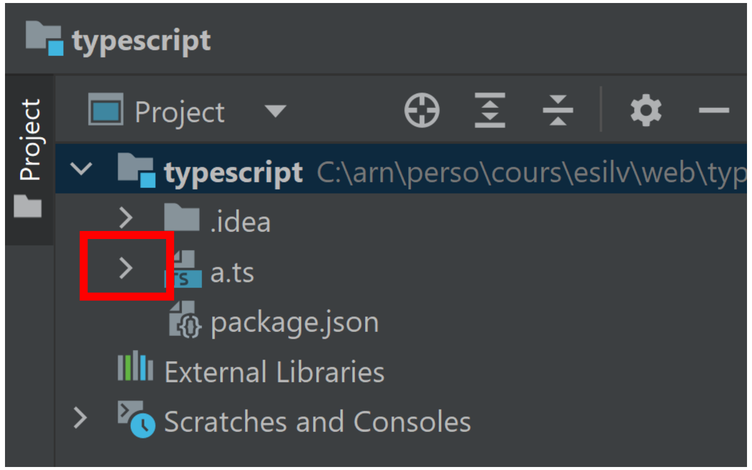
TypeScript: Bundled 5.1.3

☑ TypeScript language service

- ☑ Show project errors
- ☑ Show suggestions
- ☑ Recompile on changes

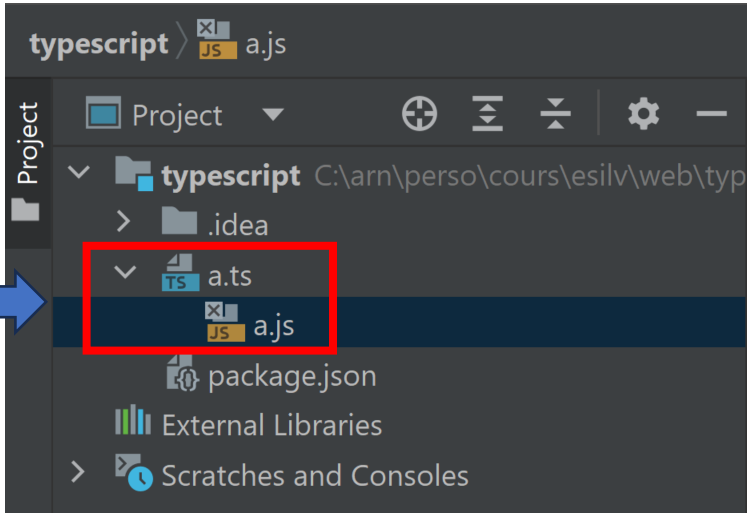
Options:

Js file generated from Ts file


typescript

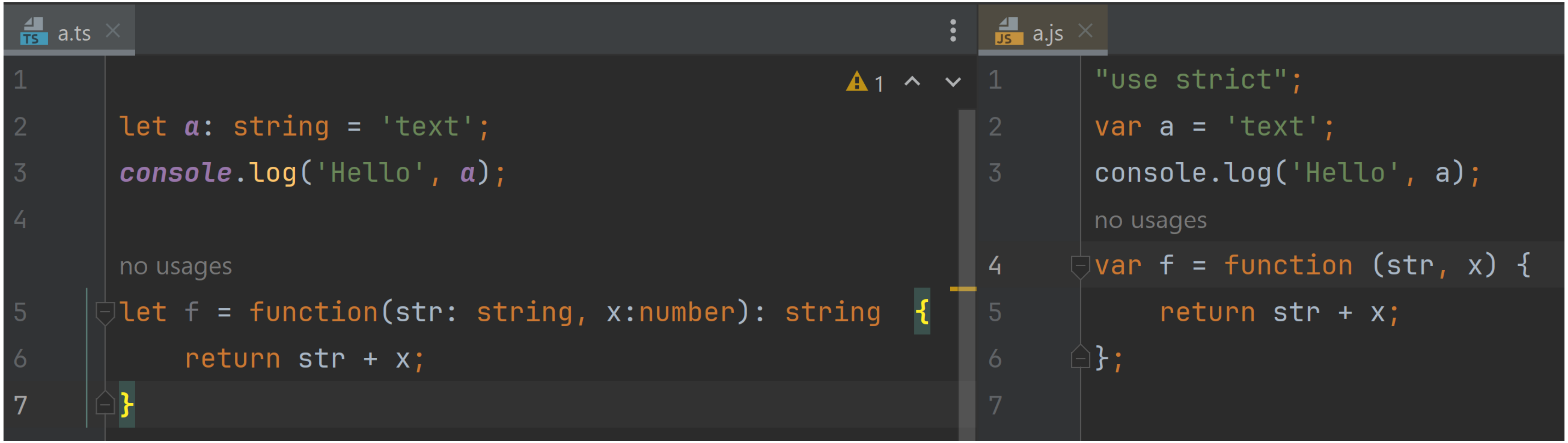
typescript C:\arn\perso\cours\esilv\web\typ
.idea
a.ts
package.json
External Libraries
Scratches and Consoles

expand to see generated file


typescript 〉 a.js

typescript C:\arn\perso\cours\esilv\web\typ
.idea
a.ts
a.js
package.json
External Libraries
Scratches and Consoles

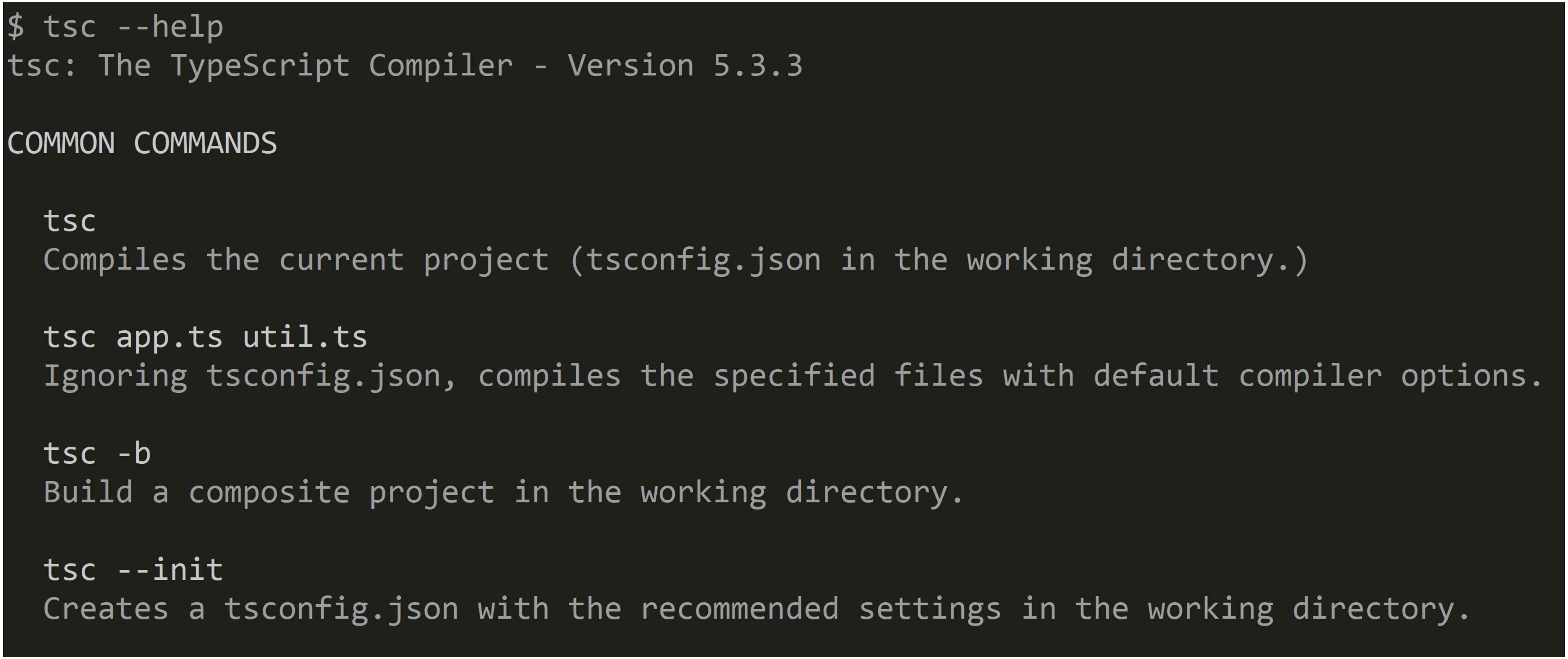
Transpile TypeScript = generate JS by mostly removing ": <typeDeclaration>"



```
let a: string = 'text';
console.log('Hello', a);

let f = function(str: string, x:number): string {
    return str + x;
}
```

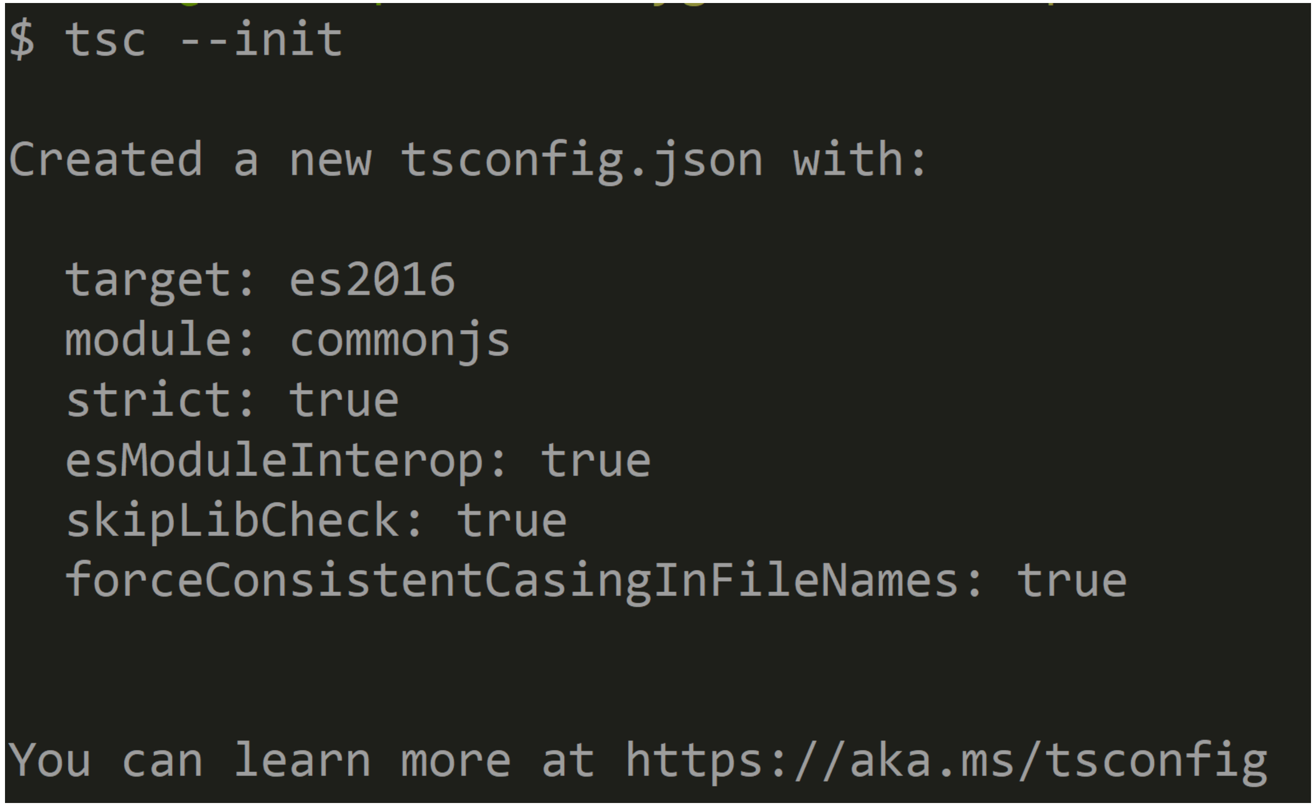
```
"use strict";
var a = 'text';
console.log('Hello', a);
var f = function (str, x) {
    return str + x;
};
```

Tsc (=TypeScript Compiler)



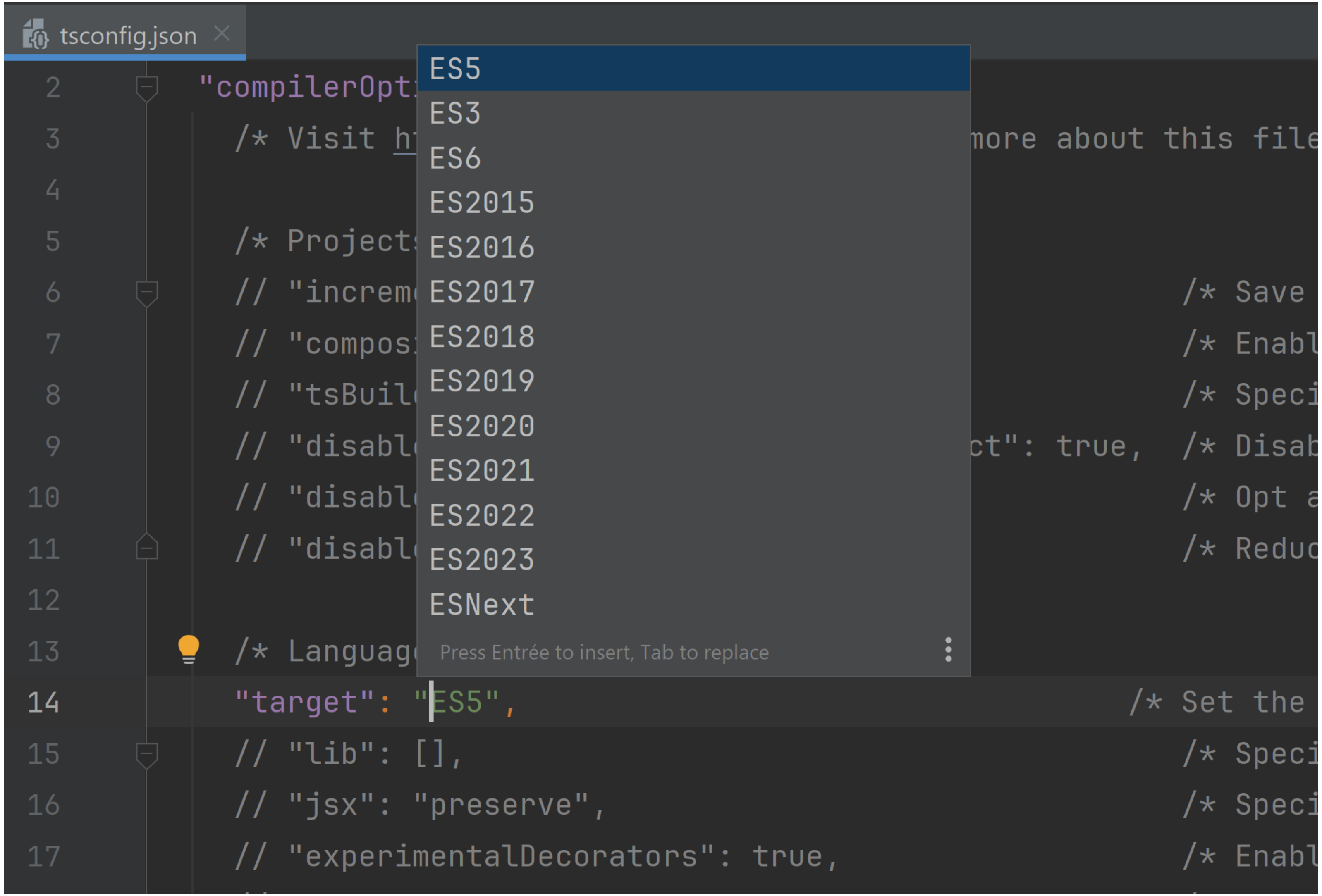
```
$ tsc --help
tsc: The TypeScript Compiler - Version 5.3.3

COMMON COMMANDS

  tsc
  Compiles the current project (tsconfig.json in the working directory.)

  tsc app.ts util.ts
  Ignoring tsconfig.json, compiles the specified files with default compiler options.

  tsc -b
  Build a composite project in the working directory.

  tsc --init
  Creates a tsconfig.json with the recommended settings in the working directory.
```

tsc --init



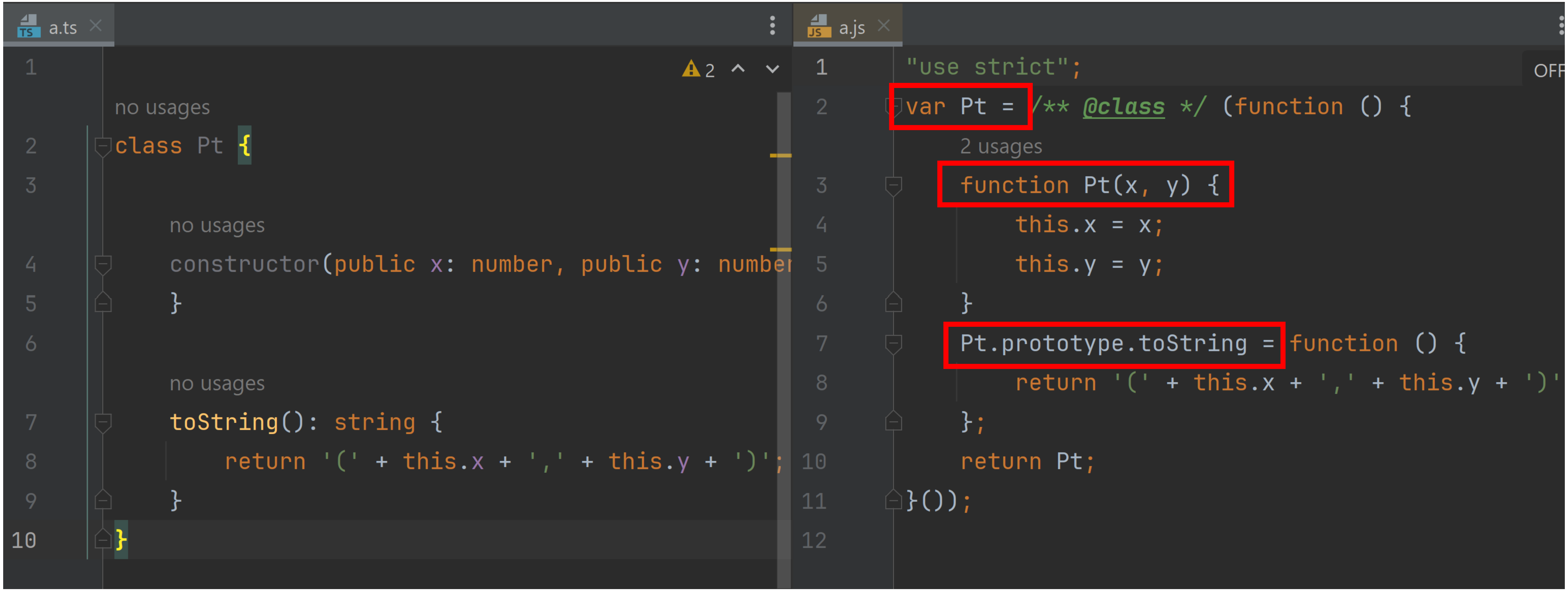
```
$ tsc --init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true


You can learn more at https://aka.ms/tsconfig
```

tsconfig.json using ES5 ... ES2023



```
  "compilerOpt
    /* Visit h                                                 more about this file

    /* Project
    // "increm                                                                 /* Save
    // "compos                                                                 /* Enabl
    // "tsBuil                                                                 /* Speci
    // "disabl                                                 ct": true,      /* Disab
    // "disabl                                                                 /* Opt a
    // "disabl                                                                 /* Reduc

    /* Languag
    "target": "ES5",                                                        /* Set the
    // "lib": [],                                                              /* Speci
    // "jsx": "preserve",                                                      /* Speci
    // "experimentalDecorators": true,                                         /* Enabl
```

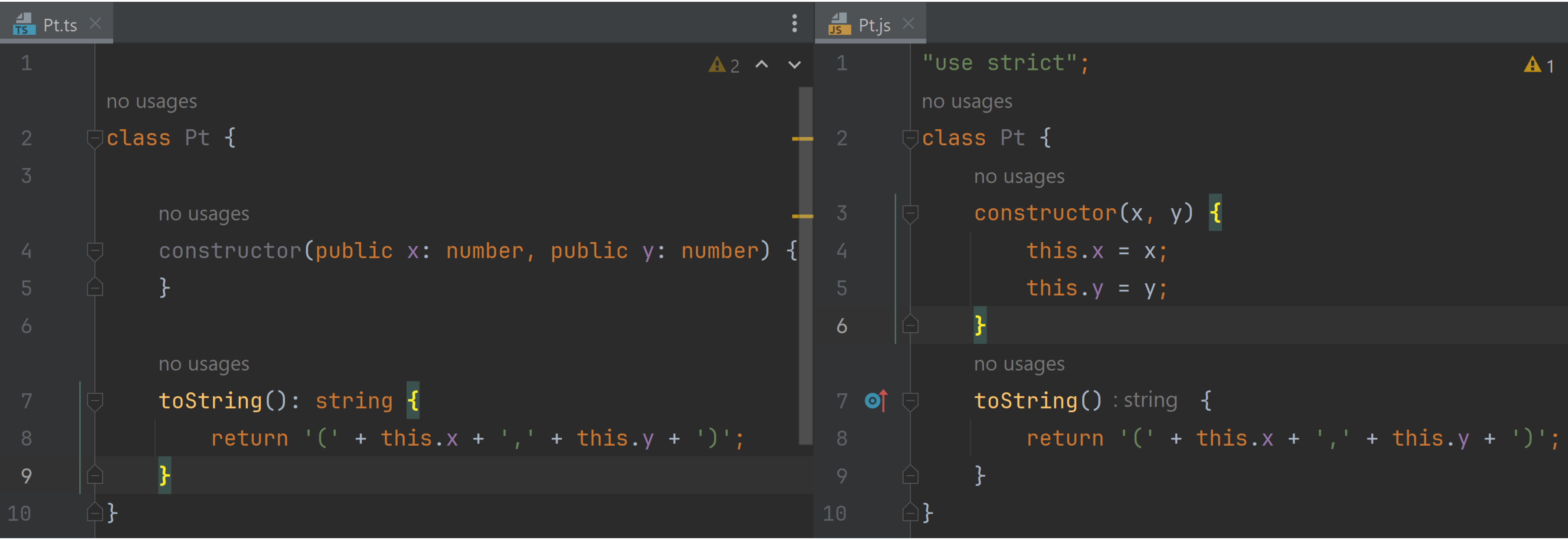
Example Emulating class with ES5



```
class Pt {

    constructor(public x: number, public y: numbe
    }

    toString(): string {
        return '(' + this.x + ',' + this.y + ')';
    }
}
```

```
"use strict";
var Pt = /** @class */ (function () {
    function Pt(x, y) {
        this.x = x;
        this.y = y;
    }
    Pt.prototype.toString = function () {
        return '(' + this.x + ',' + this.y + ')'
    };
    return Pt;
}());
```

JavaScript evolve (from TypeScript ESNext) now have class since ES2015



Pt.ts

```typescript

class Pt {

    constructor(public x: number, public y: number) {
    }

    toString(): string {
        return '(' + this.x + ',' + this.y + ')';
    }
}
```

Pt.js

```javascript
"use strict";
class Pt {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    toString() {
        return '(' + this.x + ',' + this.y + ')';
    }
}
```

JavaScript has weird "var" visibility

JavaScript has weird "var" visibility

"use strict";   // at first line

But still weird

JavaScript has weird "var" visibility

"use strict";   // at first line

use Immediatly Invoked Anonymous Function

But still weird

JavaScript has weird "var" visibility

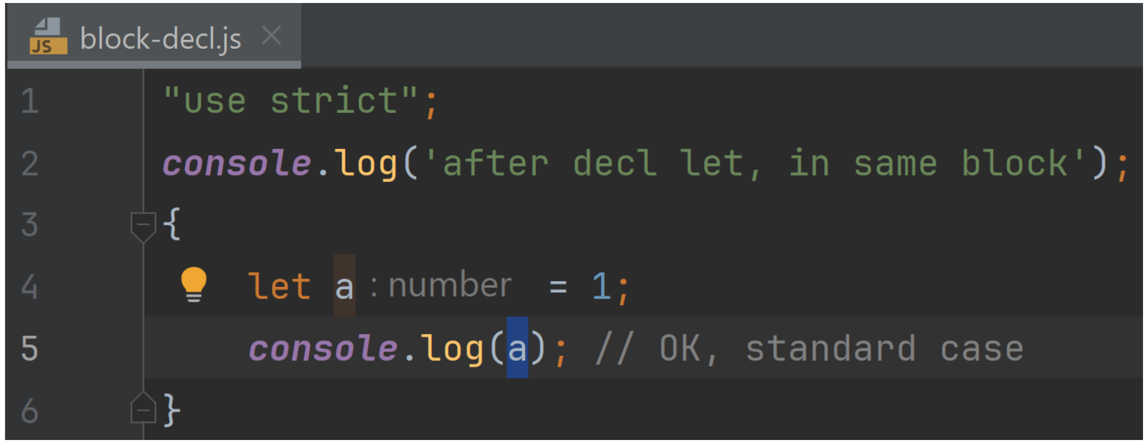
"use strict";   // at first line

use Immediatly Invoked Anonymous Function

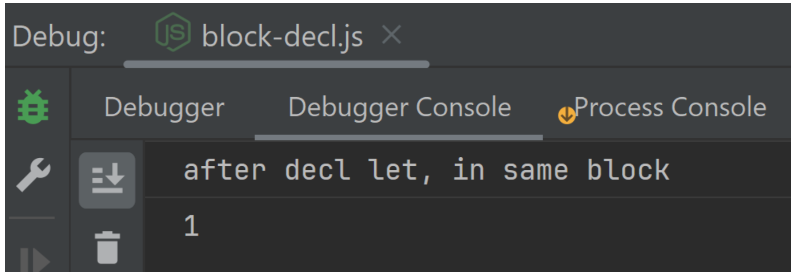
use "let" or "const" instead of "var"

use TypeScript

in "Standard Langages"
1/ declare, then 2/ init, then- 3/ use
with scope = enclosing { block }

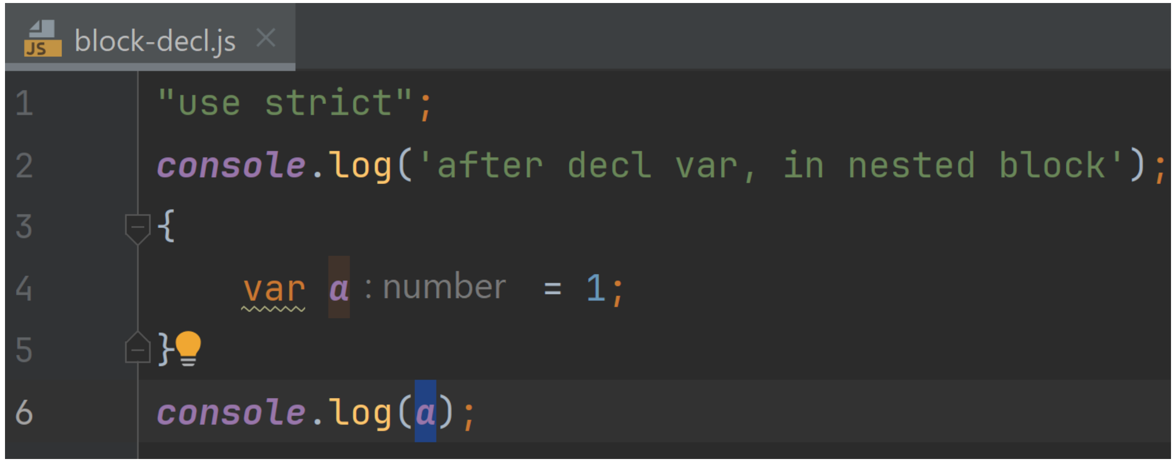


```
"use strict";
console.log('after decl let, in same block');
{
    let a :number  = 1;
    console.log(a); // OK, standard case
}
```



```
after decl let, in same block
1
```

JavaScript has weird "var" visibility

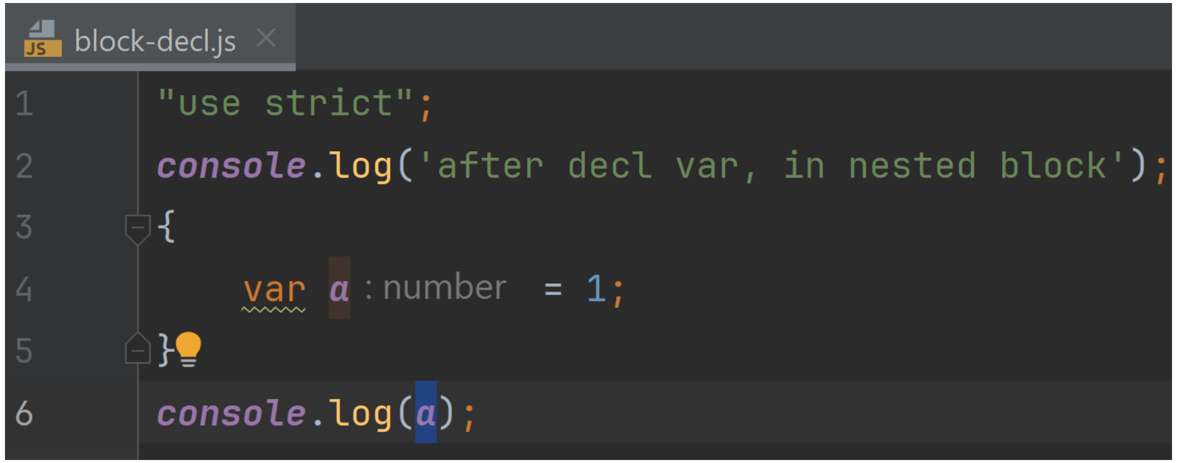


```
"use strict";
console.log('after decl var, in nested block');
{
    var a :number  = 1;
}
console.log(a);
```

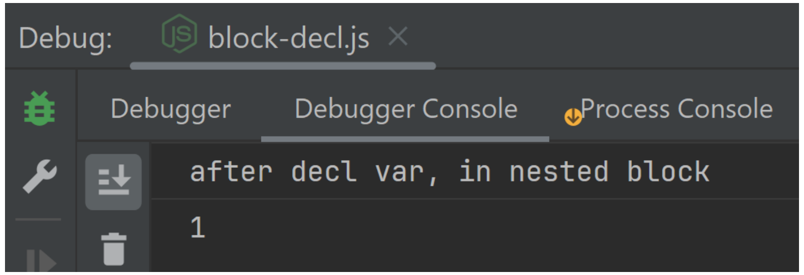
What do you think ??

a/ does not "evaluate" / "run" / "compile"
b/ throws exception
c/ print "undefined"
d/ print "1"

Answer: d/ print 1
the scope of visibility if the enclosing function (not the enclosing block)

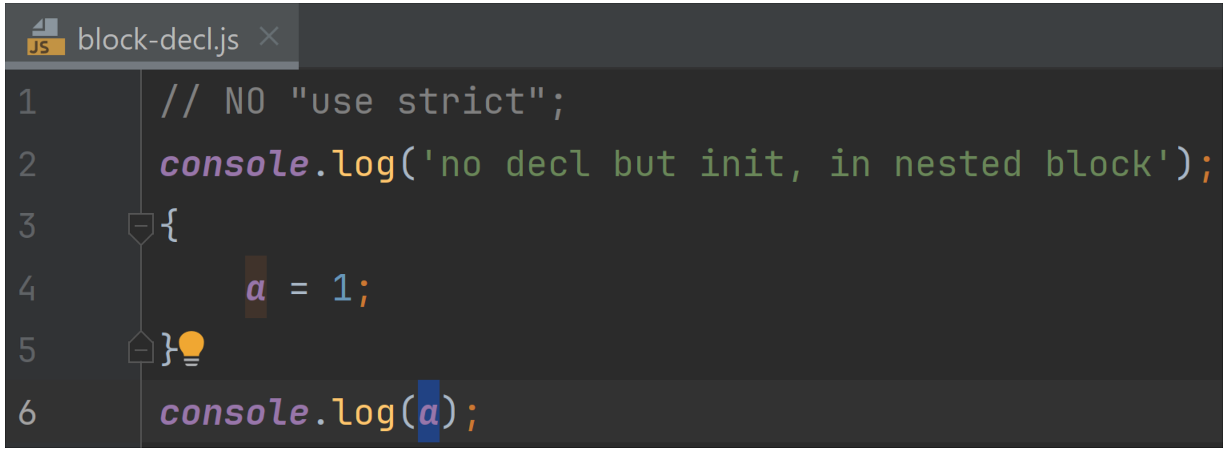


```
"use strict";
console.log('after decl var, in nested block');
{
    var a : number  = 1;
}
console.log(a);
```

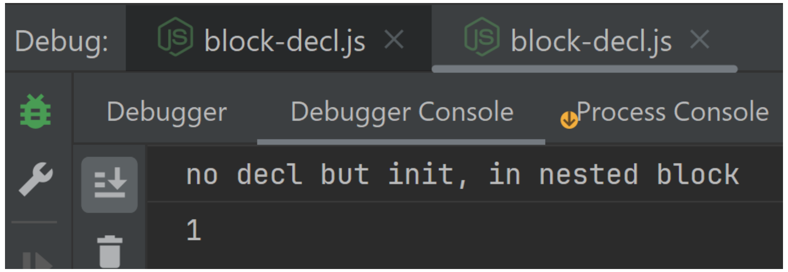


```
after decl var, in nested block
1
```

without "use strict" ... NO need to decl "var"

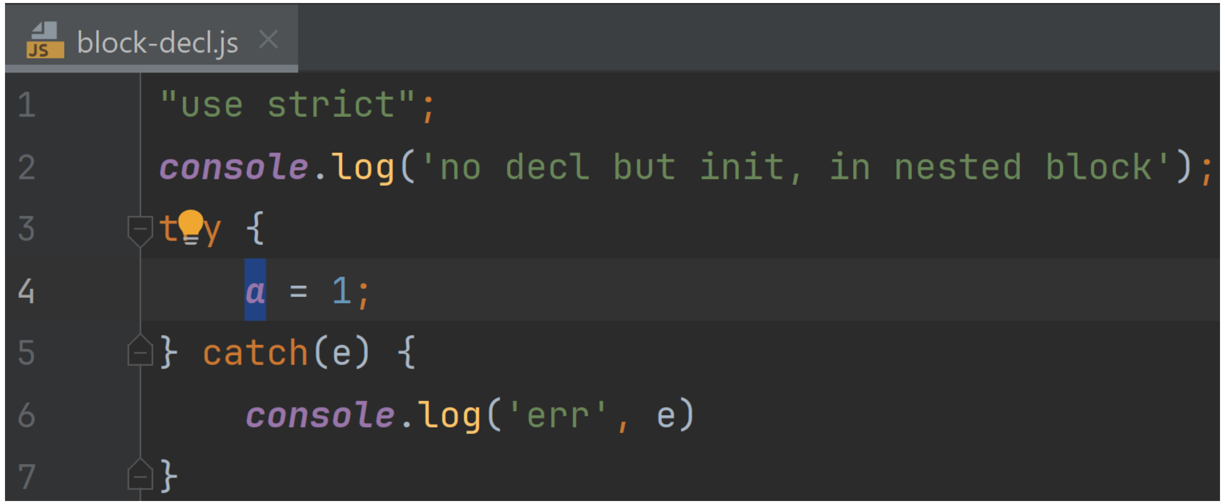


```
// NO "use strict";
console.log('no decl but init, in nested block');
{
    a = 1;
}
console.log(a);
```

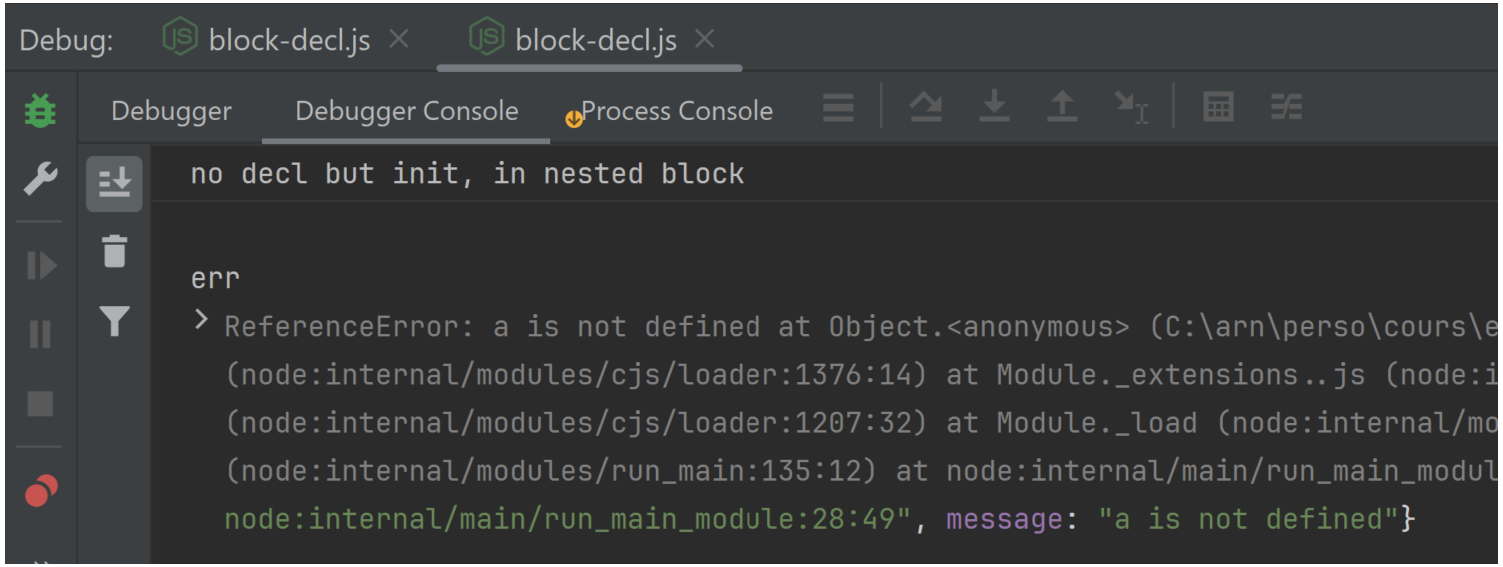


```
no decl but init, in nested block
1
```

with "use strict";

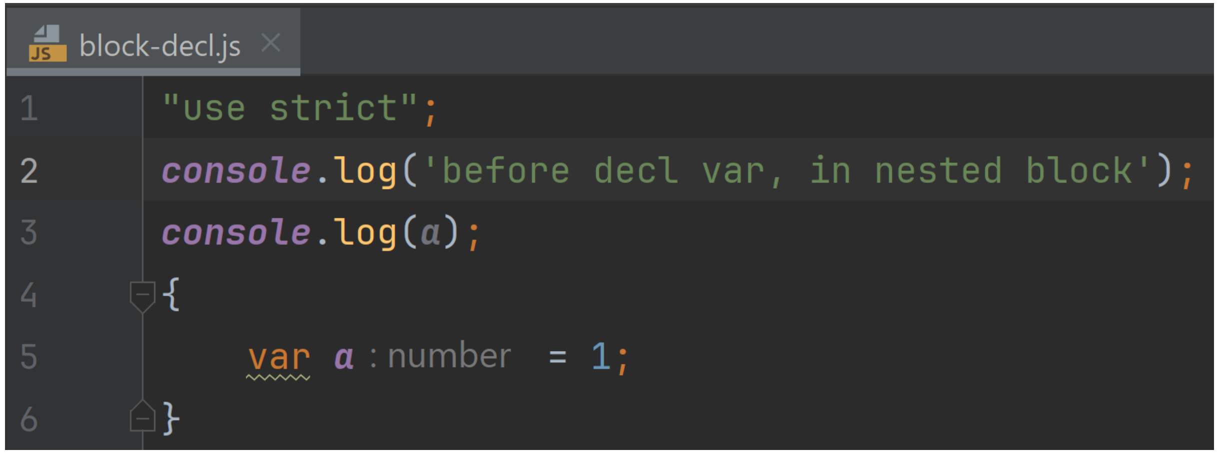


```
"use strict";
console.log('no decl but init, in nested block');
try {
    a = 1;
} catch(e) {
    console.log('err', e)
}
```

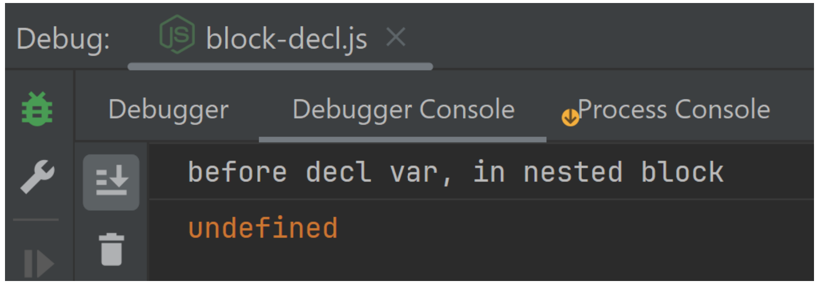


```
no decl but init, in nested block

err
> ReferenceError: a is not defined at Object.<anonymous> (C:\arn\perso\cours\e
  (node:internal/modules/cjs/loader:1376:14) at Module._extensions..js (node:i
  (node:internal/modules/cjs/loader:1207:32) at Module._load (node:internal/mo
  (node:internal/modules/run_main:135:12) at node:internal/main/run_main_modul
  node:internal/main/run_main_module:28:49", message: "a is not defined"}
```

even stranger:
use before declare in nested block

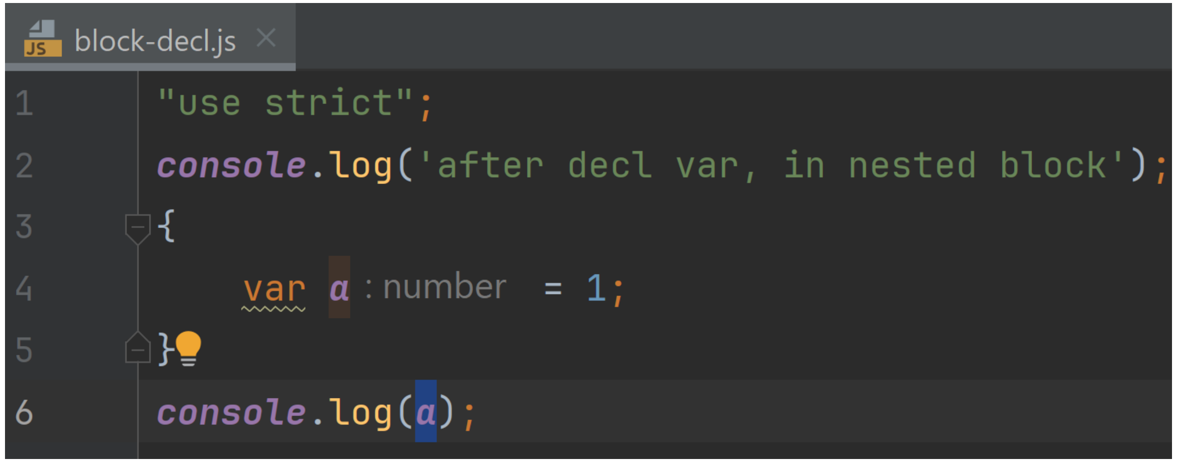


```
"use strict";
console.log('before decl var, in nested block');
console.log(a);
{
    var a : number  = 1;
}
```

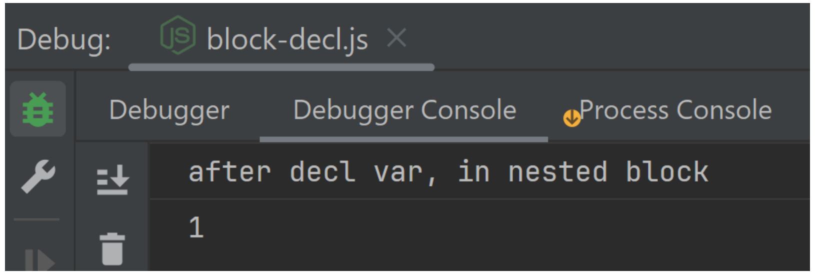


```
before decl var, in nested block
undefined
```

other strange "var" visibility

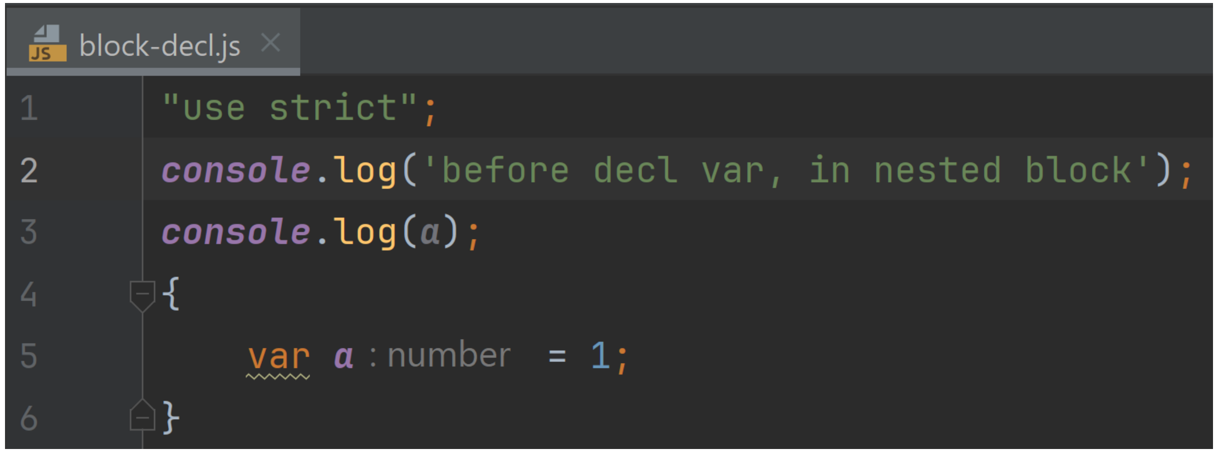


```
"use strict";
console.log('after decl var, in nested block');
{
    var a : number  = 1;
}
console.log(a);
```

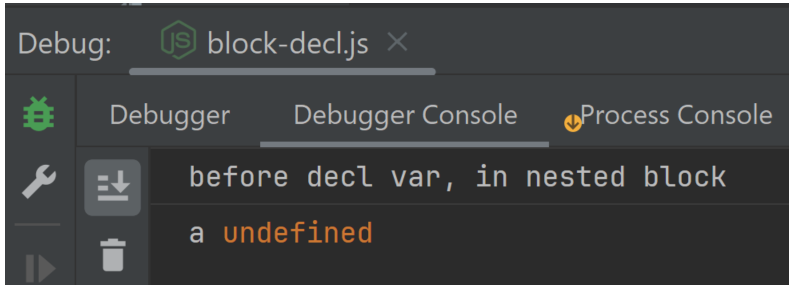


```
after decl var, in nested block
1
```

JavaScript has weird "var" visibility



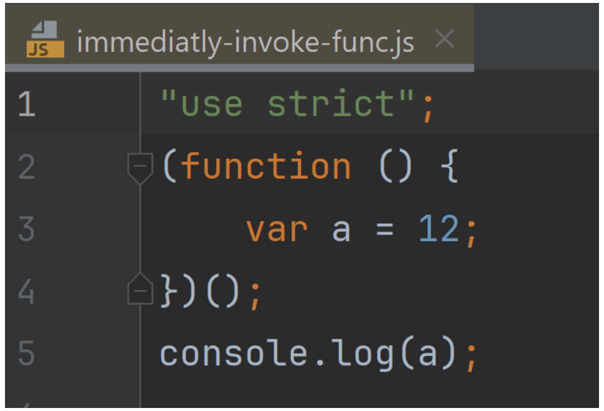
```
"use strict";
console.log('before decl var, in nested block');
console.log(a);
{
    var a :number  = 1;
}
```



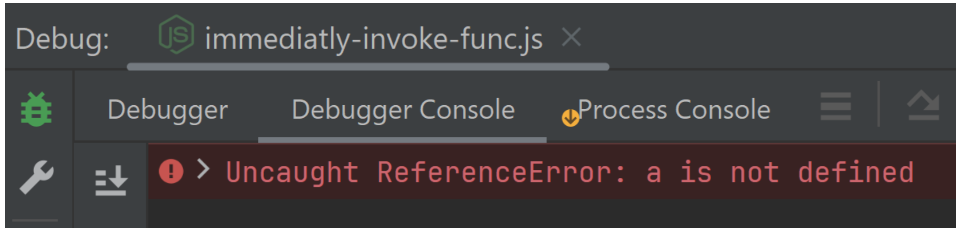
```
before decl var, in nested block
a undefined
```

declared "ok"
BUT not initialized yet => undefined !

work around for scope :
"Immediatly Invoked Anonymous Function"

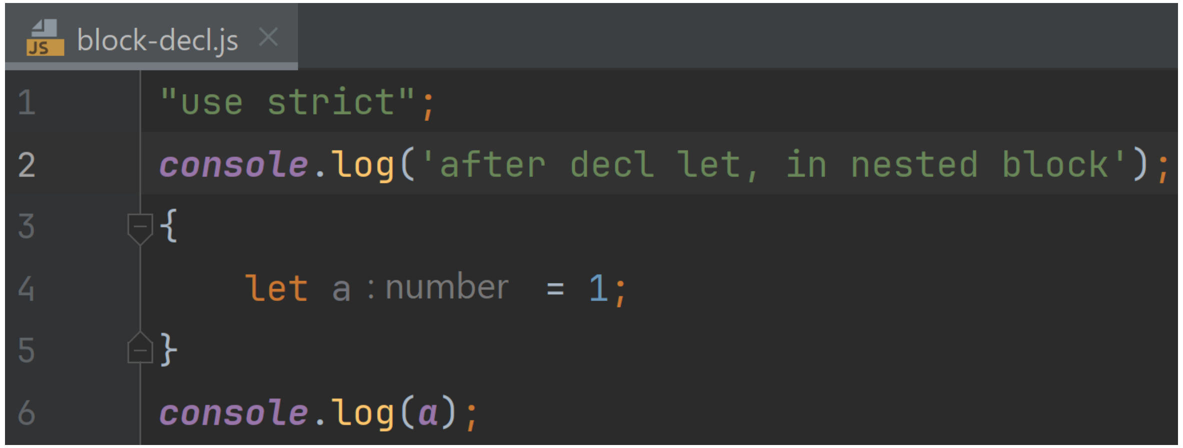


```
"use strict";
(function () {
    var a = 12;
})();
console.log(a);
```

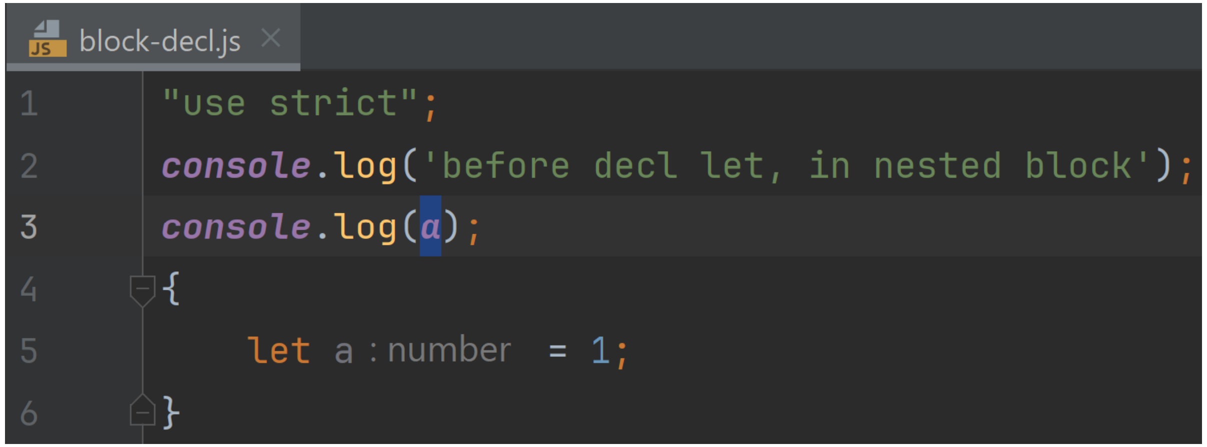


```
> Uncaught ReferenceError: a is not defined
```

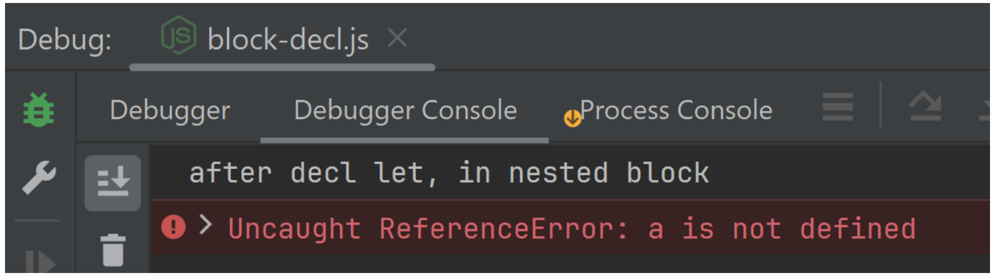
"let" | "const" ... as "var" replacement



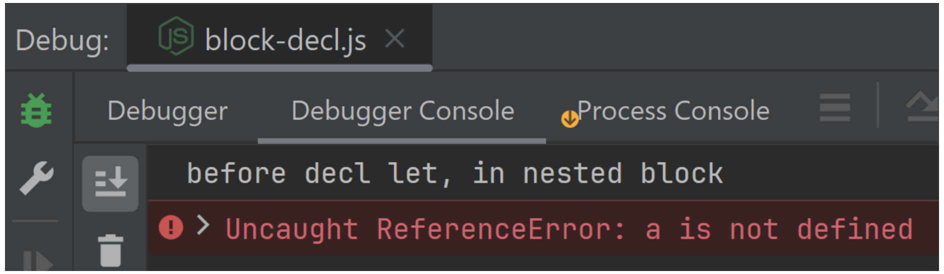
```
"use strict";
console.log('after decl let, in nested block');
{
    let a : number  = 1;
}
console.log(a);
```



```
"use strict";
console.log('before decl let, in nested block');
console.log(a);
{
    let a : number  = 1;
}
```

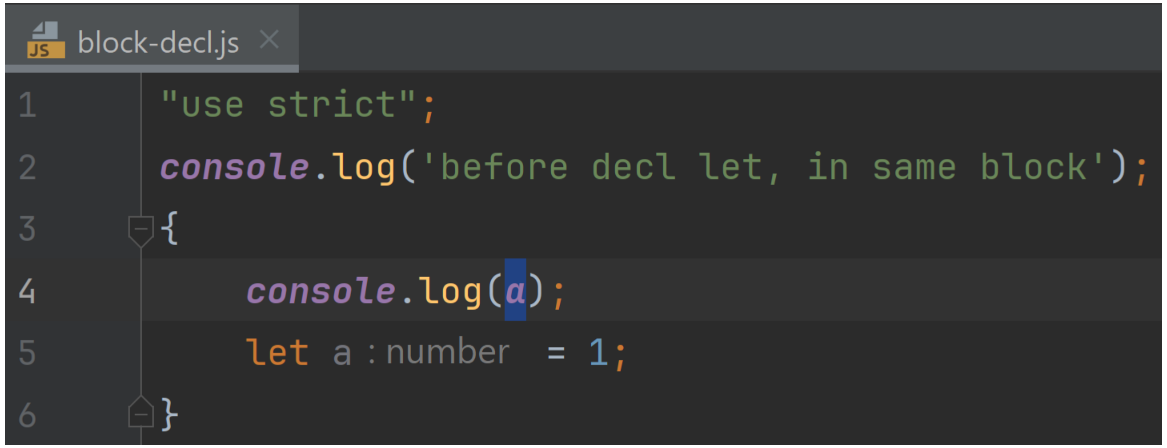


```
after decl let, in nested block
Uncaught ReferenceError: a is not defined
```

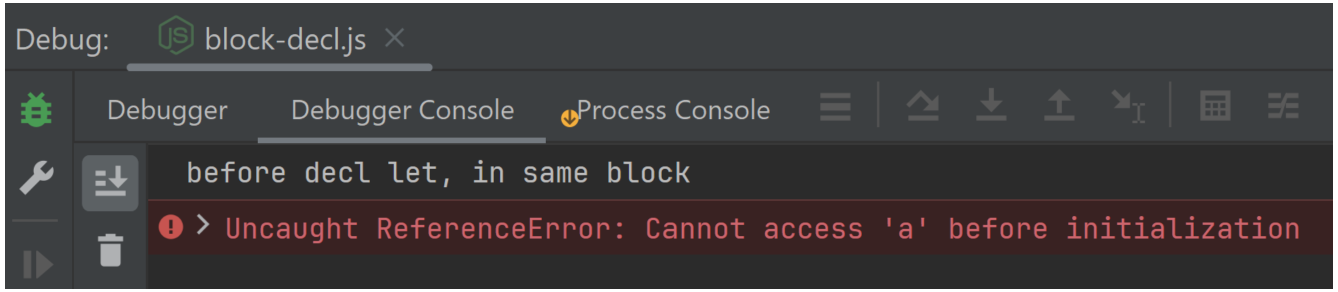


```
before decl let, in nested block
Uncaught ReferenceError: a is not defined
```

"let" (or "const") in same block

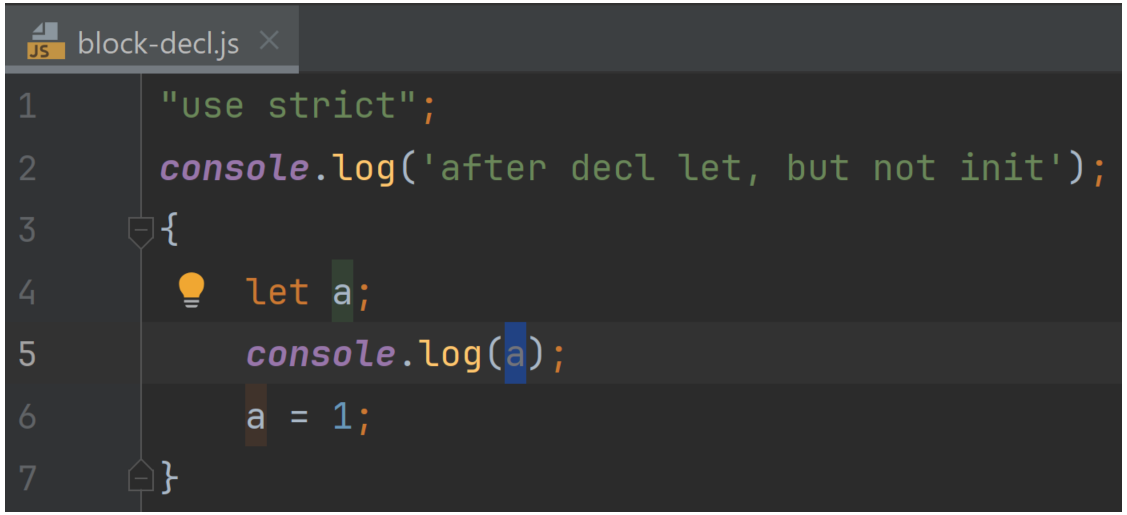


```
"use strict";
console.log('before decl let, in same block');
{
    console.log(a);
    let a : number  = 1;
}
```

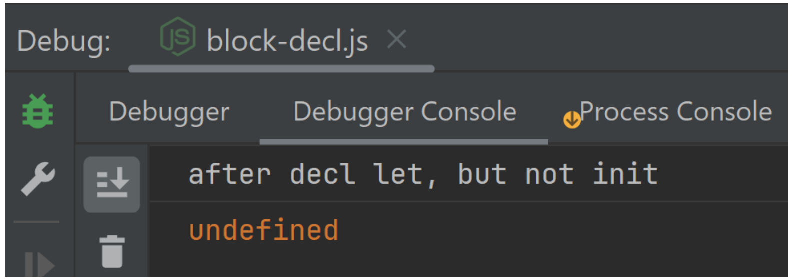


```
before decl let, in same block
Uncaught ReferenceError: Cannot access 'a' before initialization
```

not init (ok in JS !)
but does not compile in TypeScript



```
"use strict";
console.log('after decl let, but not init');
{
    let a;
    console.log(a);
    a = 1;
}
```



```
after decl let, but not init
undefined
```

everybody hate JavaScript, for million reasons

100 reasons you need TypeScript
and will probably love it

Spoiler Alert

TypeScript is NOT "more efficient" than JS

... it is just typed-checked at Compile-Time
then translated to JS

at Runtime, you still have poor old
Node.js or Web Browser

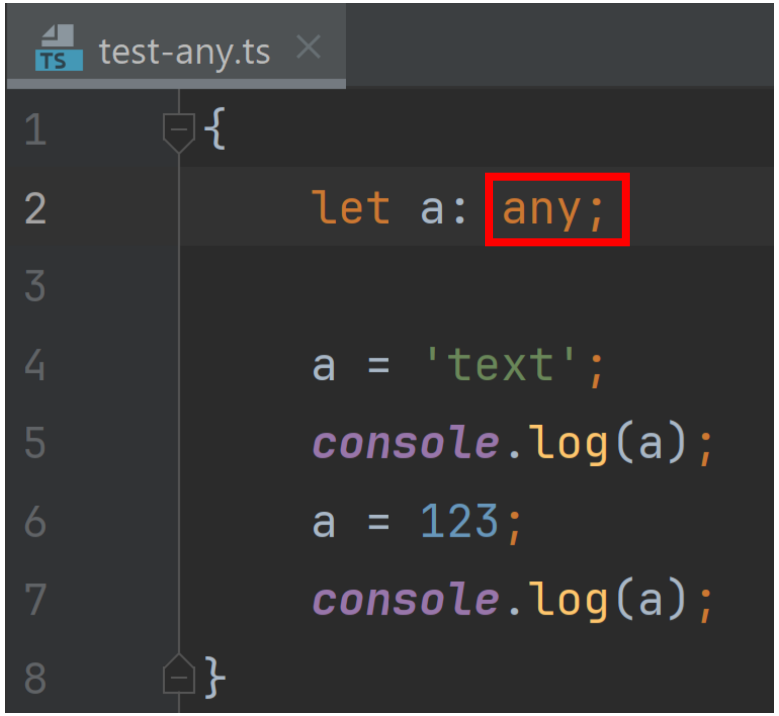
TypeScript Types

Primitive Types (for values) : any, null, undefined, boolean, number, string

Composite Types:

" **|** " Union (choice between types)
" **&** " combination of type constraint
"**readonly**" type modifier
Array
Interfaces : type constraint on existence of field / partial field in Object
" **?** " Optional Field Type (for accepting "undefined")
Function, Class, Templates, etc.

any ?



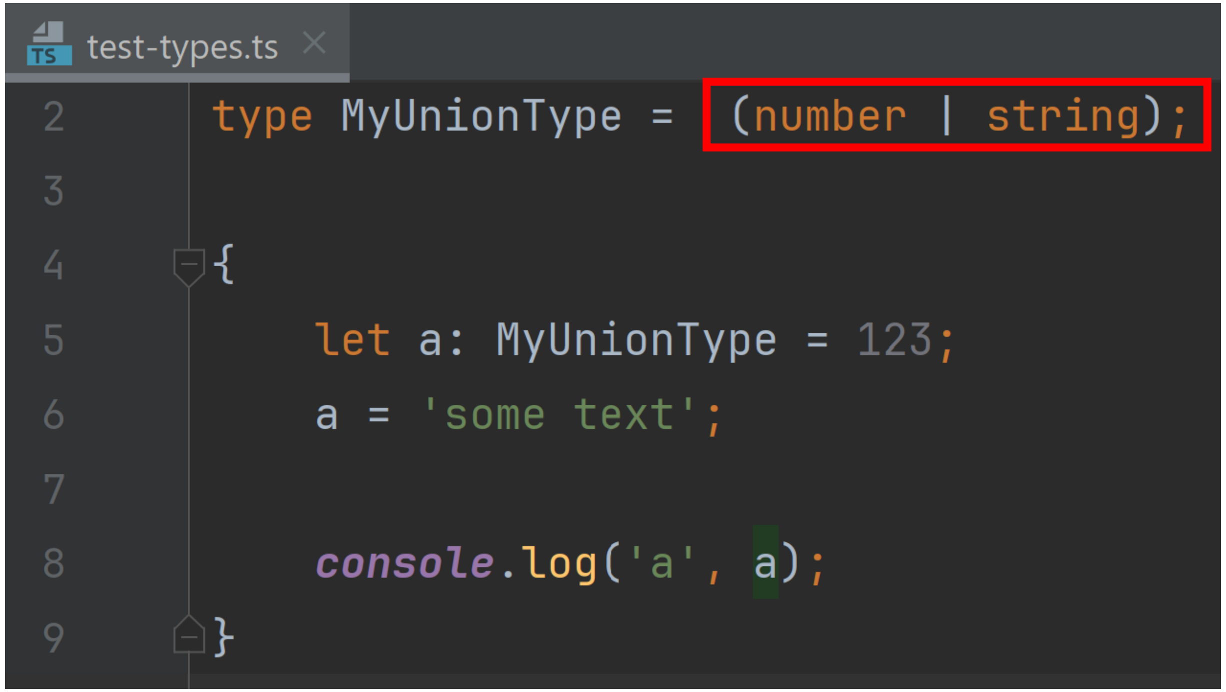
```
{
    let a: any;

    a = 'text';
    console.log(a);
    a = 123;
    console.log(a);
}
```

any ... as name implies
= No Type checking !

To use TEMPORARILY while porting JS to Ts

Good Ts program(er)s should never use "any"

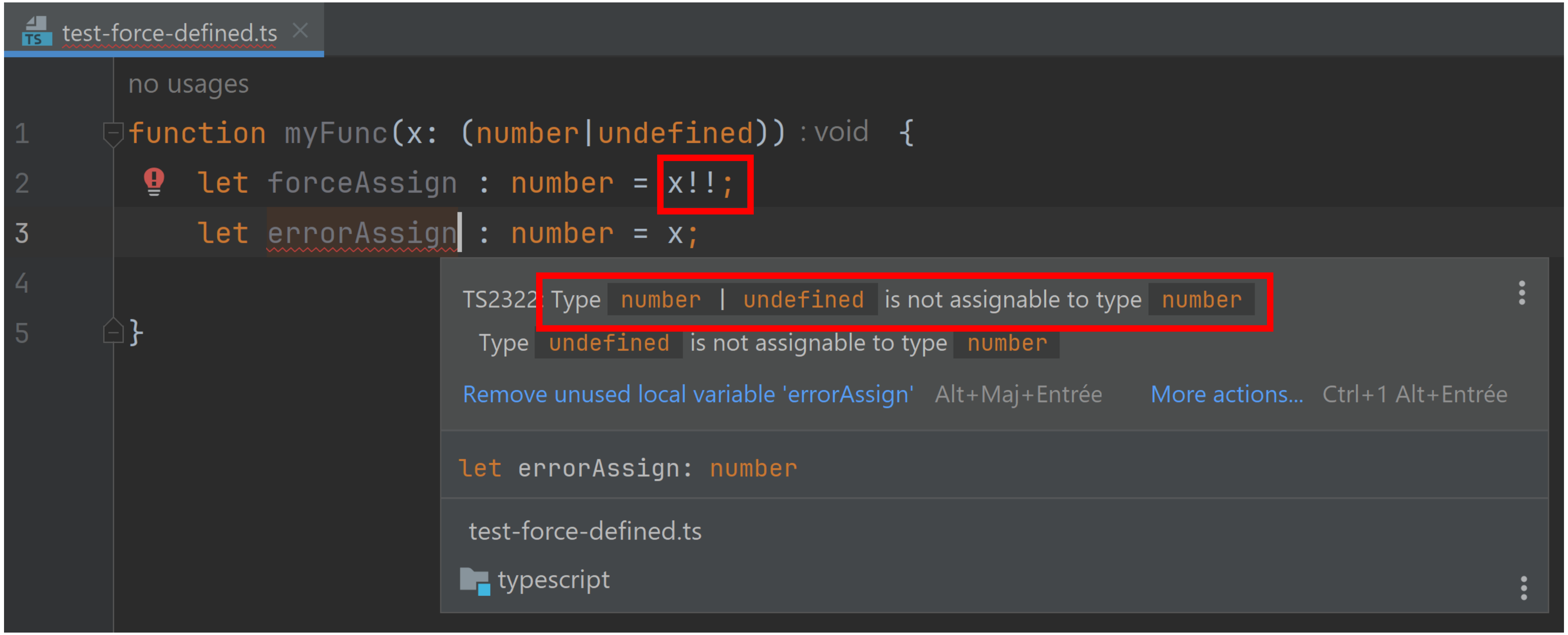
Union Type



```
type MyUnionType = (number | string);

{
    let a: MyUnionType = 123;
    a = 'some text';

    console.log('a', a);
}
```

Example usages
Type | null
Type |undefined

Force type-check with "!!"



```
no usages
function myFunc(x: (number|undefined)) : void  {
    let forceAssign : number = x!!;
    let errorAssign : number = x;

}
```

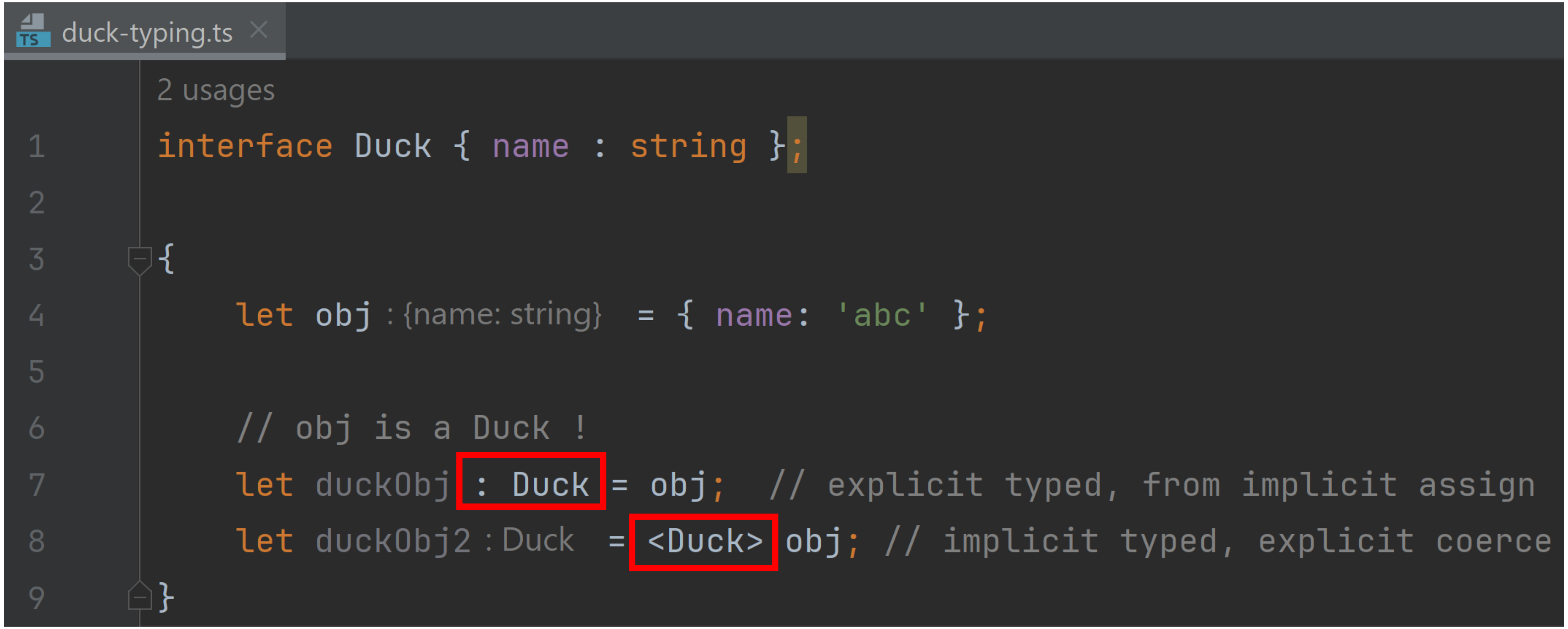
TS2322: Type `number | undefined` is not assignable to type `number`
Type `undefined` is not assignable to type `number`

Remove unused local variable 'errorAssign' Alt+Maj+Entrée
More actions... Ctrl+1 Alt+Entrée

```
let errorAssign: number
```

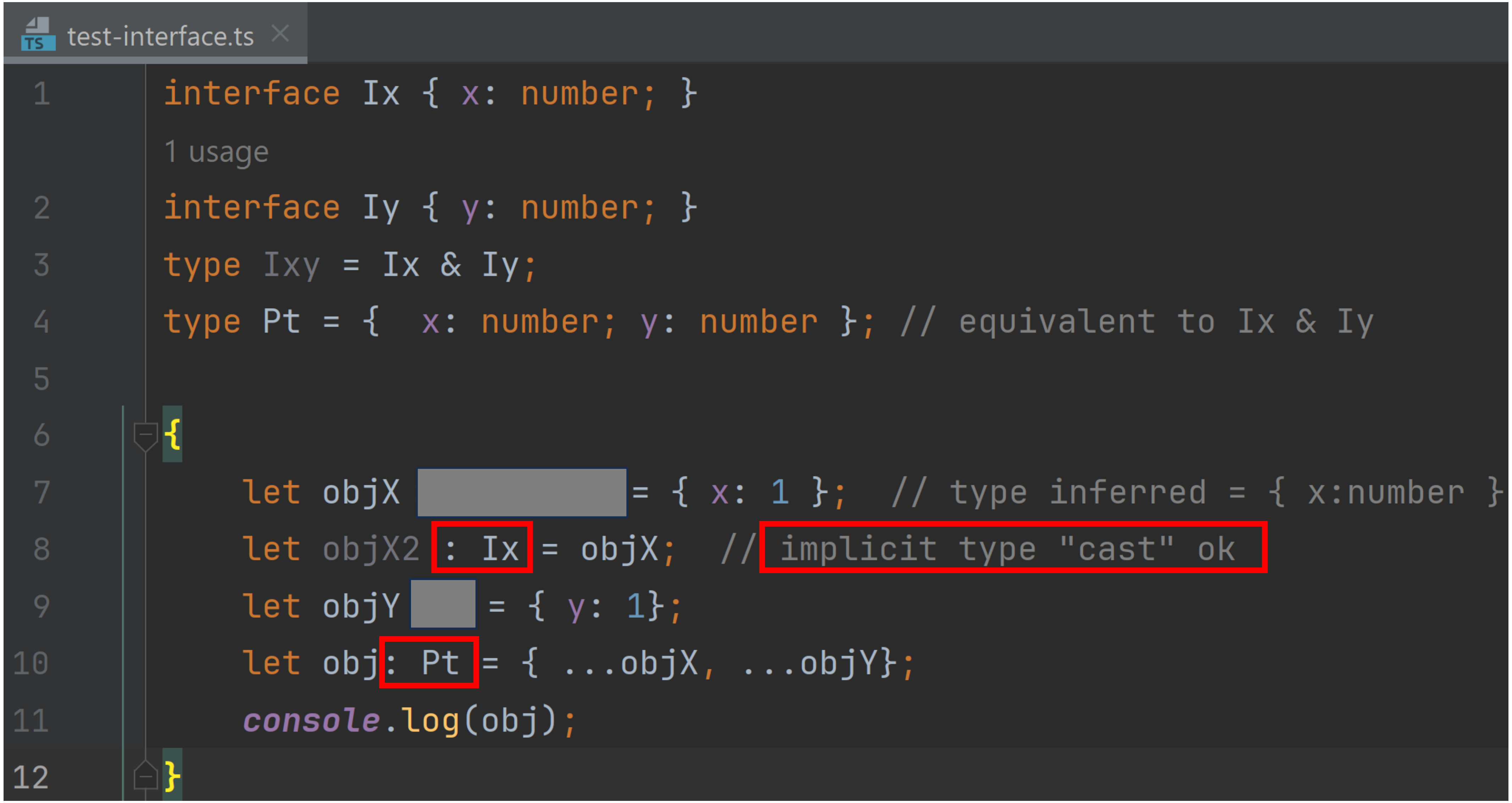
test-force-defined.ts
typescript

Duck Typing



```
2 usages
interface Duck { name : string };

{
    let obj : {name: string}  = { name: 'abc' };

    // obj is a Duck !
    let duckObj : Duck = obj;  // explicit typed, from implicit assign
    let duckObj2 : Duck  = <Duck> obj; // implicit typed, explicit coerce
}
```

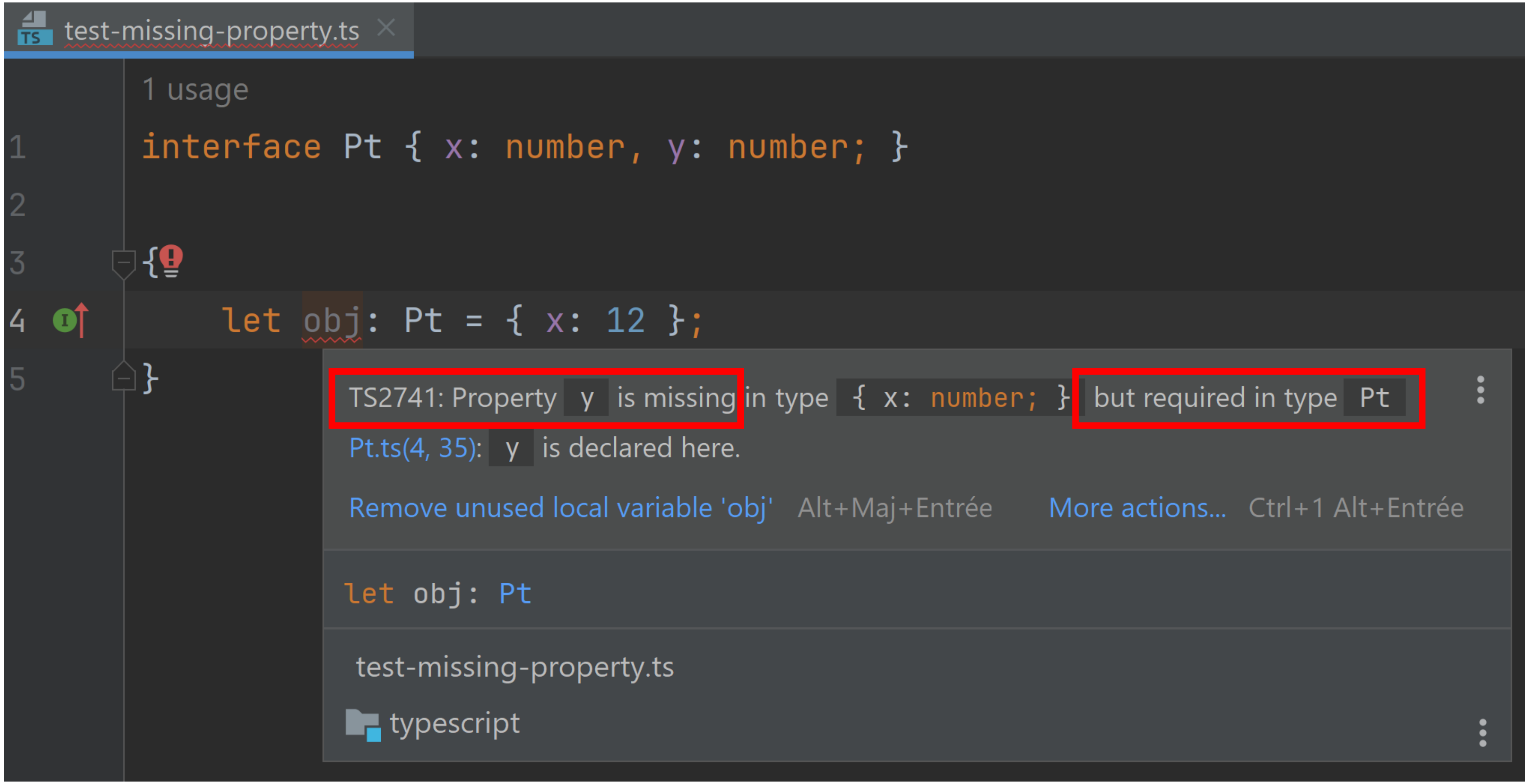
Structural Interface Typing, Combine Partial



```
interface Ix { x: number; }
1 usage
interface Iy { y: number; }
type Ixy = Ix & Iy;
type Pt = {  x: number; y: number }; // equivalent to Ix & Iy

{
    let objX           = { x: 1 };  // type inferred = { x:number }
    let objX2 : Ix = objX;  // implicit type "cast" ok
    let objY    = { y: 1};
    let obj: Pt = { ...objX, ...objY};
    console.log(obj);
}
```

▨ = type inferred (displayed by IntelliJ), NOT edited in file

Type Checking "missing property" in Interface



```
1 usage
interface Pt { x: number, y: number; }

{
    let obj: Pt = { x: 12 };
}
```

```
TS2741: Property y is missing in type { x: number; } but required in type Pt
Pt.ts(4, 35): y is declared here.
```

Remove unused local variable 'obj' Alt+Maj+Entrée More actions... Ctrl+1 Alt+Entrée

```
let obj: Pt
```

test-missing-property.ts

typescript