

The image features a stylized blue silhouette of a city skyline at the bottom. Above the skyline, the word "TypeScript" is written in a blue, sans-serif font. The text is positioned in front of a large, white, stylized cloud that occupies the upper right portion of the image. A smaller, white, stylized cloud is located in the upper left corner. The background is a light gray gradient.

TypeScript

# TypeScript là gì ?

- TypeScript là một dự án mã nguồn mở được phát triển bởi Microsoft
- TypeScript là một phiên bản nâng cao của javascript
- TypeScript bổ sung tùy chọn kiểu tĩnh và lớp hướng đối tượng mà điều này không có ở Javascript.
- TypeScript có thể sử dụng để phát triển các ứng dụng chạy ở client-side (Angular2) và server-side (NodeJS).
- TypeScript sử dụng tất cả các tính năng của của ECMAScript 2015 (ES6) như classes, modules.

# Tại sao nên sử dụng TypeScript?

- ***Dễ phát triển dự án lớn:***

Với việc sử dụng các kỹ thuật mới nhất và lập trình hướng đối tượng nên TypeScript giúp chúng ta phát triển các dự án lớn một cách dễ dàng.

- ***Hỗ trợ các tính năng của Javascript phiên bản mới nhất:***

TypeScript luôn đảm bảo việc sử dụng đầy đủ các kỹ thuật mới nhất của Javascript, ví dụ như version hiện tại là ECMAScript 2015 (ES6).

- ***Là mã nguồn mở:***

TypeScript là một mã nguồn mở nên hoàn toàn có thể sử dụng miễn phí, bên cạnh đó còn được cộng đồng hỗ trợ lớn.

- ***Bản chất của Typescript là javascript:***

TypeScript sau khi biên dịch nó tạo ra các đoạn mã javascript vì vậy có thể chạy ở bất kỳ đâu mà javascript được hỗ trợ

# TypeScript

*Một số tính năng mới: (Classes, Types, Interfaces ...)*



*Compiled to*

# JavaScript

# Ví dụ:

## Code TypeScript

```
1 class Customer {  
2     Name : string;  
3     constructor (firstName: string, lastName: string)  
4     {  
5         this.Name = firstName + " " + lastName;  
6     }  
7     GetName()  
8     {  
9         return "Hello, " + this.Name;  
10    }  
11 }
```

## Biên dịch thành Javascript

```
1 var Customer = (function () {  
2     function Customer(firstName, lastName) {  
3         this.Name = firstName + " " + lastName;  
4     }  
5     Customer.prototype.GetName = function () {  
6         return "Hello, " + this.Name;  
7     };  
8     return Customer;  
9 }());
```

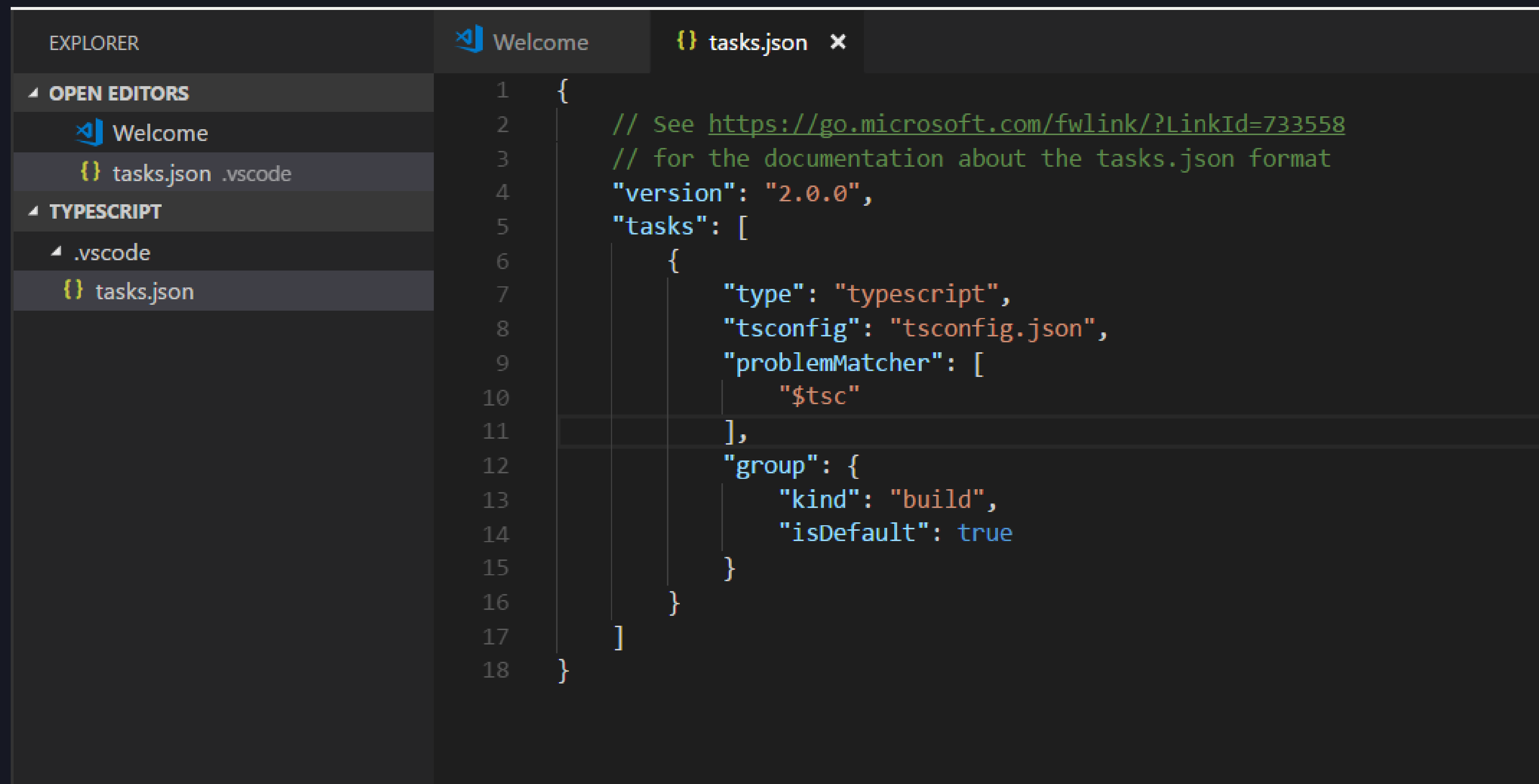
*Một đoạn chương trình mẫu được biên dịch từ typescript sang javascript (Nguồn: <https://freetuts.net>)*

# Cài đặt TypeScript và chạy demo đầu tiên?

1. Cài đặt công cụ lập trình Visual Code(<https://code.visual.com>) hoặc sublimetext.
2. Cài đặt Nodejs ( <https://nodejs.org/en/> )
3. Cài đặt typescript => **Mở terminal: npm install typescript -g**
4. Tạo folder chứa project dự án
5. Dùng công cụ Visual code mở project. Nhấn tổ hợp phím:  
**ctrl + shift + B => Configure Build Task => Open tasks.json**

6. Sau khi cài đặt xong project phát sinh file task.js (điền thông tin vào file như hình).

Hoặc lên trang copy về: <https://code.visualstudio.com/docs/editor/tasks>

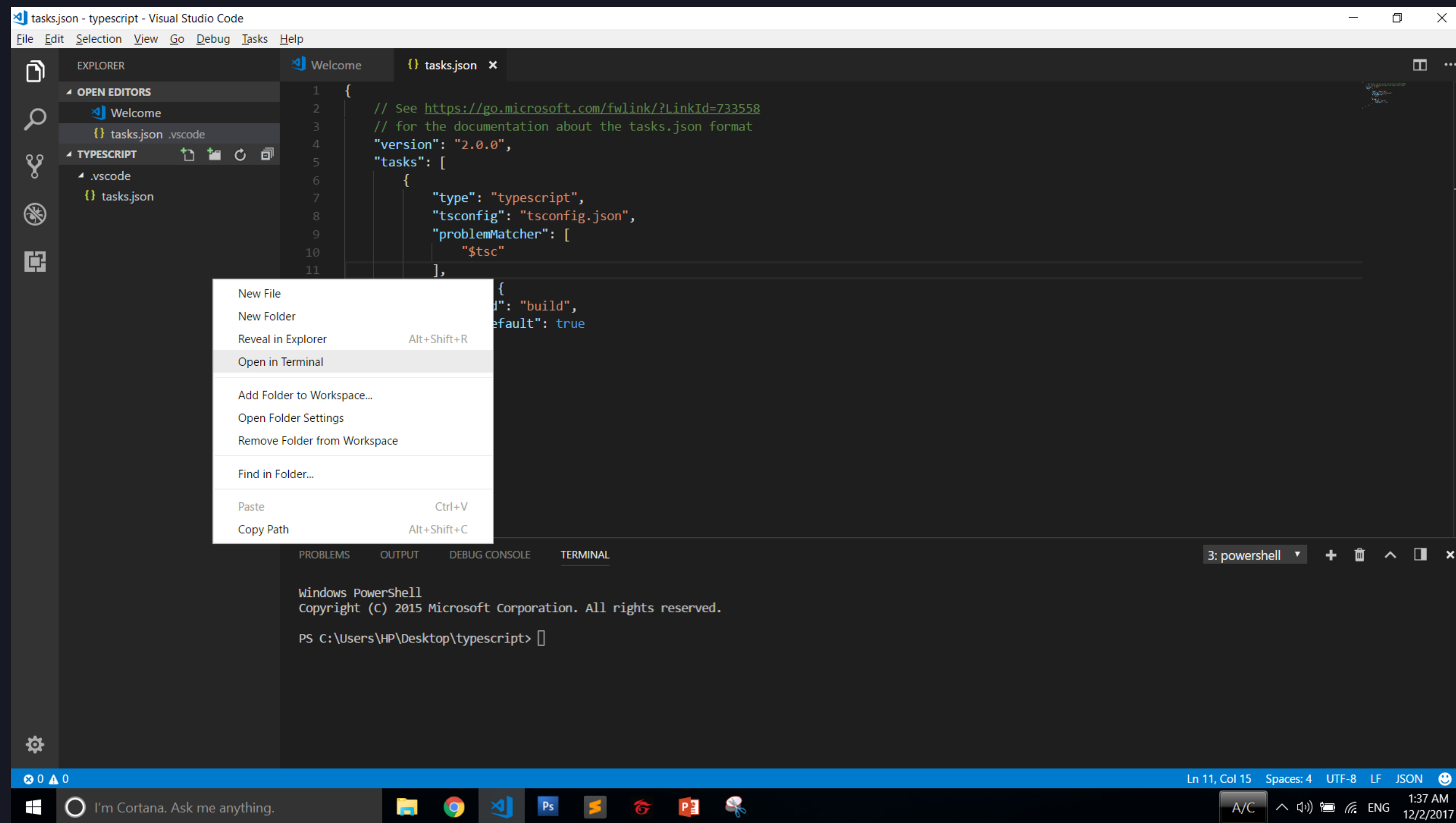


The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left and the Editor area on the right. The Explorer sidebar shows the file structure with 'tasks.json' selected under the '.vscode' folder. The Editor area displays the content of 'tasks.json' with line numbers 1 through 18. The code is a JSON configuration for a task named 'build'.

```
1 {
2     // See https://go.microsoft.com/fwlink/?LinkId=733558
3     // for the documentation about the tasks.json format
4     "version": "2.0.0",
5     "tasks": [
6         {
7             "type": "typescript",
8             "tsconfig": "tsconfig.json",
9             "problemMatcher": [
10                "$tsc"
11            ],
12            "group": {
13                "kind": "build",
14                "isDefault": true
15            }
16        }
17    ]
18 }
```



7. Cài đặt module ES6 để biên dịch code từ typescript sang javascript bằng cách tại thư mục gốc của project ( Lưu ý đồng cấp với .vscode) click phải -> Open in terminal.

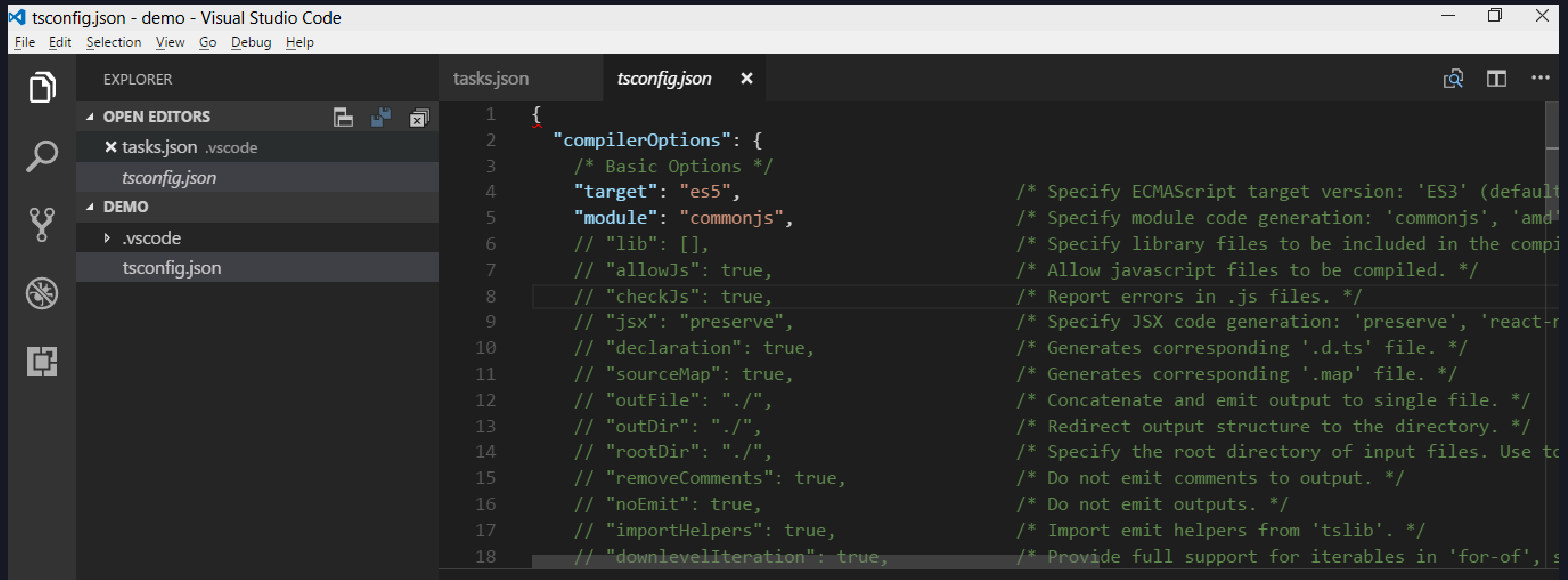


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop\demo>tsc --init
message TS6071: Successfully created a tsconfig.json file.

C:\Users\Administrator\Desktop\demo>
```





```
tsconfig.json - demo - Visual Studio Code
File Edit Selection View Go Debug Help

EXPLORER
OPEN EDITORS
x tasks.json .vscode
tsconfig.json
DEMO
.vsphere
tsconfig.json

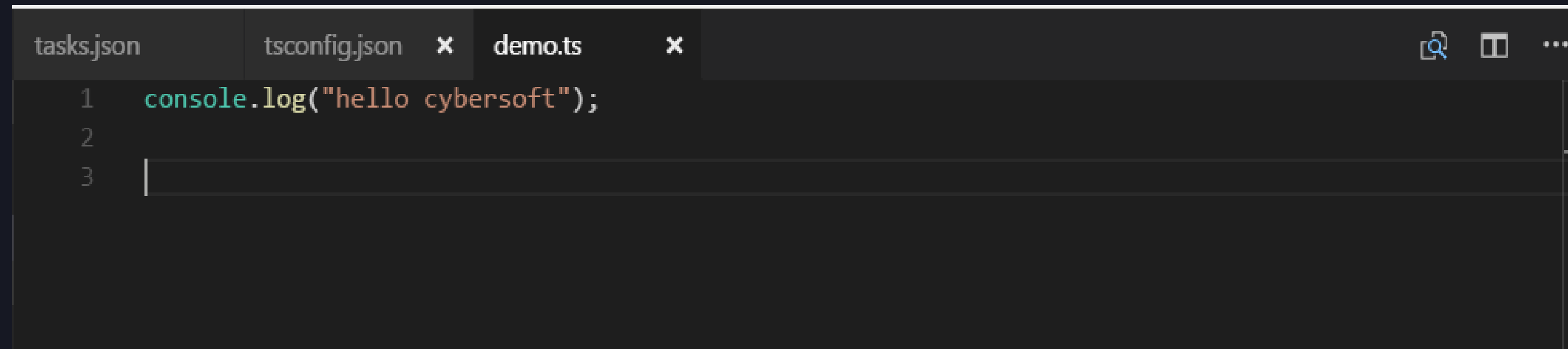
1 {
2   "compilerOptions": {
3     /* Basic Options */
4     "target": "es5",           /* Specify ECMAScript target version: 'ES3' (default)
5     "module": "commonjs",      /* Specify module code generation: 'commonjs', 'amd'
6     // "lib": [],              /* Specify library files to be included in the compilation
7     // "allowJs": true,        /* Allow javascript files to be compiled. */
8     // "checkJs": true,        /* Report errors in .js files. */
9     // "jsx": "preserve",      /* Specify JSX code generation: 'preserve', 'react-r
10    // "declaration": true,     /* Generates corresponding '.d.ts' file. */
11    // "sourceMap": true,       /* Generates corresponding '.map' file. */
12    // "outFile": "./",         /* Concatenate and emit output to single file. */
13    // "outDir": "./",          /* Redirect output structure to the directory. */
14    // "rootDir": "./",         /* Specify the root directory of input files. Use to
15    // "removeComments": true,  /* Do not emit comments to output. */
16    // "noEmit": true,          /* Do not emit outputs. */
17    // "importHelpers": true,   /* Import emit helpers from 'tslib'. */
18    // "downlevelIteration": true, /* Provide full support for iterables in 'for-of', s
```

*Sau khi tạo ta được file tsconfig.json chứa các thông tin cấu hình của typescript.*

## 8. Rút gọn file tsconfig.json

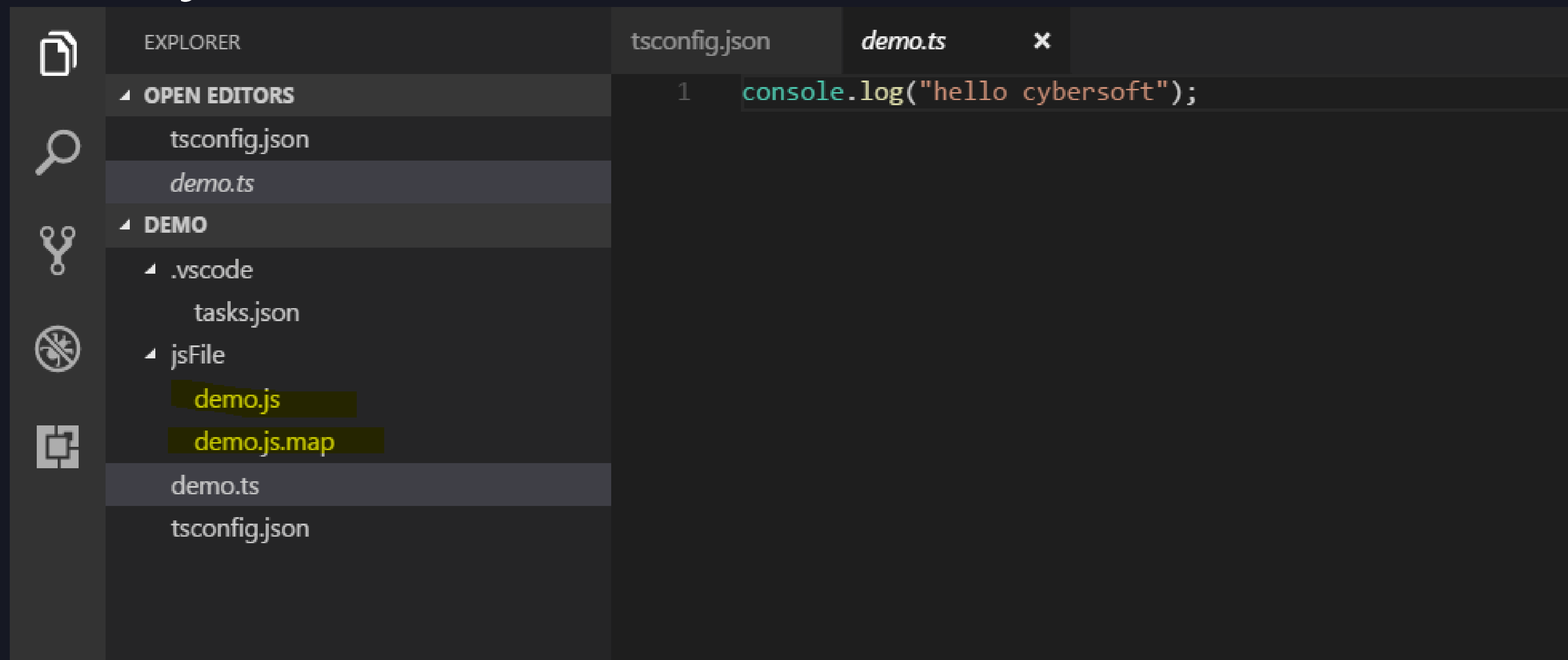
```
{
  "compilerOptions": {
    /* Basic Options */
    "target": "es5",
    "module": "commonjs",
    "sourceMap": true, //cho phép tạo ra file source map
    "outDir": "./jsFile" //Thư mục tạo ra file js được biên dịch từ file ts
  }
}
```

## 9. Tạo file demo.ts

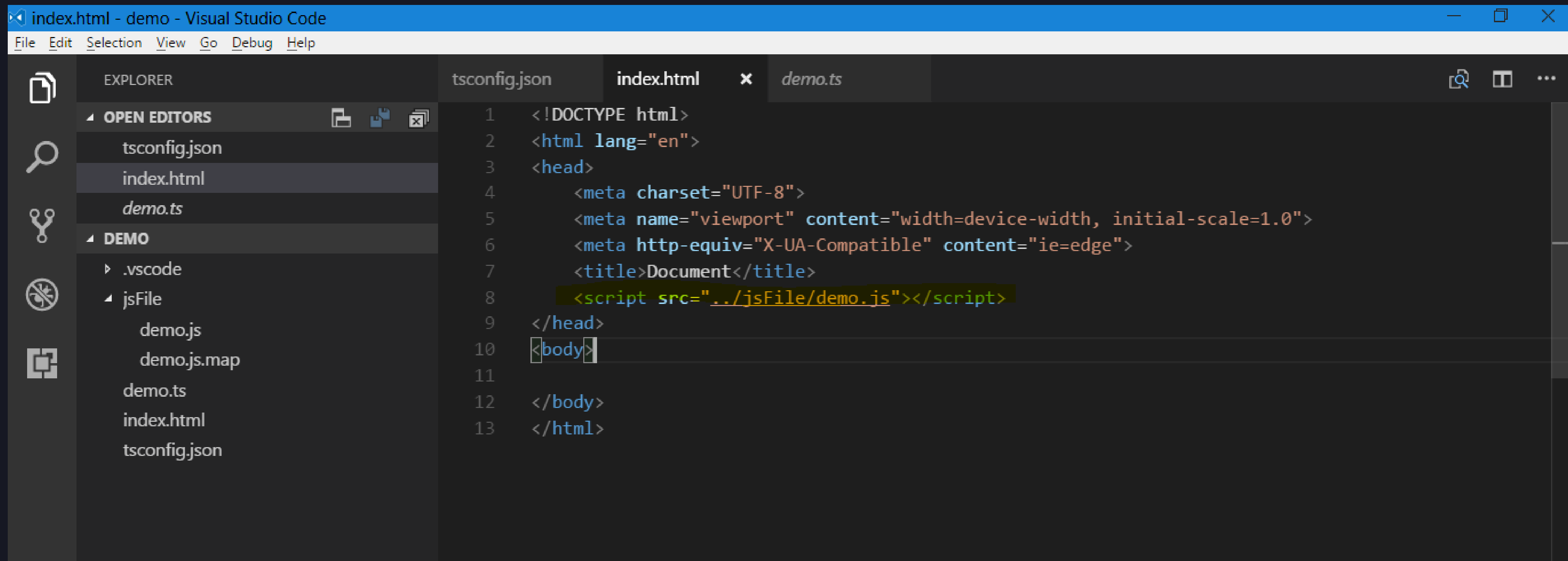


The screenshot shows a code editor with three tabs: tasks.json, tsconfig.json, and demo.ts. The demo.ts tab is active, showing a single line of code: `console.log("hello cybersoft");`. The line number 1 is visible on the left side of the editor.

10. Tiếp đến ta nhấn tổ hợp ctrl + shift + B để build => nó sẽ sinh ra file demo.js



11. Để chạy file script đó ta tạo 1 file html sau đó chèn vào thẻ script link đến file đó như bình thường.

A screenshot of the Visual Studio Code editor interface. The title bar at the top reads "index.html - demo - Visual Studio Code". Below it is a menu bar with "File", "Edit", "Selection", "View", "Go", "Debug", and "Help". The left sidebar contains the Explorer view, which is expanded to show a file tree. The tree has a "DEMO" folder containing ".vscode", "jsFile", "demo.js", "demo.js.map", "demo.ts", "index.html", and "tsconfig.json". The "index.html" file is selected and its content is displayed in the main editor area. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Document</title>
8   <script src="../../jsFile/demo.js"></script>
9 </head>
10 <body>
11
12 </body>
13 </html>
```

12. Để chạy file html ta có thể ra ngoài click vào file .html hoặc cài đặt server local để test. (Lưu ý: nên cài localhost để ứng dụng những bài sau)

## Lite-server & cài đặt lite-server

Lite-server là gì: lite server là 1 server local giả lập môi trường server để test các chức năng post, get dữ liệu lên server. Liteserver cũng như server xamp trong php.

Để cài đặt lite-server ta cũng sử dụng cú pháp npm của nodejs: **npm install lite-server**

```
C:\Users\Administrator\Desktop\demo>npm install lite-server
[.....] / fetchMetadata: sill mapToRegistry uri https://registry.n
```

Sau khi cài đặt lite-server xong ta chưa thể sử dụng được, để áp dụng lite-server vào project. Phải cài đặt thêm file **package.json**. File này quản lý thông tin cũng như các gói thư viện cài đặt vào project.

Cú pháp tạo file package.json: **npm init**

# Cài đặt package.json

## 1. Cú pháp

```
C:\Users\Administrator\Desktop\demo>npm init
```

## 2. Điền vào thông tin name còn lại bỏ trống cũng được => yes

```
C:\Users\Administrator\Desktop\demo>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

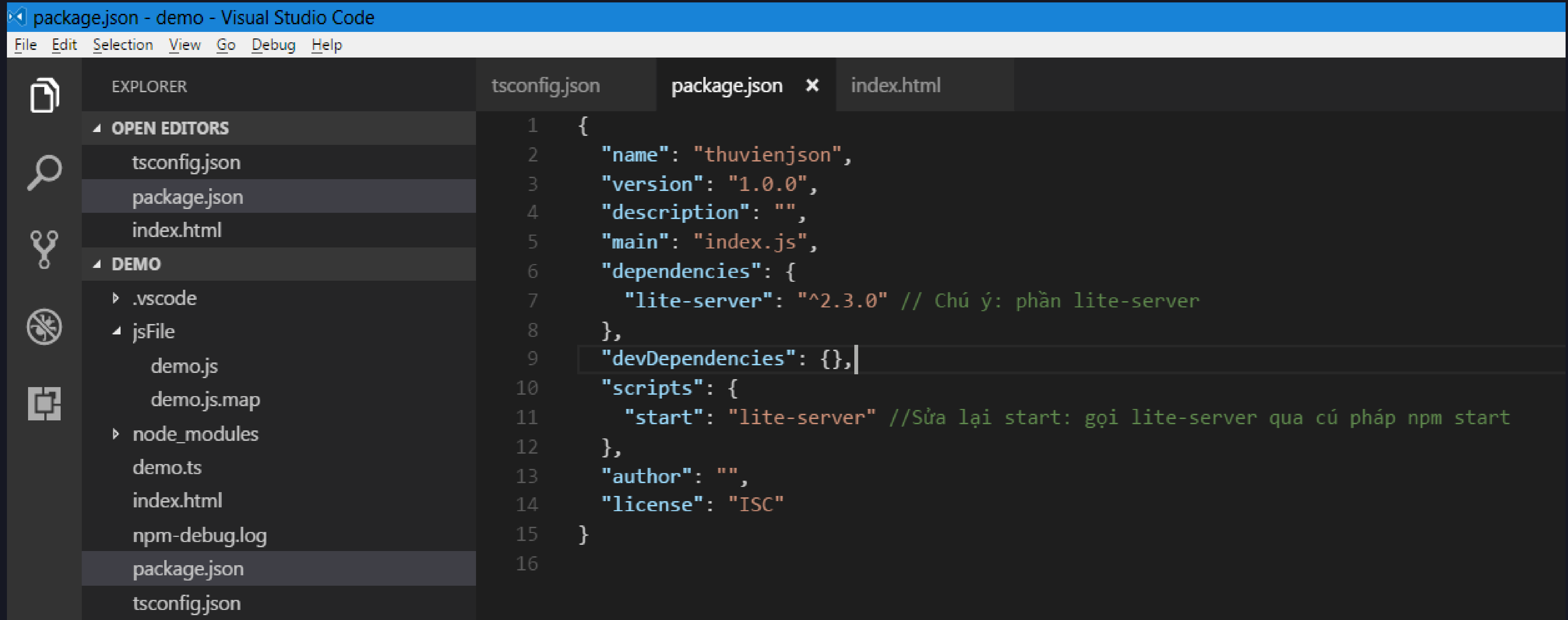
See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (demo) thuvienjson
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\Administrator\Desktop\demo\package.json:
```

**phần thông tin**

# Chỉnh sửa các node để gọi lite-server



```
package.json - demo - Visual Studio Code
File Edit Selection View Go Debug Help

EXPLORER
└─ OPEN EDITORS
    tsconfig.json
    package.json
    index.html
└─ DEMO
    └─ .vscode
    └─ jsFile
        demo.js
        demo.js.map
    └─ node_modules
        demo.ts
        index.html
        npm-debug.log
        package.json
        tsconfig.json

tsconfig.json
package.json x index.html

1 {
2   "name": "thuvienjson",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "dependencies": {
7     "lite-server": "^2.3.0" // Chú ý: phần lite-server
8   },
9   "devDependencies": {},
10  "scripts": {
11    "start": "lite-server" //Sửa lại start: gọi lite-server qua cú pháp npm start
12  },
13  "author": "",
14  "license": "ISC"
15 }
16
```

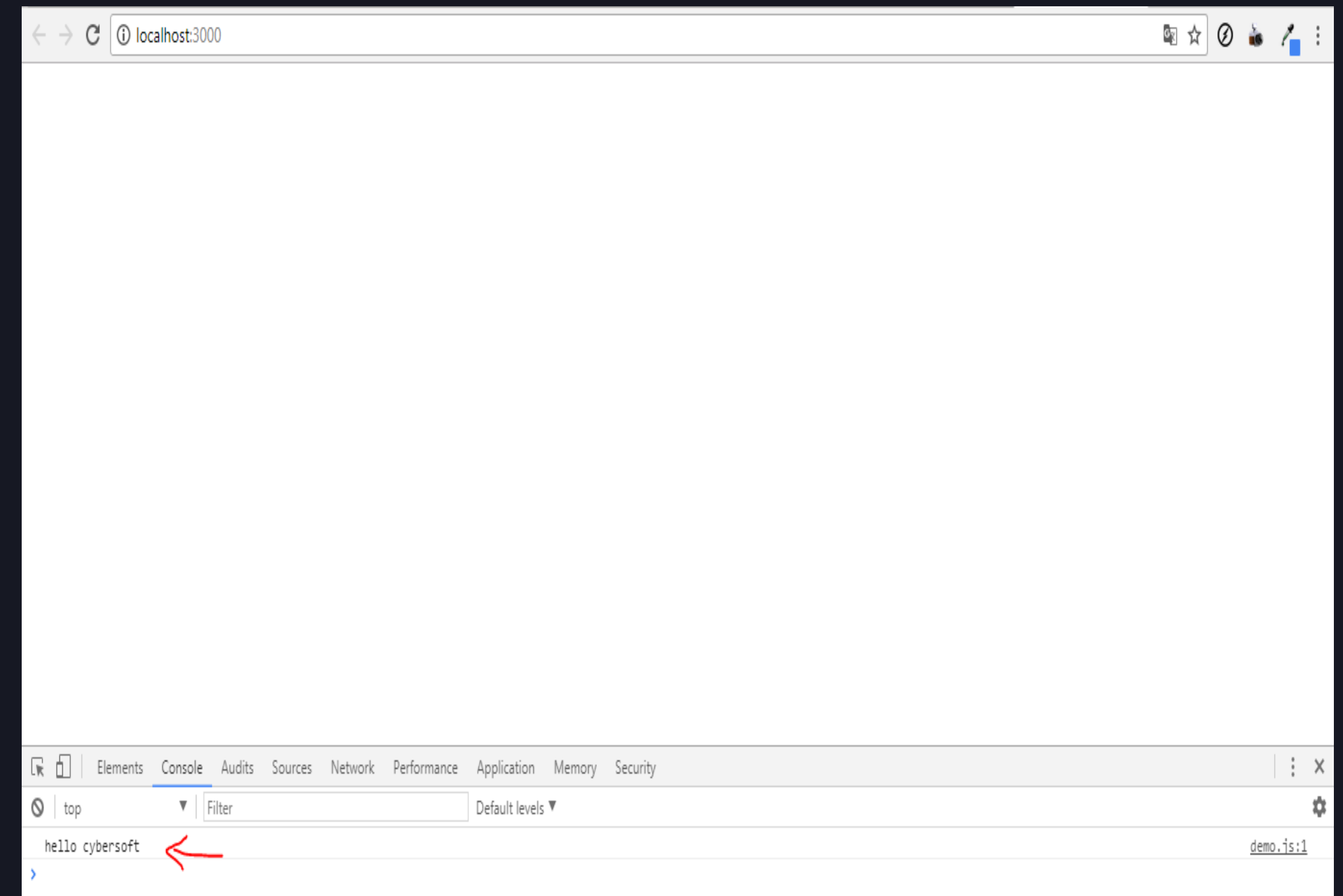


## Cú pháp khởi động lite-server: **npm start**

```
lite-server
Is this ok? (yes) yes
C:\Users\Administrator\Desktop\demo>npm start

> thuvienjson@1.0.0 start C:\Users\Administrator\Desktop\demo
> lite-server

Did not detect a `bs-config.json` or `bs-config.js` override file. Using lite-se
rver defaults...
** browser-sync config **
{ injectChanges: false,
  files: [ './**/*.html,css,js' ],
  watchOptions: { ignored: 'node_modules' },
  server: { baseDir: '.', middleware: [ [Function], [Function] ] } }
[Browsersync] Access URLs:
-----
    Local: http://localhost:3000
  External: http://192.168.1.5:3000
-----
    UI: http://localhost:3001
  UI External: http://192.168.1.5:3001
-----
[Browsersync] Serving files from: ./
[Browsersync] Watching files...
17.08.19 03:17:41 304 GET /index.html
```



Lưu ý: Phần đầu setup môi trường nên hơi phức tạp cần thực hiện đúng trình tự.

# 1. Một số khái niệm về typescript

1. Cơ chế khai báo biến
2. Các kiểu dữ liệu
3. Một số khái niệm
4. Truyền tham số
5. Truy xuất phần tử từ mảng, Object
6. String template
7. Duyệt Object
8. Định nghĩa class
9. Kế thừa class
10. Interfaces

# 1. Cơ chế khai báo biến

## 1.1 Cơ chế khai báo biến Hoisting của javascript

- Ta có thể khai báo biến trước hoặc sau khi sử dụng biến đó. Khi biên dịch typescript sẽ đẩy tất cả các biến đó lên đầu khai báo.

TS TypeScriptDemo.ts

```
1  var title = "cybersoft";
2
3  function print()
4  {
5      console.log(title);
6      var title = "cybersoft.edu.vn";
7  }
8
9  print();
```

✓ Ta thấy biến **str** ta có thể khai báo trước khi gán giá trị.

Tương tự biến **fullName** ta lại gán giá trị trước khi sử dụng.

TS TypeScriptDemo.ts

JS TypeScriptDemo.js

```
1  var title = "cybersoft";
2  function print() {
3      var title;
4      console.log(title);
5      title = "cybersoft.edu.vn";
6  }
7  print();
8  //# sourceMappingURL=TypeScriptDemo.js.map
```

# 1. Cơ chế khai báo biến

## 1.2 Khai báo với từ khóa let

Sử dụng `var`:

```
function foo() {  
  var x = 10;  
  if (true) {  
    var x = 20; // x ở đây cũng là x ở trên  
    console.log(x); // in ra 20  
  }  
  console.log(x); // vẫn là 20  
}
```

Sử dụng `let`:

```
function foo() {  
  let x = 10;  
  if (true) {  
    let x = 20; // x này là x khác rồi đấy  
    console.log(x); // in ra 20  
  }  
  console.log(x); // in ra 10  
}
```

# 1. Cơ chế khai báo biến

## 1.2 Khai báo với từ khóa let

**var** có hiệu lực đến hàm gần nhất, trong khi **let** có hiệu lực đến dấu đóng mở ngoặc gần nhất (thường sẽ nhỏ hơn phạm vi của hàm gần nhất). Thường dùng trong cấu trúc vòng lặp for, hoặc cấu trúc điều khiển if else.

```
var arr = [];  
for (var i=0; i<3; i++){  
    arr[i] = function(){  
        console.log(i);  
    }  
};  
arr[0](); // 3  
arr[1](); // 3  
arr[2](); // 3
```

Ở trên là đoạn code của ES5, biến i là biến chung trong cả function, vì vậy ở thời điểm gọi arr[0>(), arr[1](), arr[2]() thì i đã nhận giá trị 3, dẫn đến mọi output đều trở thành 3.

```
let arr = [];  
for(let i=0; i<3; i++) {  
    arr[i] = function() {  
        console.log(i);  
    }  
};  
arr[0](); //0  
arr[1](); //1  
arr[2](); //2
```

Chúng ta sẽ thử với let của ES6 Biến i chỉ nhận giá trị trong vòng for, vì vậy sẽ lần lượt mang giá trị 0,1,2 với 3 lần gọi ở cuối cùng.

# 1. Cơ chế khai báo biến

## 1.2 Khai báo với từ khóa let

```
// Đếm thời gian với var và let
for (var i = 0; i < 5; i++) {
  setTimeout(function(){
    console.log('Đếm: ', i);
  }, 1000);
}

for (let i = 0; i < 5; i++) {
  setTimeout(function(){
    console.log('Đếm ', i);
  }, 1000);
}

//Sau này học đến jquery ta rất hay gặp callback function
//$("#selector").Click(callback);
```

### Khi nào, dùng gì?

Lưu ý là chỉ khi làm việc với ES6 nhé:

- Không dùng `var` trong bất kì mọi trường hợp
- Thay vào đó thì dùng `let`
- Dùng `const` khi cần định nghĩa một hằng số

# 1. Cơ chế khai báo biến

## 1.3 Hằng số

Hằng số tương tự như biến nhưng mang một số tính chất sau:

- + Không hỗ trợ hoisting
- + Bắt buộc gán giá trị lúc khai báo
- + Không thể gán lại giá trị (readonly)
- + Đối với hằng số là object không thể new object nhưng có thể gán được các giá trị thuộc tính cho nó.

const.js	const.ts	x
1	//Khai báo hằng số	
2	const soPi = 3.14;	
3	<u>soPi</u> = 5;	
4		
5	const obJson = {name: 'Peter'};	
6	obJson.name = 'Peter1';	
7		
8	<u>obJson</u> = {name: 'abc'};	

Trình biên dịch sẽ báo lỗi nếu ta cố tình gán lại giá trị từ hằng số. Hoặc khởi tạo 1 object mới



## 2. Các kiểu dữ liệu trong typescript

### □ Các kiểu dữ liệu trong typescript

- Kiểu dữ liệu trong typescript được sử dụng để khai báo tường minh một biến.
- Một số kiểu dữ liệu trong Typescript:
  - **Boolean**: Kiểu dữ liệu mang một trong 2 giá trị là **true** hoặc **false**
  - **Number**: Kiểu dữ liệu mang giá trị là số
  - **String**: Kiểu dữ liệu mang giá trị là một chuỗi
  - **Array**: Kiểu dữ liệu mang giá trị là một mảng
  - **Tuple**: Kiểu dữ liệu mang giá trị là một mảng hỗn hợp
  - **Enum**: Kiểu dữ liệu mang các giá liệt kê
  - **Any**: Kiểu dữ liệu bất kỳ (khai báo không tường minh, có thể là number, string...)
  - **Void**: Kiểu dữ liệu giống như 1 hàm không trả về giá trị nào cả
  - **Null**: Kiểu dữ liệu mang giá trị null
  - **Undefined**: Kiểu dữ liệu mang giá trị undefined

## ❑ Boolean

- Kiểu dữ liệu mang một trong 2 giá trị là true hoặc false
  - Khai báo biến

```
let isDone: boolean = false;
```

## ❏ Number

### ➤ Kiểu dữ liệu mang kiểu giá trị là số

- Khai báo biến

```
let decimal: number = 6;  
let hex: number = 0xf00d;  
let binary: number = 0b1010;  
let octal: number = 0o744;
```

## ❑ String

- Kiểu dữ liệu mang kiểu giá trị là chuỗi
  - Khai báo biến

```
let color: string = "blue";  
color = 'red';
```

## □ Array

### ➤ Kiểu dữ liệu mang giá trị là một mảng

#### ▪ Khai báo mảng

##### ✓ Cách 1

```
let list: number[] = [1, 2, 3];
```

##### ✓ Cách 2

```
let list: Array<number> = [1, 2, 3];
```

## □ Tuple

### ➤ Kiểu dữ liệu mang giá trị là một mảng hỗn hợp

- Khai báo tuple

```
// Declare a tuple type
let x: [string, number];
// Initialize it
x = ["hello", 10]; // OK
// Initialize it incorrectly
x = [10, "hello"]; // Error
```

## □ Enum

- Kiểu dữ liệu mang các giá liệt kê. Thường dùng để gán các giá trị tham số với một tên mang ý nghĩa tường minh hơn. Tương tự Enum trong C#

- Khai báo enum

```
enum Color {Red = 1, Green = 2, Blue = 4}  
let c: Color = Color.Green;
```



## □ Any

➤ Đại diện cho một kiểu dữ liệu bất kỳ. Có thể là số hoặc chuỗi, object json,... Tương tự dynamic trong C#.

- Khai báo với kiểu any

```
let notSure: any = 4;  
notSure = "maybe a string instead";  
notSure = false; // okay, definitely a boolean
```

```
let notSure: any = 4;  
notSure.ifItExists(); // okay, ifItExists might exist at runtime  
notSure.toFixed(); // okay, toFixed exists (but the compiler doesn't check)  
  
let prettySure: Object = 4;  
prettySure.toFixed(); // Error: Property 'toFixed' doesn't exist on type 'Object'.
```

## ❑ Null And Undefined

```
// Not much else we can assign to these variables!  
let u: undefined = undefined;  
let n: null = null;
```

### 3. Một số khái niệm trong typescript

- **Arrow function:** Là một cách viết function dạng viết tắt function được thêm vào từ ES6.
- **Tham số mặc định:** Kiểu dữ liệu mang giá trị là một chuỗi

## ❑ Arrow function

➤ Là một cách viết function dạng viết tắt function được thêm vào từ ES6.

### ■ Function theo cách viết thông thường

```
TS arrowfunction.ts ✕  
1  function hello(name)  
2  {  
3    console.log("Xin chào "+ name );  
4  }  
5  hello("cybersoft");  
6  
7
```

### ■ Function theo arrow function

```
TS arrowfunction.ts ✕  
1  var hello = (name) => {  
2    console.log("Chào "+ name);  
3  }  
4  hello("cybersoft");  
5
```

# ❑ Arrow function

So sánh hai các trên thì rõ ràng cách thông thường sẽ đơn giản hơn rất nhiều. Tuy nhiên đối với trường hợp thân hàm chỉ có 1 lệnh, chúng ta có thể viết theo kiểu ví dụ phía dưới.

## ■ Function theo arrow function

```
1 | var hello = (name, message) => console.log("Chào " + name + ", bạn là " + message);
```

Ta có thể bỏ đi cặp dấu {}, Điều xảy ra khi thân hàm chỉ là 1 câu lệnh. Ngoài ra ta còn đơn giản đi được từ khóa **function**. Vì vậy arrow function thường được sử dụng khi hàm **return** về 1 giá trị nào đó đơn giản với một dòng xử lý.

## ■ Ví dụ khác:

```
//function
var calculatorSalary = function(bonus:number){
    return 10000 + bonus;
};
```

```
//Arrow function
var calculatorArrow = (bonus:number) => 10000+bonus;
|
```

## ❑ Giá trị mặc định của tham số function

Ta có thể gán giá trị mặc định cho tham số là một biểu thức như ví dụ bên dưới

```
//arrow function có giá trị trả về là 10
var getparent = () => 10;
var getBonus = function(value=10+getparent(),num=value*0.1){
    //tham số sau có thể lấy giá trị tham số trước
    //tham số trước ko lấy được giá trị tham số sau
    console.log(num);
}
getBonus();
```

Ở tham số phía sau có thể lấy giá trị của tham số phía trước. Nhưng tham số phía trước không thể lấy giá trị tham số phía sau.

## 4. Truyền tham số

### ❖ Resparameter (Truyền nhiều tham số)

Ngoài cách truyền tham số thông thường như javascript typescript còn hỗ trợ chúng ta truyền tham số không giới hạn tham số truyền vào.

```
let displayColors = function(...colors:string[]){  
    //Duyệt mảng  
    for(let i in colors)  
    {  
        console.log(colors);  
    }  
};  
  
displayColors('Red');  
displayColors('Red', 'Green');  
displayColors('Red', 'Green', 'Blue');
```

Cú pháp:

`function (...[thamso]:[kiểu dữ liệu])`

Đối số truyền vào là các biến có dùng kiểu dữ liệu với [kiểu dữ liệu] của tham số



## ❖ Spread operator (Truyền tham số là mảng)

Ngược lại để truyền tham số là một mảng ta sử dụng cú pháp như ví dụ bên dưới

```
let displayColorsSpread = function(...colors){  
  for(let i in colors)  
  {  
    console.log(colors[i]);  
  }  
};
```

```
let colors = ['Red', 'Green', 'Blue'];
```

```
displayColorsSpread(...colors);
```

Cú pháp:

`function (...[thamso])`

Đối số truyền vào là một mảng.  
Ký hiệu tham số là mảng trong  
typescript là `...[thamso]`

## 5. Truy xuất phần tử từ mảng, Object

### ❖ Destructuring Array (Bóc tách phần tử trong mảng)

Destructuring Assignments đơn giản chỉ là cách tách các phần tử của Array hoặc Object thành nhiều biến chỉ bằng một đoạn code duy nhất.

```
// Array
let date = [10, 03, 2016]

// Chuyển ba giá trị vào ba biến d, m, y
let [d, m, y] = date;

// In kết quả
console.log("Day: " + d);    // Day: 10
console.log("Month: " + m);  // Month: 03
console.log("Year: " + y);   // Year : 2016
```

Nếu muốn lấy phần tử đầu tiên.

```
let [item1,...item2]= date; //Lấy phần tử đầu tiên
console.log(item1);
```

Nếu muốn lấy phần tử ở vị trí thứ 3.

```
let [,,y] = date //Lấy giá trị ở vị trí thứ 3
console.log(y);
|
```

## ❖ Destructuring Object (Bóc tách thuộc tính của đối tượng)

Tương tự mảng. Đối với object có 2 cách để bóc tách giá trị thuộc tính của một object

### Cách 1:

```
let emp={
  fname:"Peter",
  lname:"Cech",
  level:2
};

let {fname,lname,level}=emp;
console.log(fname);
console.log(lname);
console.log(level);
```

### Cách 2: Đặt thêm bí danh

```
let emp={
  fname:"Peter",
  lname:"Cech",
  level:2
};

//Tương tự mảng ta có object
let {fname:f,lname:l,level:lv} = emp;

//Lấy ra các thuộc tính của object
console.log(f);
console.log(l);
console.log(lv);
```

## 6. String template

String template được bắt đầu bằng dấu ``. Cũng giống như chuỗi nhưng string template hỗ trợ chúng ta đưa tham số vào dễ dàng hơn. Đối với một cấu trúc chuỗi cần đưa vào nhiều tham số thì ta định nghĩa bằng string template code sẽ rõ ràng và dễ hiểu hơn.

### Ví dụ:

```
var schoolName="cybersoft";  
var hello = `welcome to ${schoolName} academy!`;   
console.log(hello);
```

Qua ví dụ trên ta có thể thấy để thêm tham số vào **string template** ta dùng cú pháp **\${[Tên biến]}**.  
Lưu ý: Biến của chúng ta sử dụng

## 7. Duyệt object

Để duyệt một object là một mảng hay chuỗi ta có thể chọn một trong 2 cách duyệt như sau

### Duyệt for in

```
var colorName = ['Red', 'Green', 'Blue', 'Yellow', 'Puple'];  
//Duyệt bằng for in  
//colorName: Đối tượng mảng, index vị trí phần tử trong mảng  
for (let index in colorName)  
{  
    console.log(colorName[index]);  
}
```

### Duyệt for of (được sử dụng nhiều)

```
var colorName = ['Red', 'Green', 'Blue', 'Yellow', 'Puple'];  
//Duyệt bằng for of  
//item: là một đối tượng thuộc mảng color ở đây đối tượng kiểu chuỗi  
for (let item of colorName)  
{  
    console.log(item);  
}
```

```
var str = "CYBERSOFT"  
for (let item of str)  
{  
    console.log(item);  
}
```

## 8. Định nghĩa class

Class thực chất là function nhưng không hỗ trợ hoisting, có nghĩa là new 1 lần trên 1 biến không làm dc 2 lần (Prototype trong JavaScript).

### ■ Định nghĩa class

```
class SinhVien //Tương tự prototype
{
    //Thuộc tính
    public hoten:string;
    public lop:string;
    constructor(HoTen:string,Lop:string)
    {
        this.hoten=HoTen;
        this.lop=Lop;
    }
    //Phương thức
    public XuấtThongTin()
    {
        console.log(`Họ ten: ${this.hoten} Lớp: ${this.lop}`);
    }
}
//Khởi tạo 1 đối tượng (Instance)
let sv = new SinhVien("Nguyễn Nhật Quang","FrontEnd01");
sv.XuấtThongTin();
```

## 8. Định nghĩa class

### ▪ Định Phương thức khởi tạo và phương thức tĩnh

Từ khóa Static  
định nghĩa  
phương thức  
tĩnh

Gọi thực thi  
bằng tên  
Class.TênPT()

```
class Hocvien{  
    public hoten:string;  
    //Phương thức khởi tạo tương tự function(): Tham số trong ngoặc  
    constructor (hoten)  
    {  
        this.hoten=hoten;  
    }  
    static GioiThieu()  
    {  
        console.log('Xin chào mọi người ^^!');  
    }  
}  
let p = new Hocvien("học viên 1");  
//Gọi trực tiếp mà không cần thông qua đối tượng  
Hocvien.GioiThieu();
```

Định nghĩa hàm tạo  
bằng từ khóa  
constructor([thamso])

Khởi tạo đối tượng.

## 9. Kế thừa class

Tương tự javascript typescript cũng có kế thừa.

### ■ Class cha

```
class NhanVien{
    public hoten:string;
    constructor(hoten){
        this.hoten = hoten;
    }
    public TinhLuong()
    {
        return 1000;
    }
}
```

### ■ Class con

```
class GiamDoc extends NhanVien{
    public quyen:string;
    constructor(name,Quyen){
        super(name);
        this.quyen = Quyen;
    }
    TinhLuong()
    {
        return super.TinhLuong() + 2000;
    }
}
```

- Đối với hàm tạo nếu class con muốn kế thừa bắt buộc phải có từ khóa super([thamso\_HamCha])
- Đối với các phương thức thì tùy, ta có thể dùng từ khóa super để kế thừa xử lý từ hàm cha hoặc không cần.



## 9. Kế thừa class

- Class cha

```
class NhanVien{  
    public hoten:string;  
    constructor(hoten){  
        this.hoten = hoten;  
    }  
    public TinhLuong()  
    {  
        return 1000;  
    }  
}
```

- Khởi tạo và sử dụng:

```
var giamDoc = new GiamDoc('Quang','Làm chủ');  
console.log(`Họ tên:${giamDoc.hoten} - Quyền: ${giamDoc.quyen} - Lương: ${giamDoc.TinhLuong()} `);
```

Từ khóa **extends**  
để kế thừa

- Class con

```
class GiamDoc extends NhanVien{  
    public quyen:string;  
    constructor(name,Quyen){  
        super(name);  
        this.quyen = Quyen;  
    }  
    TinhLuong()  
    {  
        return super.TinhLuong() + 2000;  
    }  
}
```


# 10. Interfaces

Interfaces là lớp trừu tượng định nghĩa các phương thức nhưng **không xử lý**. Mà để cho các class con kế thừa **bắt buộc** phải xây dựng phương thức xử lý cho nó.

## ■ interface

```
interface Nhan_Vien{
    ho:string;
    ten:string;
    tuoi?:number; //co the null
    ThucHienCongViec();
}
```

## ■ Class kế thừa

 Từ khóa **implements** để kế thừa

```
class NhanVien_ThietKe implements Nhan_Vien {
    ho:string;
    ten:string;
    tuoi:number;
    ThucHienCongViec(){
        console.log("Lên ý tưởng quảng cáo");
        console.log("Thiết kế banner");
    }
}
```

## ■ Class kế thừa

```
class GiamDoc_ThietKe implements Nhan_Vien {
    ho:string;
    ten:string;
    tuoi:number;
    ThucHienCongViec(){
        return "Giao công việc cho Quang";
    }
}
```

# 11. DOM trong typescript

```
//Javascript  
var manv = document.getElementById('manv');
```

```
//TypeScript  
var manv:number = parseInt((<HTMLInputElement>document.getElementById('manv')).value);
```

Typescript có thêm phần định nghĩa kiểu dữ liệu nên cú pháp sẽ dài hơn

## 12. Sự kiện trong TypeScript

Không sử dụng onclick trên thẻ nữa mà ta sẽ gán sự kiện thông qua cách này trong

**Javascript**

```
//Thêm 1 sự kiện vào nút button có id là btnThemNhanVien trong javascript  
document.getElementById("btnThemNhanVien").addEventListener("click", ThemNV);  
//ThemNV: tên function ThemNV
```

**Typescript**

```
//Tương tự javascript nhưng có phần phía trước để định nghĩa kiểu dữ liệu  
(<HTMLElement>document.getElementById("btnThemNhanVien")).addEventListener("click", ThemNV);
```

# Bài tập thực hành

## Thông Tin Nhân Viên

Mã nhân viên:

3

Họ tên:

Nguyễn Văn C

Lương CB:

1500

Loại nv: ☒ Sếp ☐ Quản Lý ☐ Thường

Thêm Nhân Viên

Xuất thông tin NV

## Danh sách sinh viên

Mã nhân viên	Họ tên	Lương CB	Loại nhân viên	Lương * hệ số
1	Nguyễn Văn A	500	Nhân viên	500
2	Nguyễn Văn B	1000	Quản Lý	1500
3	Nguyễn Văn C	1500	Sếp	4500

Xây dựng lớp đối tượng  
bằng TypeScript

Viết ứng dụng quản lý nhân  
viên như hình chụp.

- Nếu là thường thì lương CB  
\*1
- Nếu là quản lý lương CB  
\*1.5
- Nếu là sếp lương cơ bản \*3  
=> Sử dụng typescript +  
webpack để build ứng dụng.

