

Theory 5.2

TypeScript Fundamentals

WEEK 02

Nội dung chính

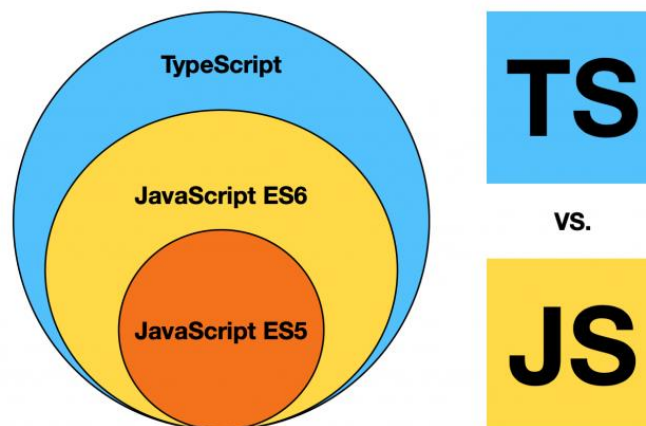
- ❑ TỔNG QUAN VỀ TYPESCRIPT 03
- ❑ TẠI SAO NÊN SỬ DỤNG TYPESCRIPT 04
- ❑ CÀI ĐẶT DỰ ÁN TYPESCRIPT 06
- ❑ CÚ PHÁP TYPESCRIPT CƠ BẢN 12
- ❑ CÚ PHÁP TYPESCRIPT OOP 17

AGENDA



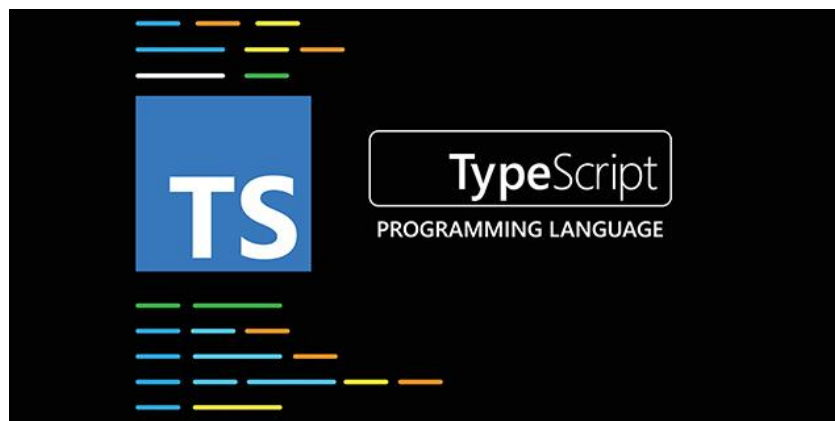
Tổng quan về TypeScript

- ❑ **TypeScript** (TS) là một ngôn ngữ lập trình được phát triển bởi Microsoft.
- ❑ **TypeScript** được coi là một phiên bản nâng cao của JavaScript bởi việc bổ sung tùy chọn kiểu tĩnh và lớp hướng đối tượng.
- ❑ **TypeScript** có thể được sử dụng để phát triển các ứng dụng chạy ở client-side (Angular2) và server-side (NodeJS).



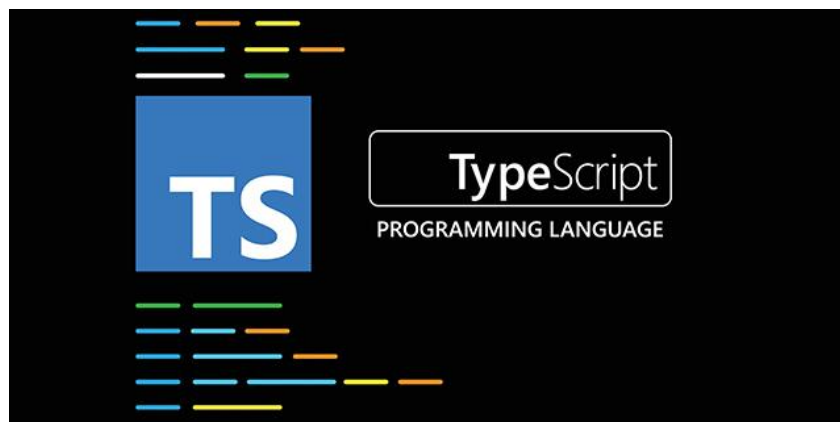
Tại sao nên sử dụng TypeScript?

- ❑ **Dễ phát triển dự án lớn:** Với việc sử dụng các kỹ thuật mới nhất và lập trình hướng đối tượng nên TypeScript giúp phát triển các dự án lớn một cách dễ dàng.
- ❑ **Nhiều Framework lựa chọn:** Hiện nay các JavaScript Framework đã dần khuyến khích nên sử dụng TypeScript để phát triển, ví dụ Angular 2.0 và Ionic 2.0.
- ❑ **Hỗ trợ các tính năng của JS phiên bản mới nhất:** TypeScript luôn đảm bảo việc sử dụng đầy đủ các kỹ thuật mới nhất của JavaScript (Ví dụ: ECMAScript 2016).



Tại sao nên sử dụng TypeScript?

- ❑ **Mã nguồn mở:** TypeScript là một mã nguồn mở nên bạn hoàn toàn có thể sử dụng mà không mất phí, bên cạnh đó còn được cộng đồng hỗ trợ
- ❑ **TypeScript là JavaScript:** Bản chất của TypeScript là biên dịch tạo ra các đoạn mã javascript nên bạn có thể chạy bất kì ở đâu miễn ở đó có hỗ trợ biên dịch Javascript. Ngoài ra, có thể sử dụng trộn lẫn cú pháp của Javascript vào bên trong TypeScript (giúp các lập trình viên dễ tiếp cận TypeScript hơn)



Cài đặt dự án TypeScript

❑ Cài đặt Node và trình biên dịch TypeScript

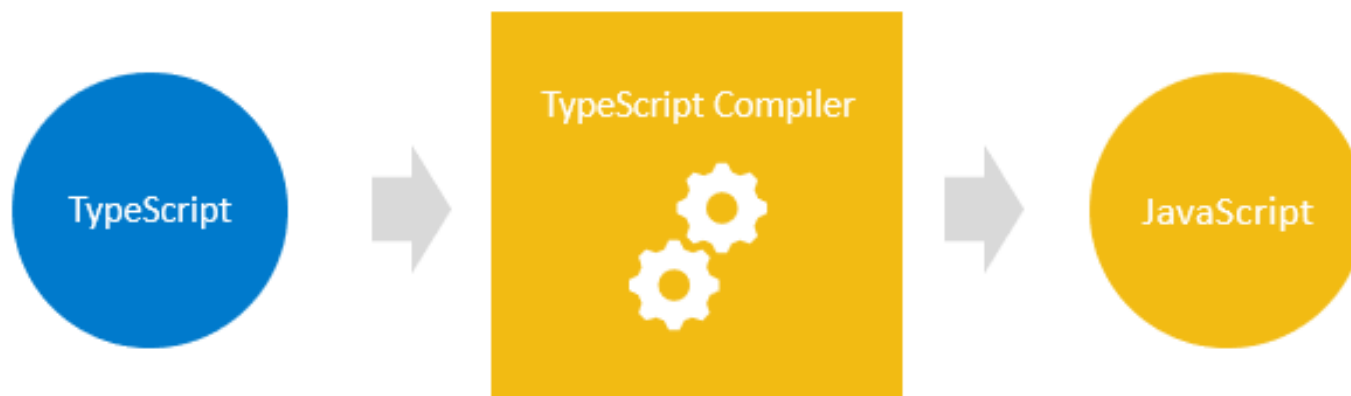
- **Node:** Kiểm tra và đảm bảo bạn đã cài đặt node.js trước đó.

- **TypeScript compiler:**

 - **Bước 1:** Chạy câu lệnh sau để cài đặt trình biên dịch TypeScript:

 - `npm i typescript`

 - **Bước 2:** Chạy câu lệnh sau để kiểm tra phiên bản TypeScript: `tsc -v`



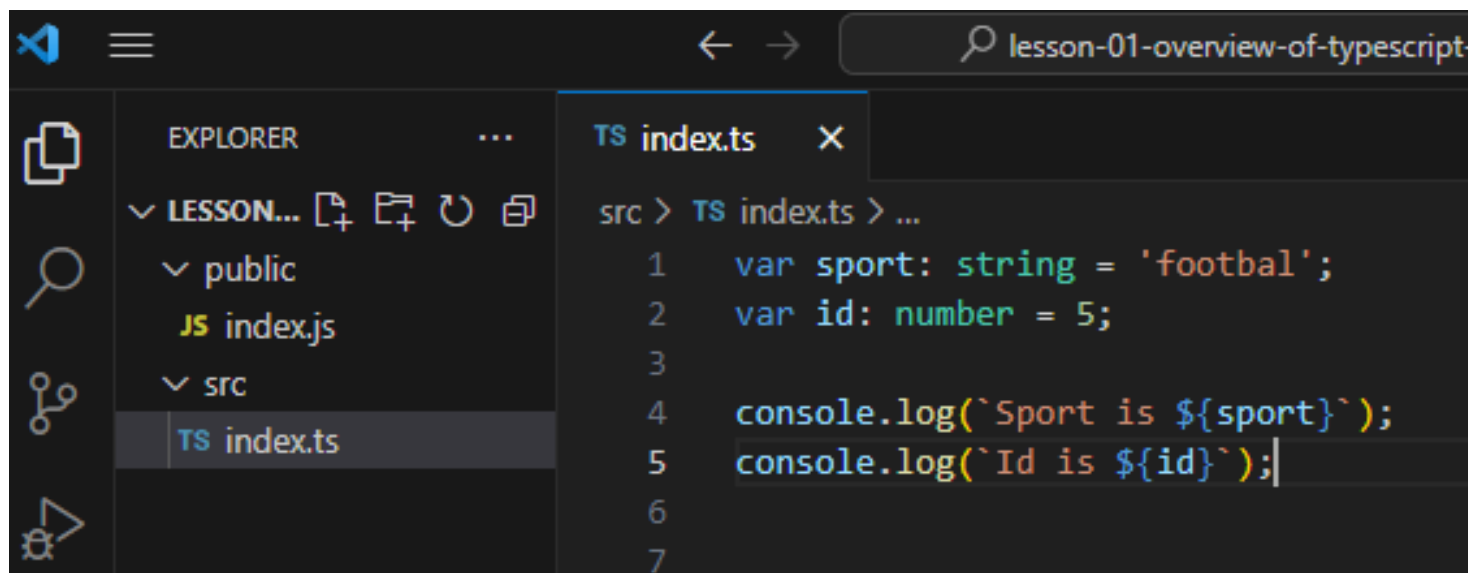
Cài đặt dự án TypeScript

❑ Cài đặt Node và trình biên dịch TypeScript

- **Node:** Kiểm tra và đảm bảo bạn đã cài đặt node.js trước đó.

- **TypeScript compiler:**

 - **Bước 3:** Viết một file TypeScript với nội dung bất kỳ



```
src > TS index.ts > ...
1  var sport: string = 'football';
2  var id: number = 5;
3
4  console.log(`Sport is ${sport}`);
5  console.log(`Id is ${id}`);
6
7
```

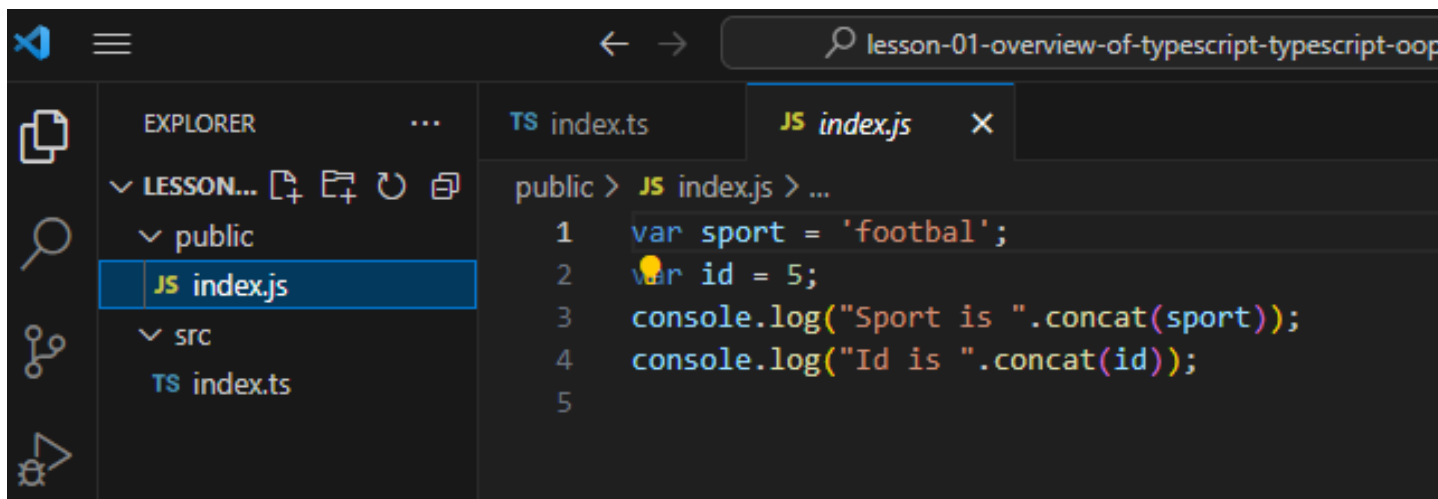
Cài đặt dự án TypeScript

❑ Cài đặt Node và trình biên dịch TypeScript

- **Node:** Kiểm tra và đảm bảo bạn đã cài đặt node.js trước đó.

- **TypeScript compiler:**

- **Bước 4:** Biên dịch file **TypeScript** thành file **JavaScript** với đường dẫn file output chi tiết, ta sử dụng câu lệnh sau: `tsc --outDir public ./src/index.ts -w`



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project structure with a folder named 'public' containing a file 'index.js'. The main editor area shows the content of 'index.js', which contains the following JavaScript code:

```
1 var sport = 'footbal';
2 var id = 5;
3 console.log("Sport is ".concat(sport));
4 console.log("Id is ".concat(id));
5
```

The console output at the bottom shows the command being executed: `public > JS index.js > ...`, followed by the execution of the code.

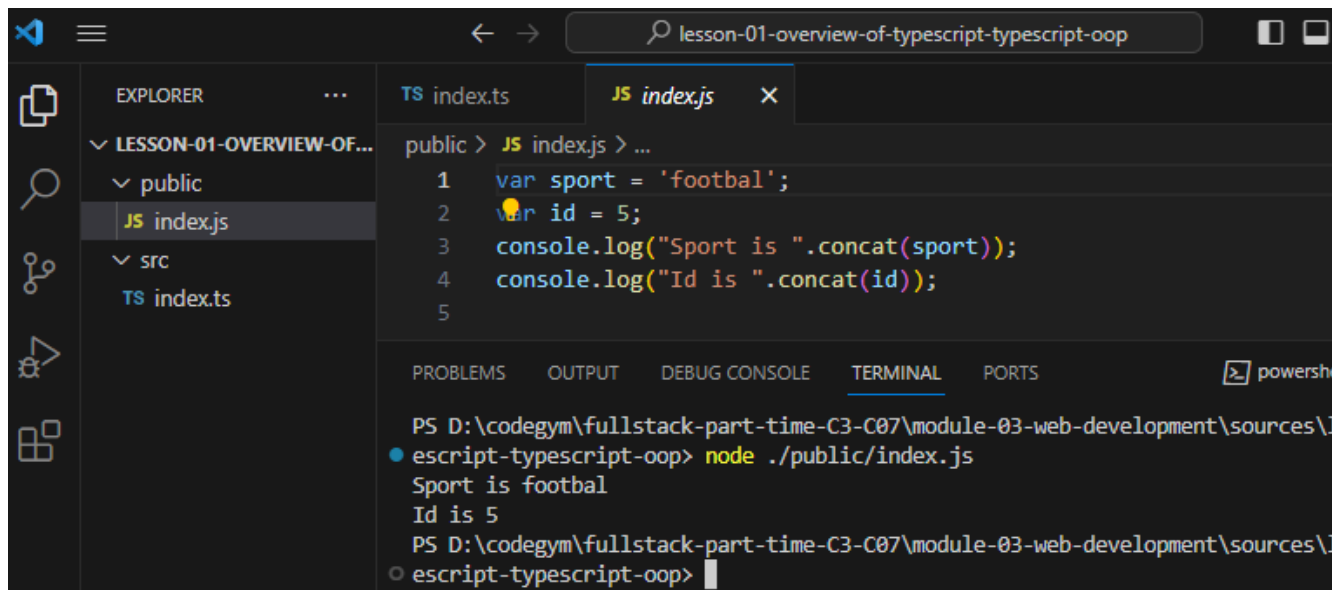
Cài đặt dự án TypeScript

❑ Cài đặt Node và trình biên dịch TypeScript

- **Node:** Kiểm tra và đảm bảo bạn đã cài đặt node.js trước đó.

- **TypeScript compiler:**

- **Bước 5:** Thực thi code trong file index.js như sau: `node ./public/index.js`



```
lesson-01-overview-of-typescript-typescript-oop
```

EXPLORER

- LESSON-01-OVERVIEW-OF...
- public
 - index.js
- src
 - index.ts

public > JS index.js > ...

```
1 var sport = 'football';
2 var id = 5;
3 console.log('Sport is '.concat(sport));
4 console.log('Id is '.concat(id));
5
```

TERMINAL

```
PS D:\codegym\fullstack-part-time-C3-C07\module-03-web-development\sources\l
escrpt-typescript-oop> node ./public/index.js
Sport is football
Id is 5
PS D:\codegym\fullstack-part-time-C3-C07\module-03-web-development\sources\l
escrpt-typescript-oop>
```

Cài đặt dự án TypeScript

❑ Cài đặt Node và trình biên dịch TypeScript

- **TypeScript config file:** Kiểm tra và đảm bảo bạn đã cài đặt node.js trước đó.
 - Tập cấu hình ts phải nằm trong thư mục gốc của dự án của bạn
 - Trong tệp này sẽ chỉ định các tệp gốc, các tùy chọn trình biên dịch và mức độ nghiêm ngặt mà chúng tôi muốn TypeScript kiểm tra dự án của mình

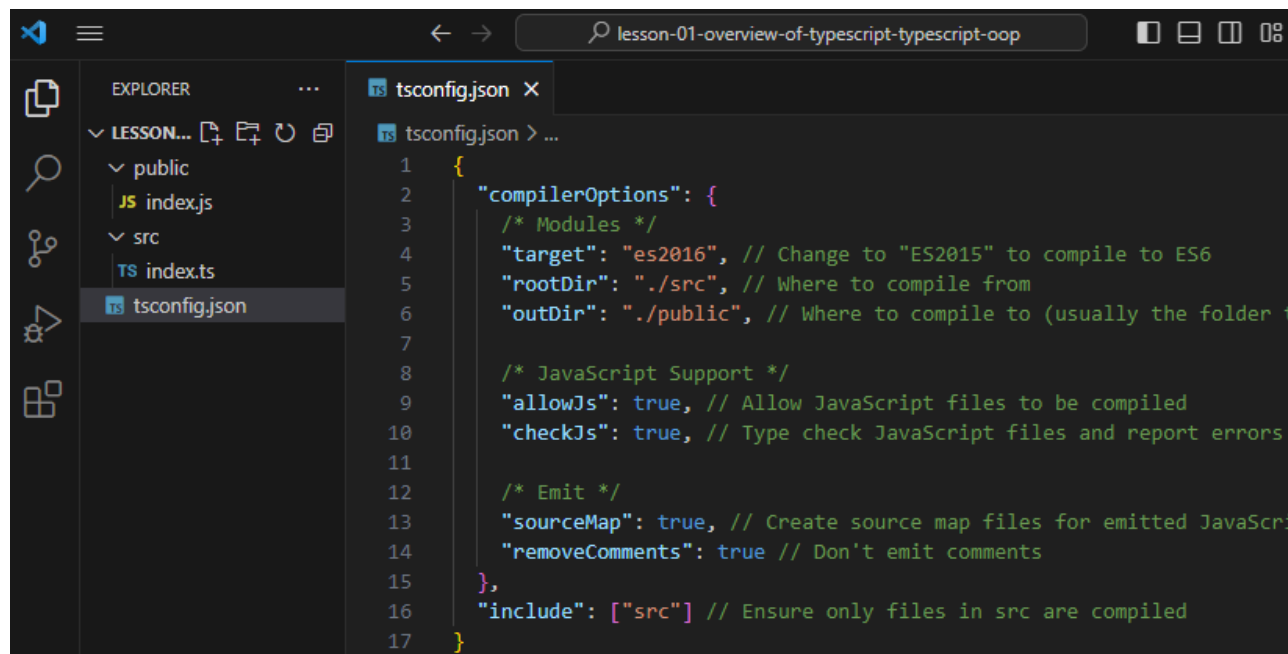


Cài đặt dự án TypeScript

❑ Cài đặt Node và trình biên dịch TypeScript

○ **TypeScript config file**: Kiểm tra và đảm bảo bạn đã cài đặt node.js trước đó.

➤ Tạo file cấu hình **tsconfig.json** ở cấp root dự án: **tsc --init**



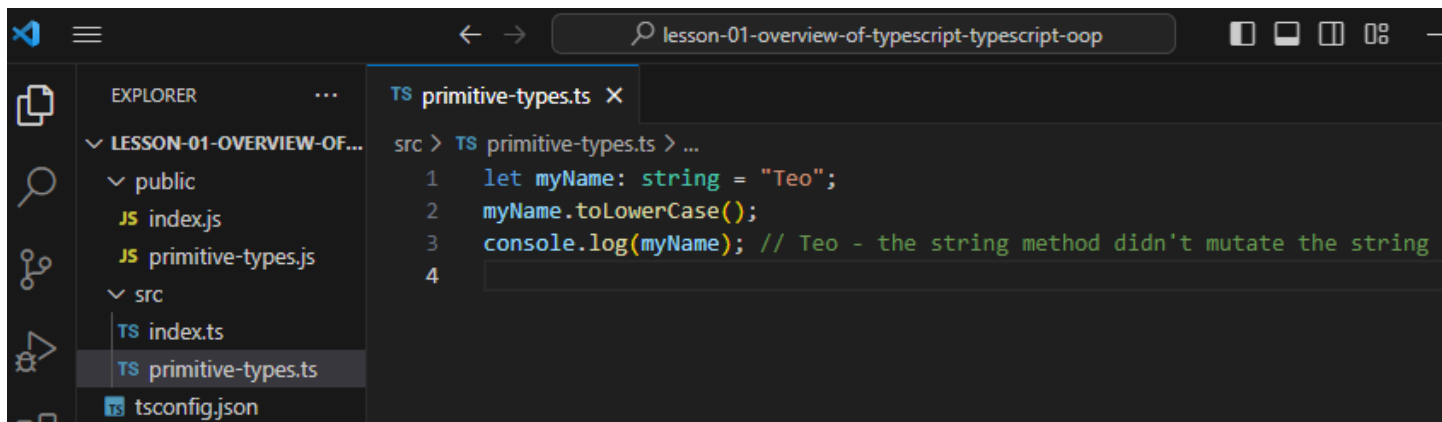
The screenshot shows a Visual Studio Code editor window with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'public' folder containing 'index.js' and a 'src' folder containing 'index.ts' and 'tsconfig.json'. The code editor displays the contents of 'tsconfig.json', which is a JSON configuration file for TypeScript. The file is named 'tsconfig.json' and is located in the root of the project. The configuration includes compiler options for target, rootDir, outDir, allowJs, checkJs, sourceMap, and removeComments, as well as an 'include' array specifying the 'src' directory.

```
1 {
2   "compilerOptions": {
3     /* Modules */
4     "target": "es2016", // Change to "ES2015" to compile to ES6
5     "rootDir": "./src", // Where to compile from
6     "outDir": "./public", // Where to compile to (usually the folder t
7
8     /* JavaScript Support */
9     "allowJs": true, // Allow JavaScript files to be compiled
10    "checkJs": true, // Type check JavaScript files and report errors
11
12    /* Emit */
13    "sourceMap": true, // Create source map files for emitted JavaScri
14    "removeComments": true // Don't emit comments
15  },
16  "include": ["src"] // Ensure only files in src are compiled
17 }
```

Cú pháp TypeScript cơ bản

❑ Kiểu dữ liệu trong TypeScript

○ **Primitive Types (Kiểu nguyên thủy)**: Trong JavaScript, có 7 kiểu: **string**, **number**, **bigint**, **boolean**, **undefined**, **null**, **symbol**. **Primitive** là **immutable** (bất biến), chúng không thể bị thay đổi, chúng ta chỉ có thể gán giá trị 1 biến cho một giá trị nguyên thủy khác. Giá trị nguyên thủy hiện tại không thể bị thay đổi như cách **object**, **array**, **function** thay đổi.

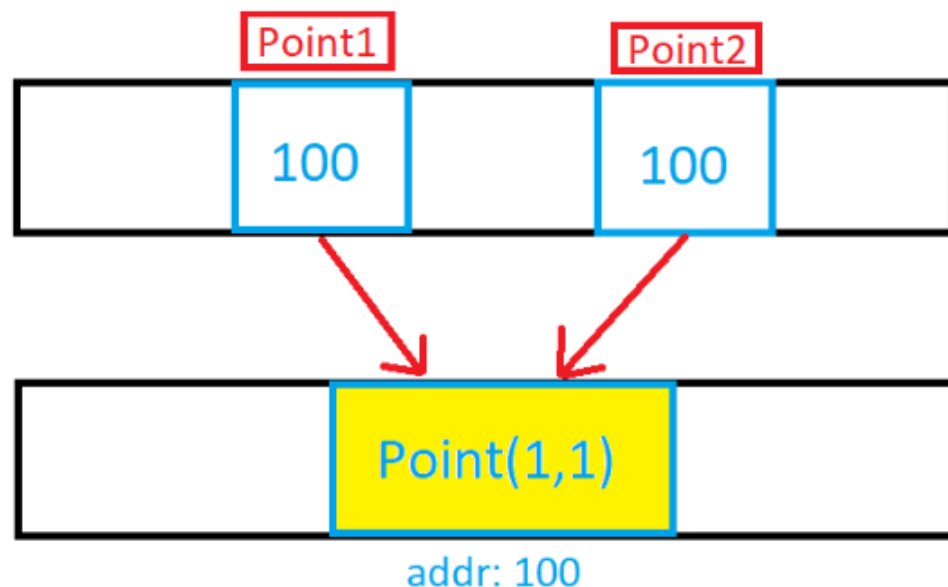


```
src > TS primitive-types.ts > ...
1 let myName: string = "Teo";
2 myName.toLowerCase();
3 console.log(myName); // Teo - the string method didn't mutate the string
4
```

Cú pháp TypeScript cơ bản

❑ Kiểu dữ liệu trong TypeScript

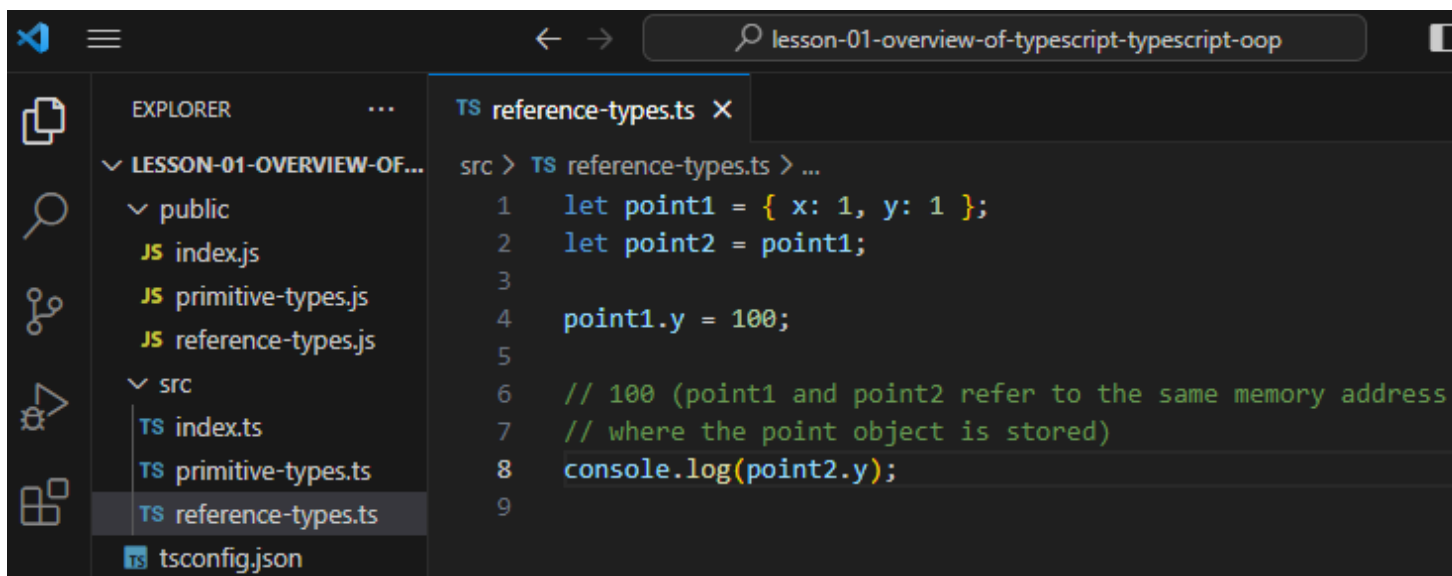
○ **Reference Types (Kiểu tham chiếu):** Trong JavaScript, các kiểm tham chiếu dùng cho đối tượng là chủ yếu. Các kiểu tham chiếu sẽ tham chiếu tới địa chỉ vùng nhớ nơi đối tượng được lưu trữ.



Cú pháp TypeScript cơ bản

❑ Kiểu dữ liệu trong TypeScript

○ **Reference Types (Kiểu tham chiếu)**: Trong JavaScript, các kiểm tham chiếu dùng cho đối tượng là chủ yếu. Các kiểu tham chiếu sẽ tham chiếu tới địa chỉ vùng nhớ nơi đối tượng được lưu trữ.



```
src > TS reference-types.ts > ...
1  let point1 = { x: 1, y: 1 };
2  let point2 = point1;
3
4  point1.y = 100;
5
6  // 100 (point1 and point2 refer to the same memory address
7  // where the point object is stored)
8  console.log(point2.y);
9
```

Cú pháp TypeScript cơ bản

❑ Mảng trong TypeScript

- Trong TypeScript, chúng ta có thể định nghĩa kiểu dữ liệu của phần tử mảng.

```
TS array.ts  X
src > TS array.ts > ...
1  let ids: number[] = [1, 2, 3, 4, 5]; // can only contain numbers
2  let names: string[] = ['Danny', 'Anna', 'Bazza']; // can only contain strings
3  let options: boolean[] = [true, false, false]; //can only contain true or false
4  let books: object[] = [
5    { name: 'Fooled by randomness', author: 'Nassim Taleb' },
6    { name: 'Sapiens', author: 'Yuval Noah Harari' },
7  ]; // can only contain objects
8  let arr: any[] = ['hello', 1, true]; // any basically reverts TypeScript back into JavaScript
9
10 ids.push(6);
11 // ids.push('7'); // ERROR: Argument of type 'string' is not assignable to parameter of type 'number'.
```

Cú pháp TypeScript cơ bản

❑ Mảng trong TypeScript

○ **Union Types**: có thể định nghĩa mảng nhiều phần tử với nhiều kiểu dữ liệu khác nhau.

```
TS array-w-union-types.ts X
src > TS array-w-union-types.ts > ...
1 let person: (string | number | boolean)[] = ["Danny", 1, true];
2 person[0] = 100;
3 // person[1] = { name: "Danny" }; // Error - person array can't contain objects
```

○ **Tuple**: là một mảng với số lượng cố định và kiểu dữ liệu đã biết trước.

```
TS tuple.ts X
src > TS tuple.ts > ...
1 let person02: [string, number, boolean] = ["Danny", 1, true];
2 // person02[0] = 100; // Error - Value at index 0 can only be a string
```


Cú pháp TypeScript OOP

❑ Đối tượng và Interface trong TypeScript

○ Đối tượng trong TypeScript phải chứa các thuộc tính có kiểu dữ liệu cụ thể.

```
TS objects X
src > TS objects > ...
1 // Declare a variable called person with a specific object type annotation
2 let person03: {
3   name: string;
4   location: string;
5   isProgrammer: boolean;
6 };
7
8 // Assign person to an object with all the necessary properties and value types
9 person03 = {
10   name: "Danny",
11   location: "UK",
12   isProgrammer: true,
13 };
14
15 // person03.isProgrammer = "Yes"; // ERROR: should be a boolean
16
17 // person03 = {
18 //   name: "John",
19 //   location: "US",
20 // };
21 // ERROR: missing the isProgrammer property
```

○ Xác định các điểm đặc trưng của 1 đối tượng, ta có thể sử dụng Interface

```
TS Person.ts X
src > interface > TS Person.ts > ...
1 interface Person {
2   name: string;
3   location: string;
4   isProgrammer: boolean;
5 }
6
7 let person1: Person = {
8   name: "Danny",
9   location: "UK",
10  isProgrammer: true,
11 };
12
13 let person2: Person = {
14   name: "Sarah",
15   location: "Germany",
16   isProgrammer: false,
17 };
```

Cú pháp TypeScript OOP

❑ Đối tượng và Interface trong TypeScript

○ Chúng ta có thể khai báo các phương thức trong đối tượng hoặc hàm theo kiểu cũ (sayHi) hoặc các hàm mũi tên ES6 (sayBye).

```
TS object-method.ts X
src > TS object-method.ts > ...
1  interface Speech {
2      sayHi(name: string): string;
3      sayBye: (name: string) => string;
4  }
5
6  let sayStuff: Speech = {
7      sayHi: function (name: string) {
8          return `Hi ${name}`;
9      },
10     sayBye: (name: string) => `Bye ${name}`,
11 };
12
13 console.log(sayStuff.sayHi("Heisenberg")); // Hi Heisenberg
14 console.log(sayStuff.sayBye("Heisenberg")); // Bye Heisenberg
15
```

Cú pháp TypeScript OOP

❑ Hàm trong TypeScript

○ Chúng ta có thể xác định loại đối số của hàm, cũng như kiểu trả về của hàm.

```
TS function.ts X
src > TS function.ts > ...
1 //Define a function called circle that takes
2 // a diam variable of type number,
3 // and returns a string
4 function getCirclePerimeter(diam: number) {
5     return 'The perimeter is: ' + Math.PI * diam;
6 }
7
8 console.log(getCirclePerimeter(10));
```

○ Với hàm tương tự, chúng ta có thể sử dụng hàm mũi tên ES6

```
TS function-es6.ts X
src > TS function-es6.ts > ...
1 //Define a function called circle that takes
2 // a diam variable of type number,
3 // and returns a string
4 const getCircleCircumference = (diam: number): string => {
5     return 'The circumference is: ' + Math.PI * diam;
6 }
7
8 console.log(getCircleCircumference(10));
9
```

Cú pháp TypeScript OOP

❑ Hàm trong TypeScript

○ Chúng ta cũng có thể khai báo kiểu của dữ liệu trả về kiểu tường minh (explicit typing) hoặc kiểu ngầm định (implicit typing) đều được.

```
TS func-explicit-implicit-typing.ts X
src > TS func-explicit-implicit-typing.ts > ...
1  // Using explicit typing
2  const getCircPerimeter01: Function = (diam: number): string => {
3    return "The circumference is " + Math.PI * diam;
4  };
5
6  // Inferred typing - TypeScript sees that circle is a function
7  // that always returns a string, so no need to explicitly state it
8  const getCircPerimeter02 = (diam: number) => {
9    return "The circumference is " + Math.PI * diam;
10 };
11
12 console.log(getCircPerimeter01(5));
13 console.log(getCircPerimeter02(5));
14
15 |
```

Cú pháp TypeScript OOP

❑ Kiểu động (Dynamic Types), kiểu định danh (Alias Types) trong TypeScript

○ Chúng ta sử dụng kiểu **any** để định nghĩa 1 biến với có thể tồn tại ở nhiều kiểu khác nhau cùng lúc tùy trường hợp.

```
TS dynamic-types.ts X
src > TS dynamic-types.ts > ...
1  let age: any = "100";
2  console.log(typeof age);
3
4  age = 100;
5  console.log(typeof age);
6
7  age = {
8    years: 100,
9    months: 2,
10 };
11 console.log(typeof age);
12
```

Cú pháp TypeScript OOP

❑ Kiểu động (Dynamic Types), kiểu định danh (Alias Types) trong TypeScript

○ Kiểu định danh (Alias Type) giúp giảm code dư thừa, hiện thực nguyên tắc DRY

```
TS alias-types.ts X
src > TS alias-types.ts > ...
1  type StringOrNumber = string | number;
2
3  type PersonObject = {
4    name: string;
5    id: StringOrNumber;
6  };
7
8  const person12: PersonObject = {
9    name: "John",
10   id: 1,
11 };
12
13 const person34: PersonObject = {
14   name: "Delia",
15   id: 2,
16 };
17
18 const sayHello = (person: PersonObject) => {
19   return "Hi " + person.name;
20 };
21
22 const sayGoodbye = (person: PersonObject) => {
23   return "Seeya " + person.name;
24 };
25
26 console.log(sayHello(person12));
27 console.log(sayGoodbye(person34));
28
```

Cú pháp TypeScript OOP

❑ Lớp trong TypeScript

○ Có thể khai báo kiểu của các tập dữ liệu giống nhau vào trong một lớp (class)

```
TS Person.ts  X
src > class > TS Person.ts > ...
1  class Person {
2      name: string;
3      isCool: boolean;
4      pets: number;
5
6      constructor(n: string, c: boolean, p: number) {
7          this.name = n;
8          this.isCool = c;
9          this.pets = p;
10     }
11
12     sayHello() {
13         return `Hi, my name is ${this.name} and I have ${this.pets} pets`;
14     }
15 }
16
17 const person04 = new Person('Danny', false, 1);
18 const person05 = new Person('Sarah', true, 5);
19 console.log(person04.sayHello());
20 console.log(person05.sayHello());
21
```

Cú pháp TypeScript OOP

❑ Lớp trong TypeScript

○ Có thể khai báo kiểu của các tập dữ liệu giống nhau vào trong một lớp (class)

```
TS Person.ts  X
src > class > TS Person.ts > ...
1  class Person {
2      name: string;
3      isCool: boolean;
4      pets: number;
5
6  >  constructor(n: string, c: boolean, p: number) {
10      }
11
12 >  sayHello() { ...
14      }
15  }
16
17  const person04 = new Person('Danny', false, 1);
18  const person05 = new Person('Sarah', true, 5);
19  console.log(person04.sayHello());
20  console.log(person05.sayHello());
21
22  let persons: Person[] = [person04, person05];
23  console.log(persons);
```


Cú pháp TypeScript OOP

❑ Lớp trong TypeScript

- Có thể khai báo kiểu các đặc tả truy xuất cho các thuộc tính trong một lớp.

```
TS AMPerson.ts X
src > class > TS AMPerson.ts > ...
1  //using AM (Access Modifier)
2  class AMPerson {
3      readonly name: string; // This property is immutable - it can only be read
4      private isCool: boolean; // Can only access or modify from methods within this class
5      protected email: string; // Can access or modify from this class and subclasses
6      public pets: number; // Can access or modify from anywhere - including outside the class
7
8      constructor(n: string, c: boolean, e: string, p: number) {
9          this.name = n;
10         this.isCool = c;
11         this.email = e;
12         this.pets = p;
13     }
14
15     sayMyName() {
16         console.log(`Your not Heisenberg, you're ${this.name}`);
17     }
18 }
19
20 const person06 = new AMPerson("Danny", false, "dan@e.com", 1);
21 console.log(person06.name); // Fine
22 // person06.name = "James"; // Error: read only
23 // console.log(person06.isCool); // Error: private property - only accessible within PAMPerson class
24 // console.log(person06.email); // Error: protected property - only accessible within AMPerson class and its subclasses
25 console.log(person06.pets); // Public property - so no problem
```

Cú pháp TypeScript OOP

❑ Lớp trong TypeScript

- Có thể rút gọn mã code bỏ việc khai báo các thuộc tính vào thẳng constructor.

```
TS ShortPerson.ts X
src > class > TS ShortPerson.ts > ...
1  class ShortPerson {
2      constructor(
3          readonly name: string,
4          private isCool: boolean,
5          protected email: string,
6          public pets: number
7      ) {}
8
9      sayMyName() {
10         console.log(`Your not Heisenberg, you're ${this.name}`);
11     }
12 }
13
14 const person07 = new ShortPerson("Danny", false, "dan@e.com", 1);
15 console.log(person07.name); // Danny
```

Cú pháp TypeScript OOP

❑ Kế thừa trong TypeScript

```
TS Person02.ts X TS Programmer.ts
src > class > TS Person02.ts > Person02
1  export class Person02 {
2      name: string;
3      email: string;
4      pets: number;
5
6      constructor(n: string, e: string, p: number) {
7          this.name = n;
8          this.email = e;
9          this.pets = p;
10     }
11
12     sayMyName() {
13         console.log(`Your not Heisenberg, you're ${this.name}`);
14     }
15 }
```

```
TS Programmer.ts X TS Person02.ts
src > class > TS Programmer.ts > Programmer > programmingLanguages
1  import { Person02 } from "../Person02";
2
3  class Programmer extends Person02 {
4      programmingLanguages: string[];
5
6      constructor(
7          name: string,
8          email: string,
9          pets: number,
10         pLs: string[]
11     ) {
12         // The super call must supply all parameters for base (Person02) class,
13         // as the constructor is not inherited.
14         super(name, email, pets);
15         this.programmingLanguages = pLs;
16     }
17
18     let pLanguages = ["Java", "JavaScript"];
19     let person09 = new Programmer("Danny", "dan@e.com", 1, pLanguages);
20     console.log(person09);
21 }
```

Cú pháp TypeScript OOP

❑ Interface với Class trong TypeScript

```
TS InterfaceWClass.ts X
src > interface > TS InterfaceWClass.ts > ...
1  interface HasFormatter {
2    format(): string;
3  }
4
5  class Person03 implements HasFormatter {
6    constructor(public username: string, protected password: string) {}
7
8    format() {
9      return this.username.toLocaleLowerCase();
10   }
11 }
12
13 // Must be objects that implement the HasFormatter interface
14 let person08: HasFormatter;
15 let person10: HasFormatter;
16
17 person08 = new Person03("Danny", "password123");
18 person10 = new Person03("Jane", "TypeScript2023");
19
20 console.log(person08.format()); // danny
21 console.log(person10.format()); // jane
```

Cú pháp TypeScript OOP

❑ Generic trong TypeScript:

Enums cho phép định nghĩa hoặc khai báo một tập hợp các giá trị liên quan, có thể là số hoặc chuỗi, dưới dạng một tập hợp các hằng số được đặt tên

```
TS ResourceType.ts X
src > enum > TS ResourceType.ts > ...
1  enum ResourceType01 {
2      BOOK,
3      AUTHOR,
4      FILM,
5      DIRECTOR,
6      PERSON,
7  }
8  console.log(ResourceType01.BOOK); // 0
9  console.log(ResourceType01.AUTHOR); // 1
10
11 // To start from 1
12 enum ResourceType02 {
13     BOOK = 1,
14     AUTHOR,
15     FILM,
16     DIRECTOR,
17     PERSON,
18 }
19 console.log(ResourceType02.BOOK); // 1
20 console.log(ResourceType02.AUTHOR); // 2
21
```

```
TS Direction.ts X
src > enum > TS Direction.ts > ...
1  enum Direction {
2      Up = "Up",
3      Right = "Right",
4      Down = "Down",
5      Left = "Left",
6  }
7
8  console.log(Direction.Right); // Right
9  console.log(Direction.Down); // Down
10
```

Cú pháp TypeScript OOP

❑ Generic trong TypeScript:

○ Generics cho phép tạo 1 thành phần có thể hoạt động trên nhiều loại khác nhau, thay vì 1 loại, giúp làm cho thành phần đó có thể tái sử dụng nhiều hơn

```
TS GenericPerson.ts X
src > generic > TS GenericPerson.ts > ...
1  const addID = (obj: object) => {
2    let id = Math.floor(Math.random() * 1000);
3
4    return { ...obj, id };
5  };
6
7  let person11 = addID({ name: "John", age: 40 });
8
9  console.log(person11.id); // 271
10 // console.log(person11.name); // ERROR: Property 'name' does not exist on type '{ id: number; }'.
11
```

Cú pháp TypeScript OOP

❑ Generic trong TypeScript:

- Ở ví dụ trước đó, **TypeScript** báo lỗi khi truy cập thuộc tính name do khi chúng ta chuyển một đối tượng vào addID, chúng ta không chỉ định những thuộc tính mà đối tượng này nên có.
- Vì vậy TypeScript không biết đối tượng đó có những thuộc tính nào mà thuộc tính duy nhất TypeScript biết có trên đối tượng được trả về là id.
- Vậy làm cách nào chúng ta có thể chuyển bất kỳ đối tượng nào vào addID, nhưng vẫn cho TypeScript biết những thuộc tính và giá trị mà đối tượng đó có?

Cú pháp TypeScript OOP

❑ Generic trong TypeScript:

- Chúng ta có thể sử dụng **generic**, **<T>** - trong đó T được gọi là tham số loại:

```
12 //Fix error by generic
13 // <T> is just the convention - e.g. we could use <X> or <A>
14 ✓ const addID02 = <T>(obj: T) => {
15     let id = Math.floor(Math.random() * 1000);
16
17     return { ...obj, id };
18 };
19 let person15 = addID02({ name: 'John', age: 40 });
20
21 console.log(person15.id); // 271
22 console.log(person15.name); // John
```


Cú pháp TypeScript OOP

❑ Generic trong TypeScript:

○ Khi không biết loại giá trị nào đó trong một đối tượng sẽ có trước thời hạn, chúng ta có thể sử dụng giá trị **generic** để chuyển vào kiểu dữ liệu tổng quát:

```
TS GenericWinterface.ts X
src > generic > TS GenericWinterface.ts > ...
1  // The type, T, will be passed in
2  interface Person05<T> {
3      name: string;
4      age: number;
5      documents: T;
6  }
7
8  // We have to pass in the type of `documents` - an array of strings in this case
9  const person20: Person05<string[]> = {
10     name: "John",
11     age: 48,
12     documents: ["passport", "bank statement", "visa"],
13 };
14
15 // Again, we implement the `Person` interface, and pass in the type for documents - in this case a string
16 const person21: Person05<string> = {
17     name: "Delia",
18     age: 46,
19     documents: "passport, P45",
20 };
21
22 console.log(person20);
23 console.log(person21);
```

Tóm tắt bài học

- ☐ Tổng quan về TypeScript
- ☐ Tại sao nên sử dụng TypeScript
- ☐ Cài đặt dự án TypeScript
- ☐ Cú pháp TypeScript cơ bản
- ☐ Cú pháp TypeScript OOP

