TUTORIAL TYPESCRIPT

# VARIABLE

## Basic

// Boolean

let free: boolean = false;

free = true

console.log(free)

// Number

let num: number = 10

console.log(num)

// String

let str: string = "TS"

console.log(str)

console.log(`TS - ${num} - ${free}`)

Phải khai báo kiểu dữ liệu của biến.

## Array

// Array num

let arrNum: Array<Number> = [1, 2, 3]

console.log('arrNum = ', arrNum)

// Array string

let arrStr: string[] = ["ABC", "DEF", "KLM"]

console.log('arrStr = ', arrStr.push("TS"), arrStr)

## Tuple

// Tuple: hỗn hợp nhiều kiểu biến

let xtuple: [string, number, Array<number>]

xtuple = ["35", 12, [1, 2, 3]]

console.log(xtuple)

## Enum

// Sử dụng để gán text thay cho code để tường minh hơn, và tạo group constant

// 100 created, 101 process, 102 finish

// -- Default: mặc định thứ tự là 0, 1, 2,…

enum STATUS { created, process, finish }     // 0, 1, 2

// let todoStatus = STATUS.created

// console.log(todoStatus)

// -- Gán giá trị cho Obj enum:

// enum STATUS { created = 100, process = 200, finish }     // 0, 1, 2

// let todoStatus = STATUS.finish                           // 201 = process + 1

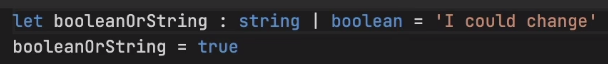
console.log(STATUS)

Text

Description automatically generated

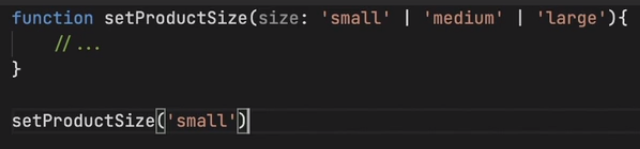
## Union type

Variable can multiply type data.



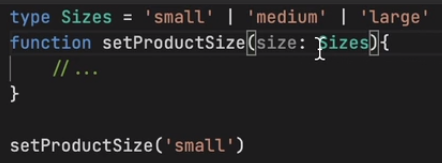
## Literal types

Kiểu dữ liệu sử dụng để giới hạn options các tham số khai báo ở funtion.



## Type alias

Giống với Literal types, sử dụng để giới hạn các tham số khai báo ở function.



Hoặc khai báo định nghĩa Object, variable custom type:

Text

Description automatically generated

## Any

let vAny: any[] = [1, true, "hello"]

console.log(vAny)

## Void

function fnVoid():void {

    console.log("Void sử dụng để thực thi khối lệnh trong function, không trả về kết quả");

}

fnVoid()

## Type assertion

let mAny : any = "Hello world"

console.log((<string>mAny).length);

## Interface obj

// Obj

let ObjOne = {}

console.log("ObjOne === ", ObjOne);

// Interface

interface CourseInterface {

    name: string,

    time: number,

    free?: boolean                  // Obj kế thừa có thể có thuộc tính này hoặc không.

}

// Obj Interface

let ObjTwo : CourseInterface;

ObjTwo = {

    name: "PHP",

    time: 8

}

console.log("ObjTwo === ", ObjTwo);

// Obj three

let ObjThree: CourseInterface = {

    name: "PHP",

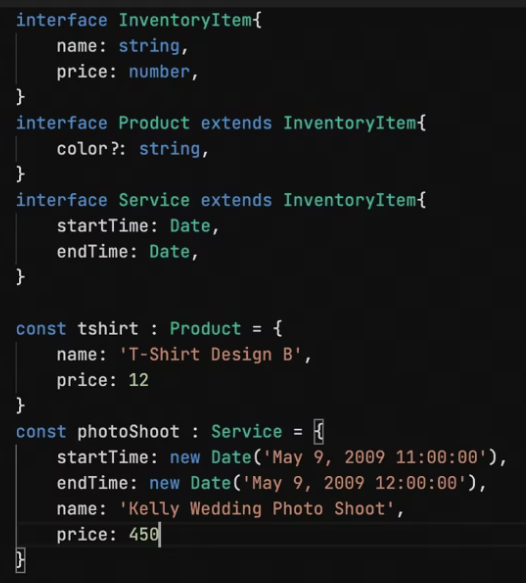
    time: 8,

    free: false

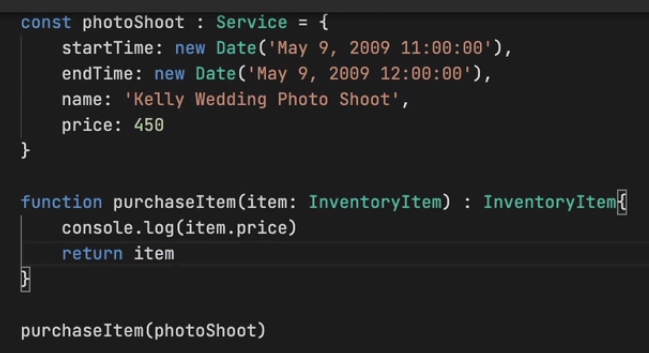
}

console.log("ObjThree === ", ObjThree);

## Extends interface



## Using interfaces



## Interface array

// Interface array

interface ListCourseInterface {

    [index:number]: string;

}

// Array

let Course: ListCourseInterface = ["PHP", "MySQL"]

console.log(Course);

# FUNCTION

## Return

// No return

function funcNoReturn(): void{

    console.log("funcNoReturn");

}

funcNoReturn()

// Return number

function funcReturnNumber(): number {

    return 2;

}

console.log(funcReturnNumber());

// Return string

function funcReturnString(): string {

    return "ABC";

}

console.log(funcReturnString());

// Return Array

function funcReturnArray(): [number, number, number] {

    return [1, 2, 3];

}

console.log(funcReturnArray());

// Return Object

function funcReturnObject(): {age: number, name: string} {

    return {age: 1, name: "ABC"};

}

console.log(funcReturnObject());

// Return Any

function funcReturnAny(): any {

    return [4, 6, 3];

}

console.log(funcReturnAny());

## Params

// 1. Đầy đủ tham số

function showInfo1(name: string, age: number): string{

    return `My name is ${name}, ${age} years old`

}

console.log(showInfo1("John", 20));

// 2. Tham số mặc định

function showInfo2(name: string = "Py", age: number = 40): string {

    return `My name is ${name}, ${age} years old`

}

console.log(showInfo2());

// 3. Tham số khuyết (?)

function showInfo3(name: string = "Jy", age?: number): string {

    if(age === undefined){

        return `My name is ${name}`

    }else{

        return `My name is ${name}, ${age} years old`

    }

}

console.log(showInfo3("Pyter"));

// 4. Tham số truyền vào nhiều hơn 1 kiểu dữ liệu

function showInfo4(x: (string | any[]), y:string[]): number {

    return x.length + y.length

}

console.log(showInfo4("abc", ['1234']));

## Rest

// rest ... --> các tham số còn lại

function showInfo(name: string, ...course: string[]): string {

    return name + ' study ' + course.join(", ")

}

console.log(showInfo("John", "ES6", "PHP", "TypeScript"));

## Function type

// Method 1:

function UserInfo01(name: string, age: number): string{

    return `My name is ${name}, ${age} years old!`

}

// console.log(UserInfo01("John", 20));

// Method 2:

let UserInfo02 = function(name: string, age: number): string{

    return `My name is ${name}, ${age} years old!`

}

// Method 3: Arrow function

let f1 = function(i:number): number { return i \* i}

let f2 = function(i:number) { return i \* i}

let f3 = (i:number) : number => { return i \* i}

let f4 = (i:number) => { return i \* i}

let f5 = (i:number) => i \* i

console.log(f5(5));

## Overload

function showName(x: string):string;

function showName(x: number):number;

function showName(x: any):any {

    return x

}

console.log(showName("12"));

console.log(showName(12));

console.log(showName("18"));

# OOP

## Class

// Khai báo class

class Task { // Định nghĩa class

    // Khai báo biến

    id: number;

    name: string;

    status: string;

    // Phương thức khởi tạo

    constructor(id: number, name: string, status: string){

        this.id = id;

        this.name = name;

        this.status = status

    }

    // Phương thức

    showTaskInfo(){

        return `${this.id} - ${this.name} - ${this.status}`;

    }

}

// Đối tượng của class

let task = new Task(1, "Coding", 'created');

// console.log('Task: ', task);

// console.log('Task Info: ', task.showTaskInfo());

**Example 2**

Text

Description automatically generated

## Class interface

// Enum

enum TaskStatus {

    created = 100,

    Active,

    InActive,

    Process,

    Finish

}

//  Định nghĩa Đối tượng TaskInterface: Đối tượng

interface TaskInterface {

    id: number,

    name: string,

    status?: TaskStatus;

}

// Khai báo class

class TaskService {

    public static user\_name = "John";  // Static: có thể truy cập trực tiếp vào từ class mà không cần tạo ra đối tượng

    // Khai báo biến

    tasks: TaskInterface [];  // Task này có kiểu dữ liệu là TaskInterface (Obj), tasks tạo ra là mảng [], chứa đối tượng TaskInterface

    // Khởi tạo

    constructor(tasks: TaskInterface []){

        this.tasks = tasks;

    }

    // Phương thức

    getItems() {

        return this.tasks

    }

}

// Tạo ra đối tượng

let tasks: TaskInterface [] = [

    { id: 1, name: "Coding" },

    { id: 2, name: "PHP" },

    { id: 3, name: "ES6", status: TaskStatus.Finish },

];

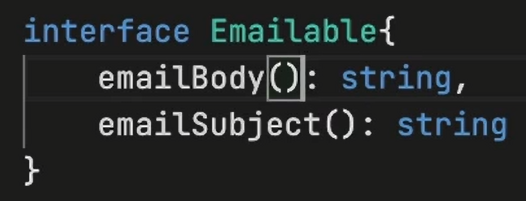
// Truy cập đối tượng

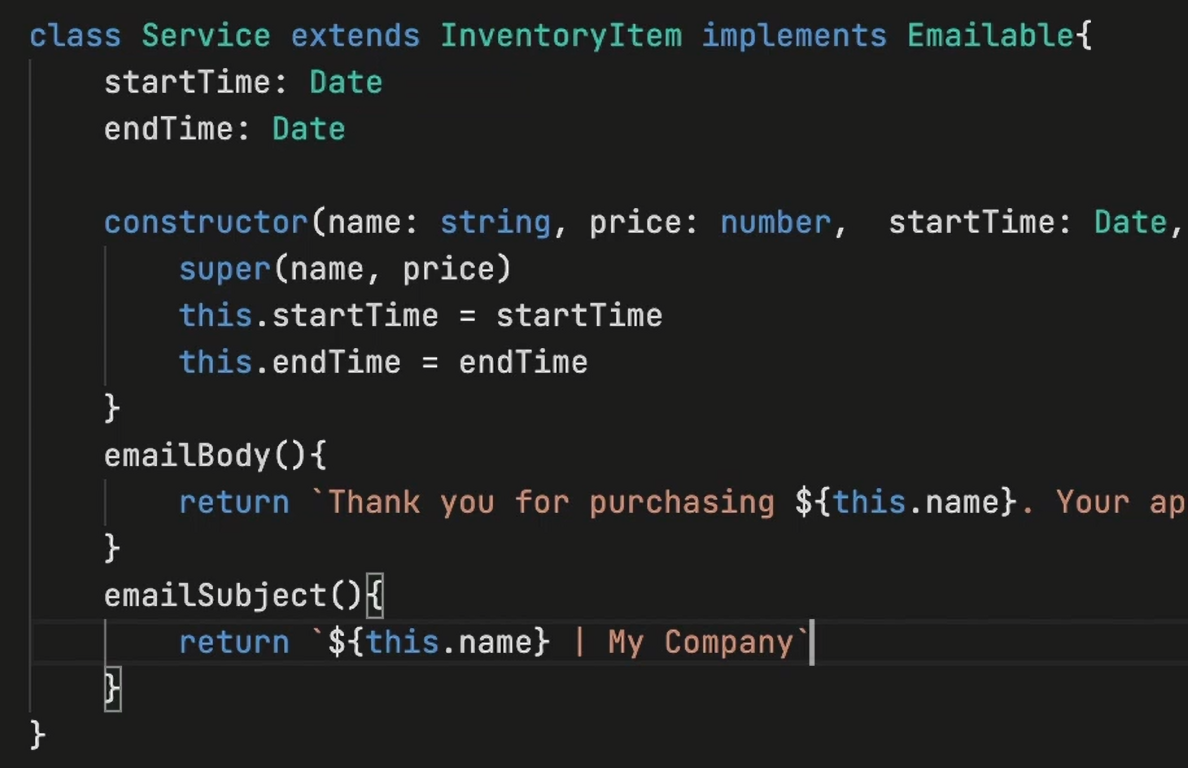
let TaskServiceObj = new TaskService(tasks)

console.log(TaskServiceObj.getItems());

console.log('User name: ', TaskService.user\_name);

Interface với khởi tạo là **phương thức**





Text

Description automatically generated

## Static

// Bài học:

// Từ khóa static đặt trước biến hoặc phương thức thì:

// -- Không cần tạo đối tượng có thể truy cập được, để truy cập cả trong và ngoài class:

// -- Truy cập bằng "Tên class"."tên biến" hoặc "Tên class"."tên hàm"

// Ví dụ:

// =====================================================================================

//  Định nghĩa TaskInterface

interface TasksInterface {

    id: number,

    name: string,

    state?: string;

}

// Khai báo class

class TasksStatic {

    public static user\_name = "John";  // Static: có thể truy cập trực tiếp biến từ class mà không cần tạo ra đối tượng

    // Khai báo biến

    static tasks: TasksInterface [];

    // Khởi tạo

    constructor(tasks: TasksInterface []){

        TasksStatic.tasks = tasks;

    }

    // Phương thức

    getItems() {

        return TasksStatic.tasks

    }

    // Show item info: thêm static để truy cập trực tiếp vào phương thức mà không cần khởi tạo đối tượng

    static showItemInfo(): void {

        for(let task of TasksStatic.tasks){ // Truy cập trong class: TasksStatic.tasks

            console.log(`${TasksStatic.user\_name} study ${task.name} `);

        }

    }

}

// Tạo ra đối tượng

let obj: TaskInterface [] = [

    { id: 1, name: "Coding" },

    { id: 2, name: "PHP" },

    { id: 3, name: "ES6"},

];

// Truy cập đối tượng

let taskObj = new TasksStatic(obj)

console.log('User name: ', TasksStatic.user\_name);  // Truy cập ngoài class TasksStatic.user\_name (không cần tạo đối tượng)

console.log(TasksStatic.showItemInfo());            // Truy cập ngoài class TasksStatic.showItemInfo() (không cần tạo đối tượng)

## Inheritance

/\*

    Nội dung: Kế thừa Class (khai báo biến và phương thức) và mở rộng thêm thuộc tính, phương thức mới

    -- Sử dụng từ khóa "extends" để kế thừa lớp cha

    -- Kế thừa thuộc tính: sử dụng từ khóa super(variable 1, variable2, ...)

    -- Kế thừa phương thức: sử dụng từ khóa super."Tên phương thức cha"

\*/

class CourseParent {

    id: string;

    name: string;

    price: number;

    constructor(id: string, name: string, price: number){

        this.id = id;

        this.name = name;

        this.price = price;

    }

    showCourseInfo(): void{

        console.log(`${this.id} - ${this.name} - ${this.price}`);

    }

}

class CourseChilren extends CourseParent { //extends: kế thừa lớp cha

    author: string;

    constructor(id: string, name: string, price: number, author: string){

        // Kế thừa thuộc tính lớp cha: từ khóa super(variable 1, variable2, ...)

        super(id, name, price);

        this.author = author;

    }

    // Kế thừa phương thức

    showCourseInfo(): void {

        // Kế thừa phương thức

        super.showCourseInfo();

        // Bổ sung phương thức

        console.log(`Author: ${this.author}`);

    }

}

let childObj = new CourseChilren('1', 'PHP', 123, "Nguyễn Văn Khánh");

childObj.showCourseInfo()

## Access modifier

/\*

    Nội dung: sử dụng từ khóa public, private, protected đối với biến và phương thức

                    Inside          Outside         Children

    public             +                +               +

    private            +                -               -

    protected          +                -               +

    Diễn giải:

    -- public: có thể truy cập ở bên trong và ngoài class, truy cập ở lớp con kế thừa từ lớp cha

    -- private: chỉ có thể truy cập ở bên trong lớp class khai báo biến, hàm đó

    -- protected: có thể truy cập ở bên trong lớp và lớp con kế thừa.

\*/

Example về protected:

Text

Description automatically generated

## Abstract

/\*

    Nội dung: học về sử dụng abstract.

    Ý nghĩa abstract: là để định nghĩa một class hoặc một phương thức; Sau đó lớp khác kế thừa từ lớp này.

    -- Đối với class: abstract class name\_class {}

    -- Đối với method: public abstract name\_method (): kiểu dữ liệu trả về"

\*/

// Khai báo lớp abstract: lớp đặc điểm chung nhất của đối tượng

abstract class Laptop {

    // Phương thức abstract chỉ khai báo tên method, không được định nghĩa method abstract.

    public abstract keyboard(): void;

    public mainboard(): void {

        console.log('Mainboard laptop');

    }

    public chipset(): void{

        console.log('Chipset laptop');

    }

}

// Tạo lớp mới kế thừa đặc điểm của lớp abstract và có một số đặc điểm riêng

class LaptopDell extends Laptop {

    // Định nghĩa method abstract của lớp cha

    public keyboard(): void {

        console.log('Keyboard Laptop Dell');

    }

    // Láp top dell có bàn khím khác về bàn phím

    public mainboard(): void {

        console.log('Mainboard Laptop Dell');

    }

}

// Tạo đối tượng

let laptopDell: Laptop = new LaptopDell()

laptopDell.keyboard()

laptopDell.mainboard()

## Interface

/\*

    -- interface: khai báo đặc điểm (property) và phương thức (lưu ý: không có phần xử lý, chỉ có tên biến và tên phương thức; không có định nghĩa phương thức)

    -- Mặc định các biến, phương thức trong interface là public (vì để class định nghĩa và thực thi).

    -- Class sử dụng để thực hiện (implement), class phải định nghĩa chi tiết rõ các biến và phương thức trong interface;

\*/

// Interface khai báo đặc điểm của con người

interface People {

    name: string;       // Khai báo đặc điểm

    eat(): void;        // Khai báo phương thức

    sleep(): void;

}

interface Bird {

    fly(): void;

}

// Class

class Machine {

    kmFly(km: number): number{

        return km

    }

}

// Class superman có đặc điểm và các hành động (implements) interface People (phương thức thì class --> phải bổ sung để hoàn thiện)

class Superman extends Machine implements People, Bird {

    name: string;

    constructor(name: string){

        super();                // Kế thừa lớp cha Machine

        this.name = name

    }

    // Name: Định nghĩa phương thức mới

    showName(): string{

        return this.name

    }

    // Định nghĩa superman eat --> cho interface People

    eat(): void {

        console.log("Superman eat()")

    }

    // Định nghĩa superman sleep --> cho interface People

    sleep(): void {

        console.log("Superman sleep()")

    }

    // Định nghĩa superman sleep --> cho interface Bird

    fly(): void {

        console.log("Superman fly()")

    }

}

// Tạo đối tượng

let supermanObj = new Superman("Khánh");

// Khai thác thông tin đối tượng

console.log('Name: ', supermanObj.showName());

supermanObj.eat();

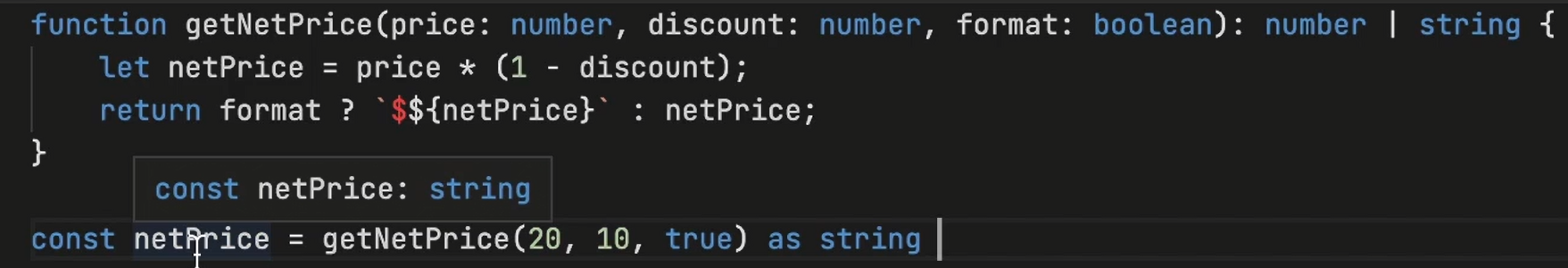
supermanObj.sleep();

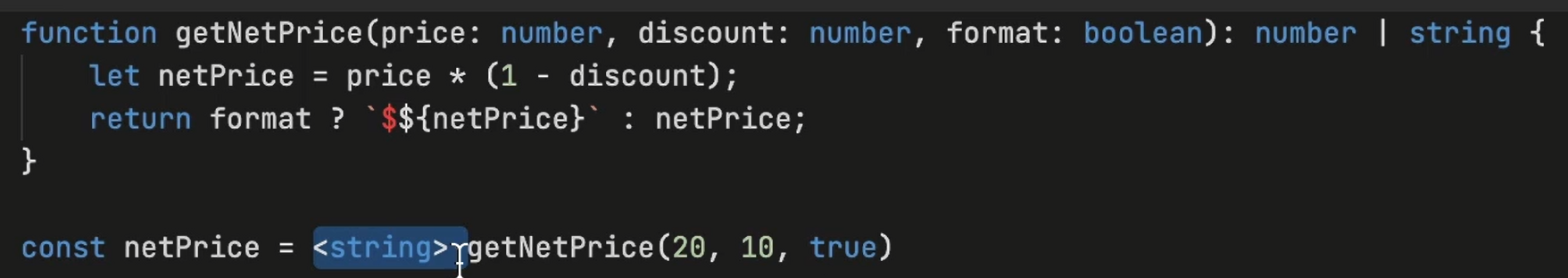
supermanObj.fly();

console.log('Km can fly: ', supermanObj.kmFly(9) ,' km');  // Kế thừa phương thức từ Machine

## Type assertion

Sử dụng cho gán kiểu dữ liệu trả về cho hàm là option. Sử dụng từ khóa **as** hoặc <>





Text

Description automatically generated

## Narrowing ( instanceof | in )

Text

Description automatically generated

Hoặc dùng **in**

Text

Description automatically generated

## Return

### By define

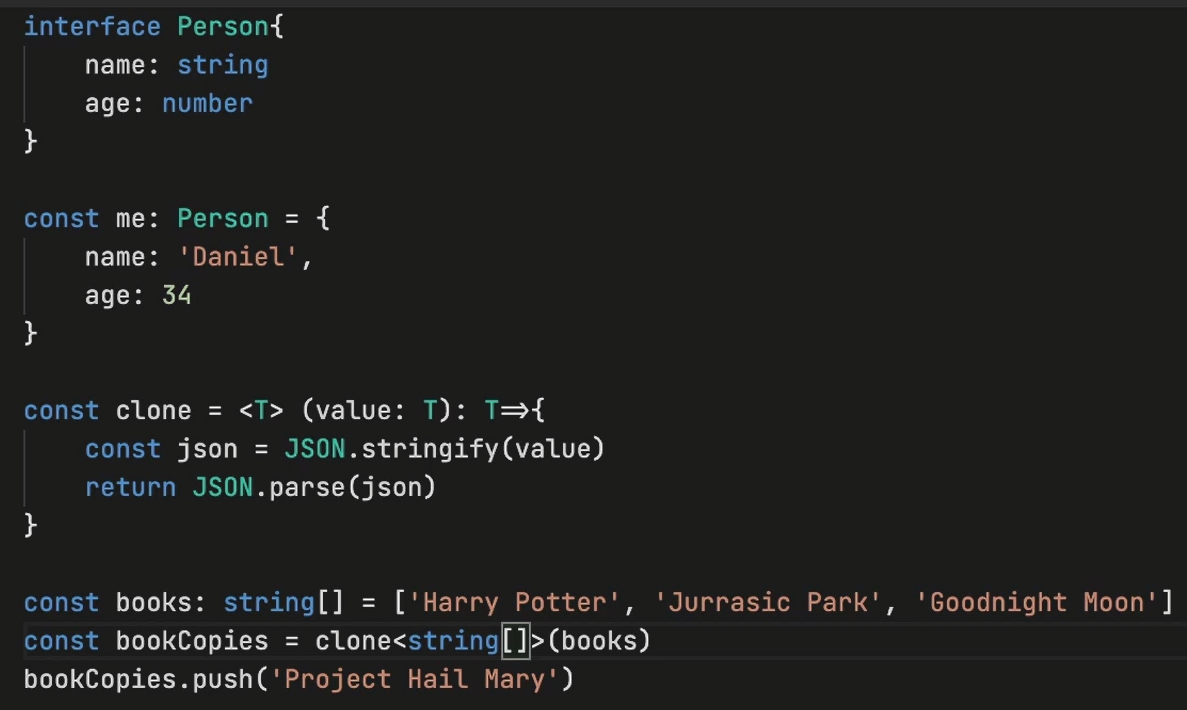
Function khi xác định kiểu dữ liệu trả về. Ở ví dụ dưới, nếu để trả về là any thì sau **myClone.** sẽ không gợi ý là age, name, phải để là Person.

Text

Description automatically generated

### Type (T)





# GENERIC

Sử dụng để khai báo kiểu dữ liệu chưa xác định, có thẻ thay đổi theo người dùng định nghĩa.

## Function

function showNumberInfo(x:number): number {

    return x;

}

function showStringInfo(x:string): string {

    return x;

}

function showInfoGeneric<T>(x: T): T {

    return x;

}

// Run function

showInfoGeneric<string>('Generic')

showInfoGeneric<number>(123)

showInfoGeneric<boolean>(true)

## Method

 class Car {

    static printArray<T>(x: T[]){

        console.log(x);

    }

 }

 Car.printArray<string>(["honda", "suzuki"])

 Car.printArray<number>([1, 2, 3])

 Car.printArray<any>(["honda", "suzuki", true, 1, 2, 3, false])

## Class

//1: id: number, name: string, price: number

//2: id: string, name: string, price: string

class Product<A, B, C> {

    id: A;

    name: B;

    price: C;

    constructor(id: A, name: B, price: C){

        this.id = id;

        this.name = name;

        this.price = price;

    }

    showProductInfo(){

        console.log(`${this.id} - ${this.name} - ${this.price}`);

    }

}

//  Product 1:

let product1 = new Product<number, string, number>(10, "Loa", 15);

product1.showProductInfo();

//  Product 2:

let product2 = new Product<string, string, string>("10", "Loa", "15 USD");

product2.showProductInfo();

# MODULE

## Export

### Variable, Class

Text

Description automatically generated

### Multi file

Text

Description automatically generated

## Import

Text

Description automatically generated



## Namespace