**1 Giới thiệu ngôn ngữ typeScript:**

* Theo hướng hướng đối tượng.
* Dược phát triển bởi Microsoft.
* .ts, sau khi nó được biên dịch ra js.
* Typescripts chính là javascript.
* Viết cho cả frontend va backend.

**Why:** - Sử dụng như PHP, C#.

* Thế hệ js đều dựa vào typeScript.

**2 Cài đặt nodejs, cài đặt git, cài đặt plugin chay thử.**

**Nodejs:** Cung cấp nền tảng có thể dịch file typeScript sang js.

Cách 1 : trên visual code:

* B1. chay câu lệnh theo dõi: tsc –init.
* B2. Nhấn tổ hợp phím ctr + shift + b.

Cách 2 : trên mọi tool code:

* Tsc tên file –watch.

**3 Kiểu dữ liệu trong trong type script.**

* String: là kiểu chữ

+ Khai báo: Let chu : string;

* Number: là kiểu số

+ Khai báo: Let so : number;

* Mảng:

+ Khai báo: Let mang : string[] = [‘teo’, ‘huy’ , ’lan’];

**4 Ba kiểu dữ liệu mới so với javascript.**

* **Enum:** gán trạng thái cho một phần tử.

+ Khai báo: enum trangThai { moiDangKy = 10 , daThuTien = 100 , hoanThanh = 1000};

+ Enum tồn tại nhãn và trang thái nếu không khai báo trạng thái thì măc địh là 0 -> …

+ Truy xuất tới trang thái: trangThai.moiDang Ky.

* **Tuple:** Mảng hỗn tạp.

+ Khai báo: Let a : [string, number] = [‘kien’, 100];

* **Boolean:** Khai báo: Let flag : boolean = true;
* **Any:** bất kì kiều gì.

+ Khai báo: Let x : any[] = [1, true , “free”];

* **Void:** Thường được dùng với function. Không return được, không trả về giá trị gì hết.

**5 Tổng kết phần kiểu dữ liệu trong typescript.**

* **Ép kiểu trong javascript:**

+ Cách 1: <string> bien : chuyển đổi sang kiểu string.

+ Cách 2: ((kieu as string).length) : ép sang kiểu string.

+ Cách 3: String(tenbien): ép sang kiểu string.

**6 Bốn kiểu function trong Typescript.**

* **Return function:** hàm có dữ liệu trả về.
* **Function có tham số:** cần phải viết tổng quát , viết một lần sử dụng nhiều lần . Có thẻ thay đổi các gí trị có nghĩa là tham số.
* **Anonymous function:** function không có tên. Thường gán cho giá trị biến. Tiết kiệm không gian tên đỡ tốn bộ nhớ cho chương trình.

let g : (x : number, y : string) => string = function(x, y){}

* **Arrow function:**

**7 Học lập trình hướng đối tượng từ đầu. Tại sao phải sử dụng lập trình hướng đối tượng.**

* **What:** Viết tắt của oop.
* **Why:**

+ Lập trình hướng đối tượng: tạo ra 1 thế giới , mỗi nhân vật trong đấy là một đối tượng trong thế giới đấy

+ Dữ liệu được che dấu, bao bọc bởi các phương thức truy xuất.

+ Các đối tượng trao đổi với nhau thông qua các hàm.

+ Chương trình được xấy dựng đi từ dưới lên.

* **How:**

+ Class: lớp một cái bản thiết kế ra đối tượng

\*Ví dụ: giỏ hàng : đối tượng của chức năng sản phẩm.

Bạn phải có bản thiết kế ,khuôn (class)của sản phẩm đấy.

Xây dựng các thuộc tính cho phương thức(property) đó : rộng bao nhiêu , dài bn, cao bn…

Xây dựng phương thức cho phương thức(method) đó: phương thức dập khuôn.

\*Ví dụ: sản phẩm: Sản phẩm

Thuộc tính: tên sản phẩm, giá , số lượng,..

Phương thức: mua sản phẩm , cập nhật giá , thay đổi ảnh…

Làm sản phẩm: nokia galaxy = instance ( một sản phẩm( thực thể ) tạo từ cái khuôn đấy ).

\*Ví dụ: Xây thành phố.

-Đối tượng là : ngôi nhà.

-Bản thiết kế ngôi nhà = class

-Thuộc tính của ngôi nhà = property: dài , rộng , cao , màu gì ,…

-Phương thức xây = method: chọn vị trí , xây 2 tầng , xây 3 tầng,..

-instance = nhà thật tạo ra từ bản thiết kế.

**9 Trừu tượng hóa, đa hình , thừa kế và đóng gói trong lập trình hướng đối tượng.**

* **Trừu tượng hóa:** Xác định các thuộc tính, phương thức kĩ năng cần thiết , cơ bản , đặc chưng đưa ra cho một class.
* **Thừa kế:** Lớp này có thể kế thừa các thuộc tính của của lớp khác.

+ extend:

* **Đa hình:** Viết lại các thuộc tính và phương thức của lớp khác.

+ abstract: Một bản thiết kế dành cho phương thức.

+ interface: Một bản thiết kế dành cho thuộc tính.

* **Đóng gói:** Cho phép người dùng thay đổi không được tự do nữa có thuộc tính được thay đổi , có thuộc tính không được thay đổi.

**10 class:** bản thiết kế , kiểu dữ liệu mới.

* Thuộc tính:
* Phương thức: là các function thao tác với thuộc tính.
* Constructor: bắt buộc phải có để tạo ra instance.
* Instance: đối tượng được tạo ra.

**Enum: ví dụ tạo trạng thái :**

* Enum state { Create = 10, Processing , Finish };
* Truyền vào biến trạng thái: trangThai : state;

**11 Ý nghĩa của Static trong lập trình hướng đối tượng.**

**Why:** phân quyền cho thuộc tính hoặc phương thức .

* Một giá trị trong class.
* Một phương thức trong class.

**How:** Không cần khởi tạo đối tượng giúp ta in ra luôn.

* Dùng luôn không cần phải tạo 1 instance nào cả.

**12 Kế thừa trong typescript.**

**What:** kế thừa là thừa hưởng lại các thuộc tính và phương thức của bố.

* Không cần làm lại cái cũ mình có thể phát triển thêm.

**Why:**

* Tiết kiệm dung lượng nhớ.
* Dễ thay đổi hàng loạt.
* Các class xây dựng theo hệ thống.

**How: extends:** khai báo ở class con để kế thừa từ class cha.

* **Super(thuộc tính muốn sử dụng): super** là bố copy nguyên code trong hàm cha paste vào vị trí đó.

**+** hàm sử dụng trong constructor để đỡ phải viết lại code.

**13 Access Modifier Có cần dùng access modifier.**

**What:** Access = truy nhập; modifier: người chỉnh sửa.

* Dùng 3 từ khóa public, private , protected -> cài đặt quyền cho thuộc tính và phương thức.

**Why:** Muốn bảo mật thì phải sử dụng access modifier.

* Đóng gói dữ liệu.
* Giúp ẩn dấu , bao bọc dữ liệu.
* Bảo mật dữ liệu.

**How:**

* **Public:** ở đâu cùng truy cập được.

+trong class : được.

+bên ngoài class : được.

+khu vực từ class con : được.

* **Private:**

+trong class: được.

+ngoài class: không được.

+khu vực từ class con: không được.

* **Protected:**

+ trong class: được.

+ ngoài class: không được.

+ khu vực từ class con: được.

**14 Lý thuyết về Accessor trong Typescript.**

**What:** accessor = người truy nhập

* Là các hàm set và gét để truy nhập đến đối tượng trong class.
* Để bảo mật dữ liệu.

**Why:** Public hết thì dữ liệu không được bảo mật tốt.

+**Private**: bảo mật tốt nhưng lại không thể truy nhập được từ bên ngoài.

Muốn truy nhập được như public, lại vừa muốn bảo mật.

* Sử dụng **accessors** : ta có thể đặt điều kiện cho việc truy nhập vào phần tử.

**How:** Tạo 2 hàm **getter** và setter.

+ các thuộc tính phải có thêm dâu: \_tên cho thuộc tính , tên hàm không có \_ ;

+ getter: lấy ra thông tin thông qua trung gian để xử lý;

+ setter: Sửa thông tin thông qua hàm để xử lý;

**15 Sử dụng Abstract class.**

**What:** class: bản thiết kế của đối tượng.

* **Abstract**: là bản thiết kế của class 🡪khi mà một class nó phức tạp quá quá mức .Người ta dùng **Abstact** mô tả lại.
* **Abstract**: bạn tóm tắt lại các class.

**Why:** Giúp cho người dùng lập trình hình dung ra các chức năng quan trọng của một class mà không bị phân tán suy nghĩ vào chi tiết.

Ví Dụ: Khi mình sử dụng điện thoại để gửi tin nhắn, thì mình chỉ ấn text và ấn gửi để gửi, 🡺 không quan tâm đến việc sẽ gửi thế nào.

* **Abstract: (** Bản thiết kế, bản vẽ , mô tả chứng năng của class **)** Là sử dụng nâng cao ở mức thiết kế class. Chỉ mô tả chức năng của class .Sử lý ở trong class. Có thể dùng hoặc không dùng.

**How: 1.Abstract class:** Không được dùng để khởi tạo đối tượng.

* Chỉ dùng để thiết kế class.
* Xem class hoạt động thế nào.
* **Abstract** class sinh ra là để cho kế thừa ( không phải để dùng ). Tạo class sử dụng phải kế thừa, sử dụng từ **abstract**.

**16 Sử dụng Abstract method.**

* Phương thức abstract: **abstract method** là một bản thiết kế cho function.
* How: Khi khai báo code bên trong đó Abstract method thì không có code bên trong đó, chỉ dữ liệu thôi.
* abstract guiTinNhan() : void;
* abstract goiDienThoai() : string;
* Trong class kế thừa phải bắt buộc viết phương thức abstract đấy.
* Giúp cho mình nhìn rõ trong class này có tính năng gì.

**17 Generic trong lập trình typescript:**

**What:** là cách khai hàm nhưng không cần chỉ ra kiểu dữ liệu của đối số và của chính hàm đó.

* function  xemThongTin4<T>(x:T):T{}
* **generic:**

+ có thể nhận vào biến tham số ,

+ còn nhận về kiểu dữ liệu của tham số đó nữa.

**Why:** Nó sẽ **mạnh** hơn hàm bình thường

* hàm này chỉ ra dữ liệu ở sau, không cần khai báo kiểu dữ liệu trước.

**How:** Sử dụng dấu <T> để làm kí hiệu generic.

**18 Generic trong class:**

Các biến có kiểu dữ liệu linh hoạt. khái báo biến kiểu dữ liệu trước.

**19 Interface trong typescripts:**

**What:** interface là giao diện

* Giao diện thể hiện chức năng bên ngoài hoạt động như thứ nào, không cho biết chi tiết bên trong.( giống như là giao diện web , cho biết chức năng chạy như thế nào , nhưng chưa triển khai thật , gần giống abstract , qui mô áp dụng nhỏ )

**Why:** Khi làm việc nhóm, thì người leader cần phải vẽ ra chức năng chi tiết cho các class các hàm . đảm bảo tính hệ thống của chương trình,

**How:** gom nhóm kiểu dữ liệu thanh 1 nhóm để sử dụng chung cho function.

**? : number** có cũng được , không có cũng không sao.

**Interface Class:** Đóng vai trò như một qui chuẩn cho class. Mô tả những phần tử cơ bản nhất.

* Những class tuân theo nó thì phải làm đúng theo bản thiết kế của interface.
* **Sử dụng** dùng từ khóa **implements.** Class phải thực hiên theo chuẩn của interface.

**19 Namespace và Export:**

Name space: là không gian bộ nhớ của tên:

1 đặt tên class không trùng từ khóa.

2 Quản lý theo nhóm gọi là module hệ thống.

* Khai báo tên trong class trùng tên với class bị lỗi: thì mình sử dụng từ khóa **module** để bọc class lại. Và sử dụng điền từ khóa **export** ( để mở gói cho thằng này bay ra ngoài ).
* Gọi new tenmodule.tenclass();

**20 Project giỏ hàng:**