Type Script

برای یادگیری TS شما باید ابتدا به طور کامل JS را فرا گرفته باشید زیرا TS دقیقا بر اساس JS نوشته شده است.

TS را می توان هم در فرانت اند و هم در بک اند و هم در ابزار های فول استک مانند Next JS استفاده کرد.

سایت اصلی TS :

https://www.typescriptlang.org/

دو قسمت مهم Handbook و Docs را دارد که قسمت اول برای یادگیری مفاهیم عمیق هست.

در VS Code اکستنشن JavaScript and Type Script Nightly وجود دارد.

روش نصب

دو روش برای نصب وجود دارد : npm و npx

روش npm :

```
npm install -g typescript
```

روش npx برای زمانی هست که می خواهیم TS نصب نشود و فقط میخواهیم روی فایلی که قبلا ساخته ایم اجرا شود :

```
npx tsc main.ts
```

در TS هر داده باید نوعش مشخص شود.

انواع تعیین نوع داده :

۱- نوع داده برای یک متغیر :

به وسیله : بعد از تعریف متغیر و قبل مقدار دهی

```
let title: string = 'book1'
let id : number = 1
let check : boolean = true
```

۱-۱ تعریف دو تایپ برای یک متغیر :

با استفاده از | می توانیم دو تایپ به یک متغیر تخصیص دهیم

```
let proId : number | string = 123;
proId = 'Amin'
```

۲- تعیین نوع داده برای یک Object :

دو روش برای این کار وجود دارد.

روش اول :

با نوشتن کلمه کلیدی object

```
let pro: object
pro = {
    id: 'p12dFrs8',
    title: 'Book',
    price: 9,
    exist: true,
}
```

روش دوم :

در این روش به صورت دقیق می توانیم به تمام المان های داخل Object به صورت اختصاصی تایپ تخصیص بدهیم توجه داشته باشید که در این روش حتما باید تمام المان های تعریف شده داخل آبجکت استفاده بشود

```
let pro : {
    id: string | number,
    title: string,
    price: number,
    exist: boolean,
}
 pro = {
    id: 'p12dFrs8',
    title: 'Book',
```

```
    price: 9,
    exist: true,
}
```

۳- تعیین نوع داده برای آرایه :

دو روش برای این کار داریم.

روش اول :

استفاده از کلمه کلیدی Array<> که داخل <> باید نوع دیتای مورد استفاده در آرایه مشخص شود.

```
let myArr : Array<string>
myArr = [ 'parsa' , 'arsam' , 'amin' ]
```

روش دوم:

ابتدا نوع دیتا را مشخص می کنیم سپس [ ]

```
let myArr : string[]
myArr = [ 'Parsa' , 'Arsam' , 'Amin' ]
```

برای تعریف آرایه چند تایپی نیز از روش زیر استفاده می کنیم:

```
let myArr : (string | number)[]
myArr = [ 'Parsa' , 'Arsam' , 'Amin' , 7 ]
```

یا

```
let myArr : Array< string | number >
myArr = [ 'Parsa' , 'Arsam' , 'Amin' , 7 ]
```

توابع در TS

توابع در TS از دو جهت مورد بررسی قرارمی گیرد، زمانی که آرگمان ورودی دارند و از لحاظ داشتن یا نداشتن مقدار بازگشتی

آرگمان ورودی :

```
function add (id : number , title: string) {
 console.log(id + title);
}
```

مقدار بازگشتی :

۱- زمانی که ما return نداریم باید کلمه کلیدی void را بنویسیم:

```
function add () : void {
 console.log(5 + 7);
}
```

۲- زمانی که ما return برای تابع داریم که باید نوع مقدار بازگشتی را مشخص کنیم :

```
function plus ( num1 : number , num2: number) : number {
 return num1 + num2;
}
```

نکته : ما زمانی که از const در تعریف متغیر استفاده می کنیم اگر تایپ در نظر نگیریم اروری نمیگیریم دلیل آن این هست که TS می داند که این مقدار از نظر نوع تغییری نخواهد کرد.

```
function plus ( num1 : number , num2: number) : number {
 const p =  num1 + num2;
 return p
}
```

تایپ های داینامیک یا (custom type) :

تا این قسمت انواع تایپ ها گفته شد اما زمانی هست که پروژه ما بزرگ هست در این صورت تعریف به شیوه های بالا کمی سخت هست تایپ اسکریپت برای این موضوع تایپ های داینامیک را هم در نظر گرفته است

روش تعریف به این صورت هست که :

۱- استفاده از کلمه کلیدی type

۲-اسم دلخواه در نظر میگیریم که این اسم حتما باید حرف اولش بزرگ باید به مانند تعریف کامپوننت ها در Next

۳-علامت = باید قرار بگیرد اما توجه داشته باشید که این علامت به معنای ریختن مقدار در متغیر نیست.

۴- تایپ مورد نظر نوشته می شود

حالا می توان این تایپ را n جای مختلف استفاده کرد.

مثال:

```
type pro = string | number

let proId :pro = 1234

proId = 'amin'
```

```
type Pro = {
    id: number,
    title: string,
    price: number,
    exist: boolean,
}

let product : Pro = {
    id: 2,
    title: 'book',
    price: 23,
    exist: false
}
```

ترکیب دو یا چند تایپ داینامیک به صورت زیر هست:

```
type CartItem = {
    pro: string[],
}

type User = {
    userName: string,
}

type UserProduct = User & CartItem

let userProduct = {
    pro : ['book1' , 'book2'],
    username: 'para',
}
```

اما روش دیگری برای نوشتن داینامیک تایپ وجود دارد که روش interface هست

در این روش ما به جای کلمه type ، interface را مینویسم اما علامت = را نداریم

```
interface Pro  {
    id: number,
    title: string,
    price: number,
    exist: boolean,
}

let product : Pro = {
    id: 2,
    title: 'book',
    price: 23,
    exist: false
}
```

ترکیب دو یا چند تایپ داینامیک به روش interface به صورت زیر هست :

```
interface CartItem  {
    pro: string[],
}

interface User  {
    userName: string,
}

interface UserProduct extends User , CartItem{}

let userProduct = {
    pro : ['book1' , 'book2'],
    username: 'para',
}
```

دو روش تفاوت خاصی با هم ندارند اما تفاوت هایی که می توان به آن اشاره کرد :

type ها عمومی تر هستند اما تمرکز interface ها بیشتر روی آبجکت ها هست.

در ترکیب دو یا چند تایپ نیز با متفاوت هستند.

تعریف متغیر در TS با مشخص کردن نوع data :

برای اینکه بگوییم داریم تگی را سلکت می کنیم که مربوط به HTML هست بعد از سلکتور و قبل از اسم آیدی یا کلاس و ... از عبارات تعریف شده در TS استفاده می کنیم.

در نظر داشته ما زمانی که می خواهیم از این سلکت ها استفاده کنیم قانونی وجود دارد که می گوید باید از علامت سوال برای متغیر مورد نظر استفاده کنیم زیرا در TS این احتمال داده می شود که این مقادیر ممکن هست null باشند که این موضوع به این معنی هست که یک متغیر ممکن است دو نوع داده داشته باشد که با اصل TS در تضاد است.

با گذاشتن علامت ؟ ما می گوییم اگر null نبود و به این صورت ادامه کد را می نویسیم.

```
const form = document.querySelector<HTMLFormElement>('#para1');
const inp = document.querySelector<HTMLInputElement>('#para2');
const inp1 = document.querySelector<HTMLInputElement>('#para3');
    type Item = {

        name: string,
        email: string,
    }

form?.addEventListener('submit' , (e) => {

    e.preventDefault();

    if( inp?.value == undefined || inp1?.value == undefined ) return



    const userItem : Item = {

        name: inp.value,
        email: inp1.value,

    }

})

function addUser (item: Item  ) {




}
```

در مثال بالا برای inp که ساخته ایم می گوییم اگر خالی نبود value رو برگردون اما اگر خالی بود undefined

Generic Types

جنریک تایپ ها برای زمانی هست که ما می خواهیم تایپ داشته باشیم اما نوع تایپ رو می خواهیم به صورت مختلف در هر جا تغییر بدهیم.

```
type Data<T> = {
    dataType : T[]
}
let text: Data<string> = {
    dataType: [ 'text1' , 'text2' ]
}
```

در اینجا به جای حرف T هر اسم دیگری می تواند باشد و در این مثال ما داریم می گوییم که یک جنریک تایپ داینامیک به اسم Data داریم که نوع آن حتما آرایه هست که به جای حرف T ما میتوانیم انواع تایپ ها را پاس بدهیم که در این مثال string در نظر گرفته ایم.

ما حتی می توانیم یک custom type را در جنریک تایپ ها پاس بدهیم.

```
type Pro = {
    title: string,
    price: number,
}
const info : Data< Pro> = {
    dataType: [ { title: 'A' , price:99} ]
}
```