

Typescript

1. TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.
2. every js file is ts file

Benefit

1. static typing (knows type of variable is determined during compiled type only)
2. code completion
3. refactoring
4. shorthand notation

ts→compiler→js

eg :

let age:number =10; ----> var age =10;

Variable Types

types in TS =types in js + new add add on

JavaScript	TypeScript
● number	● any
● string	● unknown
● boolean	● never
● null	● enum
● undefined	● tuple
● object	

we don't need to annotate each variable with type typescript automatically identify type of variable

```
let a=10 //number type
```

```
let level; // any type
```

```
let number : number[] =[];  
numbers.forEach(n=>n.toString)
```

```
let user :[number, string]=[1,'a'];  
user.push(1);
```

Enum

```
const enum Size {Small=1, Medium, Large};
let mySize:Size= Size.Medium;
console.log(2)
```

Function

```
function(income:number):number{
  if(income<50_000)
    return income *1.2;
  return income * 1.3;
}

// making parameter optional by ?
function (income?:number)
```

Object

```
let employee :{id:number, name:string}={id:1, name: 'b'}
employee.name="r";
```

```
let employee :{id:number, name?:string}={id:1}
employee.name="r";
```

```
let employee :{id:number, readonly name:string}={id:1, name: 'b'}
employee.name="r"; // will give error
```

```
let employee :{id:number, readonly name:string, retire:(date:Date)=>
void }={id:1, name: 'b', retire :(date:Date)=>{console.log(date)}}
```

Type Alias

```
type Employee ={id:number, readonly name:string, retire:(date:Date)=>
void };

let employee:Employee= {id:1, name: 'b', retire
:(date:Date)=>{console.log(date)}};
```

Union

```
function(income:number | string ):number{
  if( typeof income==='number')
    return income *1.2;
  return income * 1.3;
}
```

Intersection

```
type Draggable ={
  drag:()=>> void;
}

type Resizable ={
  height: () => void;
}

type uiWidget = Draggable & Resizable

let textBox : uiWidget={
  drag: ()=>{}
  height :()=>{}
}
```

Literal type

```
type Quantity = 50 |100

let quantity : Quantity =100
```

The optional chaining operator (?.) enables you to read the value of a property located deep within a chain of connected objects without having to check that each reference in the chain is valid.

The ?. operator is like the . chaining operator, except that instead of causing an error if a reference is [nullish](#) ([null](#) or [undefined](#)), the expression short-circuits with a return value of undefined. When used with function calls, it returns undefined if the given function does not exist.

eg:

```
type Customer = {birthday : date}

function getCustomer(id:number): Customer | null | undefined
{
  return (id==0? null : {birthday : new Date()});
}

let customer = getCustomer(0);
console.log(customer?.birthday) // output undefined
```

optional element access operator

```
customer?.[0] // for array
```

optional call

```
let log: any =null

log?.('a');
```