TYPESCRIPT

TypeScript is superset of JavaScript. Big difference: statically-typed. Which means we know the type of varaibles in compile time. TypeScript has type checking. Also other new features to make codes cleaner.

TS uses a complier to translate .ts file to .js file.

```
npm i -g typescript
tsc -v
```

COFIGURATION OF TS COMPILER

```
tsc --init
```

After configurating, we can simply use

```
tsc
```

DEBUGGING TS APPLICATION

First, enable "sourceMap" feature. Then in VSCode debug panel, create a launch.json file. In that file add:

```
"preLaunchTask": "tsc: build - tsconfig.json"
```

BUILTIN TYPES

New Types: any, unknown, never, enum, tuple

In Typescript, we don't always need to annotate the type because TS complier can infer the type of varaibles based on the value.

THE ANY TYPE

"any" can represent any type of values. If we declare a varaible and don't initialize it then TS compiler will assume it is "any".

```
let level;
level=1;
level="a";
// You can do this but it will lose the point of TS!
```

ARRAYS

In JS the elements in an array can be different types, in TS you can explicitly annotate to let them be the same type:

```
let arr: number[] = [1,2,3];
```

Code completion:

```
arr.forEach(n => n.toString());
```

TUPLE

Fix length array with each element has a particular type.

```
let user1:[number,string]=[1,"Zack"];
```

It uses plain JS array under the hood!

ENUM

```
enum Size{Small,Medium,Large};
let mySize:Size=Size.Medium;
```

FUNCTION

All the parameters and return value should be properly annotated.

```
function calculateTax(income:number):number{
  return 0;
}
```

Function default value:

```
function calculateTax(income:number,taxYear=2022):number{
  return 0;
}
```

OBJECT

In JS the shape of object can be changed:

```
let employee={id:1};
employee.name="a";
```

However this is not valid in TS.

Sometimes you want to make some properties read-only, use the "readonly" modifier.

```
let employee:{readonly id:number, name:string}={id:1,name:'Mo'};
```

If you want add function to object, add the arrow signature in the type annotation.

TYPE ALIAS

```
type TypeName={
  all properties
}

type Employee={readonly id:number, name:string};
let emp:Employee={id:1,name:'Mo'};
```

UNION

With union type we can give a varaible more than one type.

```
number | string
// number or string
```

Narrowing: narrow down the unoin type to a specific type.

```
let weight: number | string='10kg';
if(typeof weight === 'number')
  {

  }else{

  }
```

INTERSECTION

Use "&".

```
let a: number & string; // a is both a number and a string.
```

LITERAL TYPES

Sometimes we want to limit the values we can assign to a varaible.

```
// suppose we want the quantity to be either 50 or 100.
let quantity: 50 | 100 = 50;
let quantity2 :50|100 = 51; // not allowed
```

NULLABLE TYPES

Null or undefined objects are common sources of bugs.

Not recommend:

```
let a: number | null = null;
if(a)
   ...
 else
   ...
```

Optional property access operator: use "?." The property will be accessed only the object is not null or undefined. If the object is null then it will return undefined.

When dealing with arrays that can be null or undefined, you can use optional element access operator "?(0)".