JavaScript vs TypeScript. Why We Should Learn TypeScript?

#javascript #typescript #webdev #beginners

What is TypeScript?

Hello, I am a student in Vancouver, Canada and studying web development. In this article, I will show you "**How to use basic typescript data type**" and "**Why we should learn typescript**".

JavaScript is one of the most popular programming languages.
Because that is not strict syntax than other languages. In the other words, JavaScript is more free style language but if you meet a lot of codes in your project, that free style might be changed to pains.

Please look at this code.

```
// Can you guess the return value?
const pikachu = (a, b) => {
  return a + b;
}
```

Can you guess the return value of this code?
I cannot do it. Because pikachu function can take all types of variables like string, number and so on.

```
// Arguments is number.
const pikachu = (a, b) => {
  return a + b;
}
const result = pikachu(2021, 9);
console.log(result); // 2030
```

```
// Arguments is string.
const pikachu = (a, b) => {
  return a + b;
}
const result = pikachu("2021", "9");
console.log(result); // 20219
```

JavaScript is the language of dynamic typing. This is useful for writing codes more easily. But, developers have to take more care of what arguments are needed in the function, what value is returned from the function. More and More you read many codes, you realize this is stressful.

On the other hands, Please look at this code.

```
// Can you guess the return value?
const pikachu = (a: number, b: number): number => {
  return a + b;
}
```

This is typescript code. TypeScript has static typing. When we look at this, we can probably guess pikachu function return value of **number**.
This is so useful to understand codes because companies have a lot of codes and that is so complicated.
So, we should use more readable methods because we take much time to read and understand codes which someone wrote in the past.

Basic Typescript

TypeScript has some primitive data types like string, number, boolean, null, undefined and so on.

This is the code of simple data types.

```
// string, number and boolean.
const caterpie01: number = 2021;    // OK
const caterpie02: number = false;   // NG

const Metapod01: string = "sleepy"; // OK
const Metapod02: string = true;     // NG

const Wartortle01: boolean = true;  // OK
const Wartortle02: boolean = 1111;  // NG
```

We get compiled errors like this.

```
typescript.ts:10:7 - error TS2322: Type 'boolean' is not assignabl

10 const caterpie02: number = false;   // NG
         ~~~~~~~~~~

typescript.ts:13:7 - error TS2322: Type 'boolean' is not assignabl

13 const Metapod02: string = true;     // NG
         ~~~~~~~~~

typescript.ts:16:7 - error TS2322: Type 'number' is not assignable

16 const Wartortle02: boolean = 1111;  // NG
         ~~~~~~~~~~~
```

Next, please think about data type of null and undefined.

```
// null and undefined.
const Butterfree: null = null;
const ButterfreeNull: string = Butterfree;
console.log(ButterfreeNull) // null

const Kakuna: undefined = undefined;
const KakunaNull: string = Kakuna;
console.log(KakunaNull) //undefined
```

This codes works in my environment. We can assign null and undefined value to string value.
In this case, I did not set the strict mode. Once I did assign strict mode to true, this code works like this.

```
typescript.ts:21:7 - error TS2322: Type 'null' is not assignable t

21 const ButterfreeNull: string = Butterfree;
         ~~~~~~~~~~~~~~

typescript.ts:25:7 - error TS2322: Type 'undefined' is not assigna

25 const KakunaNull: string = Kakuna;
```

That is good! We can catch type error.

You can set strict mode in **tsconfig.json** or use **tsc** command argument like **--strict**. If you are not sure how to set up typescript environment, please check this web site.

What is any data type?

TypeScript has **any** data type. It allows all data types to work without type error. This is like vanilla javascript.
Please look at this sample code.

```
// any data type
let pidgey: any = 1991;
console.log(typeof pidgey) // number

pidgey = "bird";
console.log(typeof pidgey) // string

pidgey = false;
console.log(typeof pidgey) // boolean

pidgey = null;
console.log(typeof pidgey) // object

pidgey = undefined;
console.log(typeof pidgey) // undefined
```

pidgey variable can be received all data type!

This is magical data types.🙀
If we use any data type, we do not use TypeScript at all. We just write code by using JavaScript.

TypeScript can guess data types if you do not defined that.
we can replace above sample codes with below codes.

```
// typescript can guess data types.
const caterpie01: number = 2021;     // number
const caterpie001 = 2021;            // number   - typescript gues


const Metapod01: string = "sleepy";  // string
const Metapod001 = "sleepy";         // string   - typescript gues

const Wartortle01: boolean = true;   // boolean
const Wartortle001 = true;           // boolean  - typescript gues
```

This is more readable and shorter. Of course, we cannot assign another data type to this variable.

```
let caterpie001 = 2021;            // number
caterpie001 = "text";              // type error
```

On the other hands, if we do not defined the data type of arguments in function, typescript judge the data type as **any**. Please check this code.

```
const pikachu = (a, b): number => {
  return a + b;
}
pikachu(2021, 9);
```

I got the error like this.(My environment is that strict mode is true. If you turn off strict mode, you can success compile and do not see type error)

```
typescript.ts:57:18 - error TS7006: Parameter 'a' implicitly has a

57 const pikachu = (a, b): number => {
                    ~

typescript.ts:57:21 - error TS7006: Parameter 'b' implicitly has a

57 const pikachu = (a, b): number => {
```

Because typescript cannot guess what values are received.
So, any data type were defined by typescript. When we use function in typescript, we have to defined data types of arguments like this.

```
const pikachu = (a: number, b: number): number => {
  return a + b;
}
```

or

```
// Do not define the return value's data type.
const pikachu = (a: number, b: number) => {
  return a + b;
}
```

If you create function with typescript, you absolutely have to define the specific data type. I recommend we do not have to use any data type anytime except specific situation. For one example, migrating codes from JavaScript to TypeScript.

Object data type

TypeScript can define the object data type with **interface**.
At first, look at this code.

```
// define object data type with interface.
interface PokemonObj {
  name: string,
  age: number,
  skill: string
}
// assign data type to object.
const pokemon: PokemonObj = {
  name: "pikachu",
  age: 6,
  skill: "Electric Shock!"
}
```

We can use **interface** syntax for creating object data type. And then, assign it to object.
If we change data type of object, we get type error like this.

```
// define object data type with interface.
interface PokemonObj{
  name: string,
  age: number,
  skill: string
}
// assign data type to object.
const pokemon: PokemonObj = {
  name: "pikachu",
  age: "change age",       // change
  skill: "Electric Shock!"
}
```

This is type error message.

```
typescript.ts:75:3 - error TS2322: Type 'string' is not assignable

75   age: "change age",
     ~~~

  typescript.ts:69:3
    69   age: number,
         ~~~
    The expected type comes from property 'age' which is declared
```

We get type error. It is useful to define the data type of object with **interface**.
Of course, we can define data type directly like this code.

```
// assign data type directly to object.
const pokemon: {name: string, age: number, skill: string} = {
  name: "pikachu",
  age: 6,
  skill: "Electric Shock!"
}
```

Array data type

Array with data type is like this.

```
// define array data type
const pokemon: string[] = ["pikachu", "Raichu", "Charizard"];
```

If we change the data type, you get type error.

```
// change array data type
const pokemon: string[] = ["pikachu", "Raichu", false];
```

This is type error message.

```
typescript.ts:80:49 - error TS2322: Type 'boolean' is not assignab

80 const pokemon: string[] = ["pikachu", "Raichu", false];
```

This is so useful and powerful because we do not have to take care of data type of each array elements. By the way, I want to show you another way of expression. This is the same as above code. It looks like this.

```
// defined array with another way.
const pokemon: Array<string> = ["pikachu", "Raichu", "Charizard"];
```

As next data type, I will show you generics data type. This is general data type. After we define generics data type, we can define it. Sample code is like this.

```
// defined array with generics data type.
type Pokemon<T> = T[];
// After defined generics type, we can define specific data type.
const pokemon: Pokemon<string> = ["pikachu", "Raichu", "Charizard"

// Above code is the same as this.
const pokemon: string[] = ["pikachu", "Raichu", "Charizard"];
```

We can define some data type with generics data type.
This is not good sample but easy to understand how to use generics data type. Sample is like this.

```
// defined array with generics data type.
type Pokemon<T> = T[];
// After defined generics type, we can define specific data type.
const pokemon01: Pokemon<string> = ["pikachu", "Raichu", "Charizar
const pokemon02: Pokemon<number> = [6, 14, 16];
const pokemon03: Pokemon<boolean> = [true, true, false];
```

What is union?

If you want to use union data type, you can define multiple data type. Please look at this sample code.

```
let pokemon: (string | number) = "pikachu";  // OK
pokemon = 6;
```

This code works correctly because pokemon variable can take string or number data types. But this case is wrong.

```
let pokemon: (string | number) = "pikachu";
pokemon = 6;
pokemon = false;      // NG
```

Because pokemon variable does not take boolean data type and get complied error. If we want to create an array including multiple data type, of course we can use this union data type. This is the sample code.

```
// define data type with array and union
let pokemon: (string | number)[] = ["pikachu", "Raichu", 6, 14];
```

This code is correctly.
But if we add the another data type, we get type error like this.

```
// define data type with array and union
let pokemon: (string | number)[] = ["pikachu", "Raichu", 6, 14, fa
```

This is type error message.

```
typescript.ts:105:65 - error TS2322: Type 'boolean' is not assigna

105 let pokemon: (string | number)[] = ["pikachu", "Raichu", 6, 14
```

If you want to add multiple data type to the array, you can use this union data type.

What is tupple

Tupple is so strict data type.
Beginning, you can check this code.

```
let pokemon: [string, number] = ["pikachu", 6];
```

This code works well. This tupple data type allows only two elements and string and number.
I will show you some wrong case below.

```
typescript.ts:109:36 - error TS2322: Type 'number' is not assignab

109 let pokemon02: [string, number] = [6, "pikachu"];      // NG
                                       ~

typescript.ts:109:39 - error TS2322: Type 'string' is not assignab

109 let pokemon02: [string, number] = [6, "pikachu"];      // NG
                                          ~~~~~~~~~

typescript.ts:110:47 - error TS2322: Type 'string' is not assignab

110 let pokemon03: [string, number] = ["pikachu", "text"]; // NG
                                                  ~~~~~~

typescript.ts:111:5 - error TS2322: Type '[string, number, number]
  Source has 3 element(s) but target allows only 2.

111 let pokemon04: [string, number] = ["pikachu", 6, 14];  // NG
```

Tupple is so strict data type. But it is easy to understand what purpose is this array. It means that the array take only two elements. First, value of string data type. Second, value of number data type.

Conclusion

In this article, I wrote that basic data type of TypeScript.
If you learn typescript, you can check npm packages created by typescript but also write readable and maintained codes.
If you belong to the companies, you know many codes exists there. So, you have to read a lot of codes and understand that. TypeScript helps us understand codes!

This article is just basic knowledge of typescript.
I am planning to write new posts about more typescript data type or React with typescript.
If you are interested in this article, please comment to me!

Thank your for taking your time to read this article.