**TypeScript-Notes**

**Overview**

* TypeScript is a typed superset of JavaScript compiled to JavaScript. In other words, TypeScript is JavaScript plus some additional features.
* TypeScript is aligned with the ECMAScript6 specification.
* JavaScript is TypeScript. This means that any valid .js file can be

renamed to .ts and compiled with other TypeScript files

**Components of TypeScript**

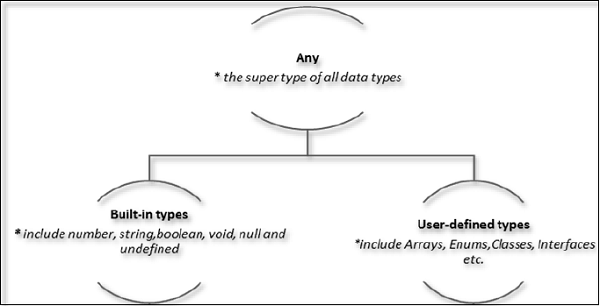
At its heart, TypeScript has the following three components −

* **Language** − It comprises of the syntax, keywords, and type annotations.
* **The TypeScript Compiler** − The TypeScript compiler (tsc) converts the instructions written in TypeScript to its JavaScript equivalent.
* **The TypeScript Language Service** − The "Language Service" exposes an additional layer around the core compiler pipeline that are editor-like applications. The language service supports the common set of a typical editor operations like statement completions, signature help, code formatting and outlining, colorization, etc.

### **Declaration Files**

When a TypeScript script gets compiled, there is an option to generate a **declaration file** (with the extension **.d.ts**) that functions as an interface to the components in the compiled JavaScript. The concept of declaration files is analogous to the concept of header files found in C/C++.

**TYPES**



Built-in types

The following table illustrates all the built-in types in TypeScript −

|  |  |  |
| --- | --- | --- |
| **Data type** | **Keyword** | **Description** |
| Number | number | Double precision 64-bit floating point values. It can be used to represent both, integers and fractions. |
| String | string | Represents a sequence of Unicode characters |
| Boolean | boolean | Represents logical values, true and false |
| Void | void | Used on function return types to represent non-returning functions |
| Null | null | Represents an intentional absence of an object value. |
| Undefined | undefined | Denotes value given to all uninitialized variables |

**Note** − There is no integer type in TypeScript and JavaScript.

**VARIABLE DECLARATION**

|  |  |
| --- | --- |
| **S.No.** | **Variable Declaration Syntax & Description** |
| 1. | **var name:string = ”mary”**  The variable stores a value of type string |
| 2. | **var name:string;**  The variable is a string variable. The variable’s value is set to undefined by default |
| 3. | **var name = ”mary”**  The variable’s type is inferred from the data type of the value. Here, the variable is of the type string |
| 4. | **var name;**  The variable’s data type is any. Its value is set to undefined by default. |

## Type Assertion in TypeScript

TypeScript allows changing a variable from one type to another. TypeScript refers to this process as *Type Assertion*. The syntax is to put the target type between < > symbols and place it in front of the variable or expression. The following example explains this concept −

var s:string = "Mary"

//Type Assertion

var s2:number = <number> <any> s;

// Ab s2 k andar s ki value number type mein convert karke store hogi. This does not mean ki s number mein convert hogya.

## Type Inferring in TypeScript

/\*TypeScript encourages declaring a variable without a type. In such cases,

the compiler will determine the type of the variable on the basis of the value assigned to it.

TypeScript will find the first usage of the variable within the code, determine the type to which it has been initially set

and then assume the same type for this variable in the rest of your code block.\*/

/\* The last line is important \*/

var num = 1; //<-- Compiler infers it as number

//some code

num = "12"; //<--- error: Type not assignable to number

**Functions**

/\* How to wrtite functions in TS

1. Normal function

2. Optional Params

3. Rest params (varargs)

\*/

function printDetails(name:string, id:number){

alert(name + id);

}

printDetails("Megha", 23); //<-- Prints Megha23

printDetails("Megha", "23"); //<-- I tried to change the type. It gives an assigning error

function OptionalParams (name:string, id:number, mailid?:string){

/\*

- If no value is passed, undefined is assumed

- Declaring a value as optional makes it easy/intuitive for us to check for a

possible undefined value for that variable.

\*/

}

//Rest Parameters

/\*Rest parameters don’t restrict the number of values that you can pass to a function.

However, the values passed must all be of the same type \*/

function varArgs(...array: number[]) {

var sum = 0;

for (var i = 0; i < array.length; i++) {

sum = sum + array[i]

}

alert(sum);

}

varArgs(1,2,3,4); //<-- outputs 10

function varArgs2(...array: number[], dummy2:string) {} // Throws an error. Rest params should be the last params in the function

function varArgs3( dummy2:string, dummy3:string, ...array: number[]) {} // Valid. You can have >=1 params before rest. But not after it

//Also the calling should be such that the parameter happens exactly.

**ANONYMOUS FUNCTION AND FUNCTION EXPRESSIONS**

Anon : without name wala function and FE is jab koi function ko variable mein assign kar dette h

var res = function(a:number,b:number) {

return a\*b;

};

console.log(res(12,2))

**FUNCTION CONSTRUCTOR**

var myFunction = new Function("a", "b", "return a \* b");

Did not give much importance

**LAMBDA EXPRESSIONS**

/\* Lamba Expressions\*/

var foo = (x: string) => {

alert(x)

}

foo("Megha") //Alerts Megha

/\*

1. For all functions in TS the parameter type is optional

2. In Lambda, if the params is single the parantheses is optional

Hence,

all the below examples are Valid.

\*/

var foo2 = () => { alert("Empty Params") }

foo2();

var foo3 = x => {alert("Single Params, no brackets" + x)}

foo3(4);

**FUNCTION OVERLOADING**

You need different signature.  
Signature = Data type of params, number of params.