

Type Script Language

Language Basics

- Variables
- Data Types
- Operators
- Statements
- Functions

Variables in TypeScript

- Variables are storage locations in memory, where you can store a value and use it as a part of any expression.
- Variable Configuration comprises of 3 stages
 - Declaration
 - Rendering / Assigning
 - Initialization

Initialization
└─
var x = 10;
└─
Declaration
x = 20;
└─
Rendering
(or)
Assigning

- JavaScript can directly assign or render a value into variable without any declaration if it is not in strict mode.
- If JavaScript is in strict mode the declaring variable is mandatory.
- TypeScript is by default using strict mode of JavaScript.
- In TypeScript declaring variable is mandatory.
- You can declare variables in TypeScript by using 3 keywords
 - var
 - let
 - const

| Keyword | Description |
|---------|---|
| var | <ul style="list-style-type: none"> - It defines a function scope for variable. - You can declare any where in a function and access from any block in the function. <p>Ex:</p> <pre>function f1() { var x = 10; if(x==10) { var y = 20;</pre> |

| | |
|--|---|
| | <pre> } console.log(`x=\${x}\ny=\${y}`); } f1(); </pre> <ul style="list-style-type: none"> - It supports <i>declaration</i>, <i>rendering</i> and <i>initialization</i>. <p>Ex:</p> <pre> function f1() { var x; // declaration x = 10; // rendering if(x==10) { var y = 20; // initialization } console.log(`x=\${x}\ny=\${y}`); } f1(); </pre> <ul style="list-style-type: none"> - It supports shadowing. - Shadowing is the process of re-declaring same name identifier within the given scope. <p>Ex:</p> <pre> function f1() { var x = 0; x=10; if(x==10){ </pre> |
|--|---|

| | |
|-----|---|
| | <pre> var x = 20; // shadowing } console.log(`x=\${x}`); } f1(); </pre> <ul style="list-style-type: none"> - It allows Hoisting. - It is a technique following by compiler to identify the variables declaration before rendering. - Var support hoisting hence there is no order for declaring and rendering. You can first define the rendering block then following by declaration block. <p>Ex:</p> <pre> function f1() { x = 10; console.log(`x=\${x}`); var x; // hoisting } f1(); </pre> |
| let | <ul style="list-style-type: none"> - It defines block scope variable. - It can be accessed only within the block where it is declared. - You can't access outside the block. <p>Ex:</p> |

```
function f1()
{
  let x;
  x=10;
  if(x==10)
  {
    let y = 20; // block scope
    console.log(`x=${x}\ny=${y}`); //
OK
  }
  console.log(`x=${x}\ny=${y}`);
// y not defined
}
f1();
```

- let allows declaration, rendering and initialization.

Ex:

```
function f1()
{
  let x;    // declaring
  x=10;     // rendering
  if(x==10)
  {
    let y = 20; // initialization
    console.log(`x=${x}\ny=${y}`);
  }
}
f1();
```

| | |
|-------|---|
| | <ul style="list-style-type: none"> - let will not allow Shadowing. - You can't re-declare same name identifier in the block. - let will not allow Hoisting. |
| const | <ul style="list-style-type: none"> - It defines block scope variable. - It will allow only initialization. - It will not allow declaring and rendering. - It will not allow shadowing. - It will not allow hoisting. |

- Variable Global scope is defined by declaring variable outside the function.
- You can declare or initialize by using var, let and const for global scope.

Ex:

```
var Name = "TV";
let Price = 45000.55;
const Id = 1;
function f1()
{
  console.log(`F1 -
Name=${Name}\nPrice=${Price}\Id=${Id}`);
}
function f2()
```

```
{
  console.log(`F2 -
Name=${Name}\nPrice=${Price}\Id=${Id}`);
}
f1();
f2();
```

FAQ: Can we define any variable in a function and make it global in access?

A.Yes. Client-Side global scope for variable in function is defined by using “window” object. “window” is a browser object.

Ex:

```
function f1()
{
  window.productName = "Samsung TV";
}
function f2()
{
  document.write("Name=" +
window.productName);
}
f1();
f2();
```

- Variable Naming Conventions

- Variable name must start with an alphabet or underscore “_”.
- It can be an alpha numeric name. It can contain numbers.
- But it can't start with a number.

```
let productname = "TV";      // valid
let _productname = "TV";    //valid
let product2020 = "TV";      //valid
let product_2020 = "TV";    //valid
let 2020_product = "TV";    // invalid
```

[underscore is used to define that the declared content is marked for implementation, which means that its definition is incomplete [not implemented].

- Variable name can't exceed 255 chars.
- Variable name is case sensitive.


```
let Name = "John";
let name = "sam";
console.log(name); // sam
```
- Variable name must be defined in “Camel Case”.
- Variable must speak, what it is.

Data Types

- The term data type is derived from “Data Structure”.
- In memory to store data we need a schema [structure] for data.
- Data structure defines the size of value and type of value.
- Data Type determines the type and size of value that can stored in memory.
- JavaScript is implicitly typed. The data type will be determined according to the value assigned. The data type is **not a strongly typed** in JavaScript. After initializing a specific type of value into memory you can store any contradictory value.

EX:

```
<script>
    function f1(){
        var x = 10;
        x = "john";
        x = true;
        document.write(`x is ${typeof x} type`);
    }
    f1();
</script>
```

- TypeScript is strongly typed. Once the data type is defined it will not allow any contradictory data.

Syntax:

```
let variableName: DataType;  
let price:number = 34000;  
price = "TV"; // invalid
```

- The latest versions of TypeScript support "Type Inference".
- Type Inference identifies the data type of value you initialized into variable and configures the data type as initialized type.

Ex:

```
let price = 34000;  
price = 50600;  
price = "TV"; // invalid
```

TypeScript Data Types

- **"any"** is the root type for handling all types of values in TypeScript.
- The TypeScript data type are classified into 2 groups
 - Primitive Data Types
 - Non-Primitive Data Types

Primitive Data Types

- These are Immutable types.
- The variable and its state can't be changed after defining.
- The memory allocated for a variable can't change dynamically.
- Primitive types are allocated with "Stack" memory. [LIFO]
- **Primitive types have a fixed range for storing value.**
- TypeScript primitive types are
 - number
 - string
 - boolean
 - null
 - undefined

Non-Primitive Data Type

- These are Mutable types.
- The variable and its state can be changes after defining.
- The memory allocated for a variable can change dynamically.
- Non-Primitive types are allocated with memory heap.

- The range of value varies according to the memory available.
- They don't have any fixed range for storing value.
- TypeScript non-primitive types are
 - Array
 - Object
 - Regular Expression

TypeScript Primitive Data Types

- **number**
- **string**
- **boolean**
- **null**
- **undefined**

Number Type

- Number type is defined by using "number" keyword.
- It is used to handle numeric values.
- TypeScript number type can handle
 - **Signed Integer**
Ex:
let a:number = +10;
let b:number = -10;

```
console.log("a=" + a);  
console.log("b=" + b);
```

- **Un Signed Integer**

Ex:

```
let a:number = 10;  
console.log("a=" + a);
```

- **Floating Point value**

- **Double value**

- **Decimal value**

Ex:

```
let float:number = 45.54;  
let double:number = 456.563;  
let decimal:number = 456.496997979548382;  
// 29  
console.log("float=" + float);  
console.log("double=" + double);  
console.log("decimal=" + decimal);
```

- **Exponent value**

Ex:

```
let exponent:number = 2e3; // 2000    2 x 103  
console.log("exponent=" + exponent);
```

- **Binary value**

Ex:

```
let binary:number = 0b1010;  
console.log("binary=" + binary);
```

- **Octal value**

Ex:

```
let octal:number = 0o744;  
console.log("octal=" + octal);
```

- **Hexadecimal value**

Ex:

```
let hexa:number = 0xf00d;  
console.log("hexa=" + hexa);
```

Note: TypeScript is case sensitive. ‘Number’ and ‘number’ are different. Data Type is defined with lowercase.

String Type

- String is a literal with group of characters [alphabet, number, special chars] enclosed in
 - Double Quotes [“ ”]
 - Single Quotes [‘ ’]
 - Back Tick [` `] – new from ES5
- String type is defined by using “string” keyword

Syntax:

```
let variableName:string = “ab123$”;
```

Ex:

```
let str1:string = "john123$";
```

```
let str2:string = 'john123$';
```

```
let str3:string = `john123$`;
```

```
console.log(str1);
```

```
console.log(str2);
```

```
console.log(str3);
```

- String can swap between inner and outer strings by using double and single quotes.
 - Outer String defined in double quote then inner string is defined with single quote.
 - Outer String defines in single quote then inner string is defined with double quote.
 - TypeScript with angular prefers Outer string in single quotes.
 - TypeScript with other languages prefers Outer string in double quotes.
 - “tsLint.json” defines rules for language.

Syntax:

```
let link1:string = "<a href='home.html'>Home</a>";
```

```
let link2:string = '<a href="about.html">About</a>';  
document.write(link1 + "<span>|</span>" + link2);
```

- TypeScript supports Back Tick [` `] that defines a string embedded with expression.
[In general string is concatenated with expression]
- Back Tick allows to embed any expression into a string using binding expression notation **`${}`**

Syntax:

```
let variableName:string = `Your Text ${your  
expression} Your Text continues`;
```

Ex:

```
let age:number = 22;
```

```
let uname:string = "John";
```

```
console.log("Hello !" + " " + uname + " " + "You will be"  
+ " " + (age+1) + " " + "Next Year.");
```

```
console.log(`Hello ! ${uname} You will be ${age+1}  
Next Year`);
```


- Back Tick allows to configure a template for presentation. Template comprises of logic and presentation.

Ex:

- Create a new typescript file “vardemo.ts”

```
let title:string = "User Login";
let login:string =
`
<h2>${title}</h2>
<dl>
<dt>User Name </dt>
<dd><input type="text"></dd>
<dt>Password</dt>
<dd><input type="password"></dd>
<dd> <button>Login</button> </dd>
</dl>
`;
document.write(login);
```

- Transcompile into JavaScript file
- Create a new HTML file “index.html”
<script src="Variables/vardemo.js"></script>

String Escape Issues

- Several characters defined in a string are not printable. Due to compiler inability to compile, as they are reserved characters in typescript language.
- To print the non-printable character, you have to use “\” meta character.

Ex:

```
let path:string = "C:\\Images\\Cars\\audi.jpg";
```

```
let message:string = "\"Hello ! Welcome\"";
```

```
console.log(`Path = ${path}`);
```

```
console.log(message);
```

String Manipulation