**TypeScript - Variable**

Variables can be declared using: var, let, and const.

same rules of java script

**var**

Variables in TypeScript can be declared using var keyword, same as in JavaScript. The scoping rules remains the same as in JavaScript.

**let**

ES6 introduced two new types of variable declarations in JavaScript, using the keywords let and const. TypeScript, being a superset of JavaScript, also supports these new types of variable declarations.

**Variable Declaration**

var [identifier] : [type-annotation] = value;

var [identifier] : [type-annotation];

var [identifier] = value;

var [identifier];

let employeeName = "ravi"; or let employeeName:string = "ravi";

The let declarations follow the same syntax as var declarations.

variables declared with let have a block-scope.

e.g. function, if else block or loop block

let num1:number = 1;

function letDeclaration() {

let num2:number = 2;

if (num2 > num1) {

let num3: number = 3;

num3++;

}

while(num1 < num2) {

let num4: number = 4;

num1++;

}

console.log(num1);

console.log(num2);

console.log(num3); //Compiler Error: Cannot find name 'num3'

console.log(num4); //Compiler Error: Cannot find name 'num4'

}

letDeclaration();

**Advantages of using let over var**

1)The var keyword was introduced with JavaScript.

The let keyword was added in ES6 (ES 2015) version of JavaScript.

2) Block-scoped let variables cannot be read or written to before they are declared.

console.log(n1);

var n1:number=10;

console.log(n1);

let n1:number=10;

3)Let variables cannot be re-declared

var num:number = 5;

var num:number = 6;

let num:number = 5;

let num:number = 6;

4)var can be declared globally and can be accessed globally.

let can be declared globally but cannot be accessed globally.

5)var has global scope.

let has limited block scope.

**Const - Variable**

The const declaration is used to declare permanent value, which cannot be changed later. It has a fixed value. The const declaration follows the same scoping rules as let declaration, but we cannot re-assign any new value to it.

**TypeScript Data Type - Number**

The number is used to represents both integers as well as floating-point values.TypeScript also supports Binary(Base 2), Octal(Base 8), Decimal(Base 10), and Hexadecimal(Base 16) literals.

let first: number = 12.0; // number

let second: number = 0x37CF; // hexadecimal

let third: number = 0o377 ; // octal

let fourth: number = 0b111001; // binary

|  |  |  |
| --- | --- | --- |
| **Sno** | **Property Name** | **Description** |
| 1 | MAX\_VALUE | It returns the largest possible value of a number in JavaScript and can have 1.7976931348623157E+308. |
| 2 | MIN\_VALUE | It returns the smallest possible value of a number in JavaScript and can have 5E-324. |
| 3 | NEGATIVE\_INFINITY | It returns a value that is less than MIN\_VALUE. |
| 4 | POSITIVE\_INFINITY | It returns a value that is greater than MAX\_VALUE. |
| 5 | NaN | When some number calculation is not representable by a valid number, then TypeScript returns a value NaN. It is equal to a value that is not a number. |
| 6 | Prototype | It is a static property of the Number object. It is used to assign new properties and methods to the Number object in the current document. |

**Number Methods**

|  |  |  |
| --- | --- | --- |
| **Sno** | **Property Name** | **Description** |
| 1 | toExponential() | It is used to return the exponential notation in string format. |
| 2 | toFixed() | It is used to return the fixed-point notation in string format. |
| 3 | toLocaleString() | It is used to convert the number into a local specific representation of the number. |
| 4 | toPrecision() | It is used to return the string representation in exponential or fixed-point to the specified precision. |
| 5 | toString() | It is used to return the string representation of the number in the specified base. |
| 6 | valueOf() | It is used to return the primitive value of the number. |

let myNumber: number = 123456;

let fixedNumber: number = 10.8788;

let localNumber: number = 10667.987;

let preNumber: number = 10.5679;

let mNumber: number = 123;

let vNumber = new Number(123);

myNumber.toExponential();

myNumber.toExponential(1);

fixedNumber.toFixed();

fixedNumber.toFixed(1);

myNumber.toLocaleString(); // US English

myNumber.toLocaleString('de-DE'); // German

myNumber.toLocaleString('ar-EG'); // Arebic

myNumber.toPrecision(1);

myNumber.toPrecision(2);

mNumber.toString();

mNumber.toString(2);

mNumber.toString(4);

console.log(vNumber)

console.log(vNumber.valueOf())

**TypeScript - String**

String is another primitive data type that is used to store text data. String values are surrounded by single quotation marks or double quotation marks

let employeeName:string = 'Rama krishna'; OR let employeeName:string = "Ravi Ram";

**String methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| charAt() | Returns the character at the given index |
| concat() | Returns a combination of the two or more specified strings |
| indexOf() | Returns an index of first occurrence of the specified substring from a string (-1 if not found) |
| replace() | Replaces the matched substring with a new substring |
| split() | Splits the string into substrings and returns an array |
| toUpperCase() | Converts all the characters of the string into upper case |
| toLowerCase() | Converts all the characters of the string into lower case |
| charCodeAt() | Returns a number that is the UTF-16 code unit value at the given index |
| codePointAt() | Returns a nonnegative integer Number that is the code point value of the UTF-16 encoded code point starting at the specified index |
| includes() | Checks whether a string includes another string |
| endsWith() | Checks whether a string ends with another string |
| LastIndexOf() | Returns the index of last occurrence of value in the string |
| localeCompare() | Checks whether a string comes before, after or is the same as the given string |
| match() | Matches a regular expression against the given string |
| normalize() | Returns the Unicode Normalization Form of the given string |
| padEnd() | Pads the end of the current string with the given string |
| padStart() | Pads the beginning of the current string with given string |
| repeat() | Returns a string consisting of the elements of the object repeated in the given times. |
| search() | Searches for a match between a regular expression and a string |
| slice() | Returns a section of a string |
| startsWith() | Checks whether a string starts with another string |
| substr() | Returns a string beginning at the specified location and of the given characters |
| substring() | Returns a string between the two given indexes |
| toLocaleLowerCase() | Returns a lower case string while respecting current locale |
| toLocaleUpperCase() | Returns an upper case string while respecting current locale |
| trim() | Trims the white space from beginning and end of string |
| trimLeft() | Trims the white space from left side of the string |
| trimRight() | Trims the white space from right side of the string |

.

**TypeScript Data Type - Boolean**

Boolean values are supported by both JavaScript and TypeScript and stored as true/false values.

let isValid:boolean = true;

**TypeScript - Arrays**

An array is a special type of data type which can store multiple values of different data types sequentially using a special syntax.

**Characteristics of an Array**

An array stores elements which have the same data type.

Array elements stored in contiguous memory locations.

The storage of 2-D array elements is rowed by row in a contiguous memory location.

Array name represents the address of the starting element.

The size of an array should be initialized at the declaration time.

Array size should be a constant expression and not a variable.

We can retrieve array elements by specifying the element's corresponding index value.

**Advantage**

Code Optimization: An array helps to make the code optimized, which increases the speed and performance of the program. It allows us to retrieve or sort the array data more efficiently.

Random access: It provides the ability to access any data of an array in constant time (independent of its position and size). Thus, we can get any data of an array located at any index position directly.

**Disadvantage**

Size Limit: An array allows us to store only the fixed number of elements. Once the array is declared, we cannot alter its size. Hence, if we want to insert more element than declared, it is not possible.

There are two ways to declare an array:

1. Using square brackets. This method is similar to how you would declare arrays in JavaScript.

let fruits: string[] = ['Apple', 'Orange', 'Banana'];

2. Using a generic array type, Array<elementType>.

let fruits: Array<string> = ['Apple', 'Orange', 'Banana'];

**Array Object**

Array objects allow us to store multiple values in a single variable. We can create an array by using the Array object

**declaring Array Object**

A numeric value which represents the size of an array or

A list of comma-separated values.

Syntax

let arr\_name:datatype[] = new Array(values);

let test:sumber[] = new Array(5);

**Methods in array**

---------------

Method Description

pop() Removes the last element of the array and return that element

push() Adds new elements to the array and returns the new array length

sort() Sorts all the elements of the array

concat() Joins two arrays and returns the combined result

indexOf() Returns the index of the first match of a value in the array (-1 if not found)

copyWithin() Copies a sequence of elements within the array

fill() Fills the array with a static value from the provided start index to the end index

shift() Removes and returns the first element of the array

splice() Adds or removes elements from the array

unshift() Adds one or more elements to the beginning of the array

includes() Checks whether the array contains a certain element

join() Joins all elements of the array into a string

lastIndexOf() Returns the last index of an element in the array

slice() Extracts a section of the array and returns the new array

toString() Returns a string representation of the array

toLocaleString() Returns a localized string representing the array

**TypeScript - Tuples**

TypeScript introduced a new data type called Tuple. Tuple can contain two values of different data types.

var empId: number = 1;

var empName: string = "Steve";

var employee: [number, string] = [1, "Steve"];

employee[0]; // returns 1

employee[1]; // returns "Steve"

**TypeScript Data Type - Enum**

Enums or enumerations are a new data type supported in TypeScript.

It is used to define the set of named constants,

**Why Enums?**

It makes it easy to change values in the future.

It reduces errors which are caused by transporting or mistyping a number.

It exists only during compilation time, so it does not allocate memory.

It saves runtime and compile-time with inline code in JavaScript.

It allows us to create constants that we can easily relate to the program.

It will enable developers to develop memory-efficient custom constants in JavaScript, which does not support enums, but TypeScript helps us to access them.

There are three types of Enums in TypeScript. These are:

TypeScript Enums

Numeric Enums

String Enums

Heterogeneous Enums

Numeric enums are number-based enums, which store values as numbers. It means we can assign the number to an instance of the enum.

enum Direction {

Up = 1,

Down,

Left,

Right,

}

console.log(Direction);

Enums are always assigned numeric values when they are stored. The first value always takes the numeric value of 0, while the other values in the enum are incremented by 1.

String Enum

String enums are similar to numeric enums, except that the enum values are initialized with string values rather than numeric values.

enum PrintMedia {

Newspaper = "NEWSPAPER",

Newsletter = "NEWSLETTER",

Magazine = "MAGAZINE",

Book = "BOOK"

}

PrintMedia.Newspaper;

PrintMedia['Magazine'];

Heterogeneous Enum

Heterogeneous enums are enums that contain both string and numeric values.

enum Status {

Active = 'ACTIVE',

Deactivate = 1,

Pending

}

Reverse Mapping

Enum in TypeScript supports reverse mapping. It means we can access the value of a member and also a member name from its value.

enum PrintMedia {

Newspaper = 1,

Newsletter,

Magazine,

Book

}

PrintMedia.Magazine;

PrintMedia["Magazine"];

PrintMedia[3];

**TypeScript - Union**

TypeScript allows us to use more than one data type for a variable or a function parameter.

Syntax:

(type1 | type2 | type3 | .. | typeN)

let code: (string | number);

code = 123;

code = "ABC";

code = false;

let empId: string | number;

empId = 111;

empId = "E111";

empId = true;

**TypeScript Data Type - Any**

Any data type to handle any type of datatypes

let s:any ="hello"

s=435

s=boolean

any type Array Copy

let arr: any[] = ["Ravi", 212, true];

arr.push("Ram");

console.log(arr);

**TypeScript Data Type - Void**

Similar to languages like Java, void is used where there is no data

function does not return any value then you can specify void as return type.

function sayHi(): void {

console.log('Hi!')

}

let speech: void = sayHi();

console.log(speech);

**TypeScript Data Type - Never**

TypeScript introduced a new type never,

The never type is used when you are sure that something is never going to occur.

you are write a function which will not return to its end point or always throws an exception.

function throwError(errorMsg: string): never {

throw new Error(errorMsg);

}