

**Typescript**

Contents

**[1](#_Toc493238773)****[Basic Types (Primitive)](#_Toc493238773)** [3](#_Toc493238773)

**[2](#_Toc493238774)****[Composite Types](#_Toc493238774)** [3](#_Toc493238774)

**[3](#_Toc493238775)****[Interface Type](#_Toc493238775)** [3](#_Toc493238775)

1. **Basic Types (Primitive)**

|  |  |
| --- | --- |
| Type | Value |
| boolean | True/false |
| number | Floating point values, Supports  Binary (prefix ‘0b’), Decimal (6, 60), Octal (prefix ‘0o’) and hexadecimal (prefix ‘0x’) |
| string | Supports Single quote, double quote for constant strings and backtick/backquote ( ` ) for template strings which supports embedded expression of the form ${ expr } |
| void | Represents no return type when used in function return types |
| enum | Enumerator like in C#, By default the start member has value 0. |
| null | It is an assignment value. Null is an object type |
| undefined | Means a variable is declared but no value is assigned yet |
| never \* | It represents the type of values that never occur. An internal type for typescript compiler, we cannot actually see the type as it is unreachable to end user |
| symbol | A symbol value may be used as an identifier for object properties. |

1. **Composite Types**

|  |  |
| --- | --- |
| Type | Value |
| Array, [] | Has dedicated type syntax for arrays |
| Tuple | With tuples one can define type of data at every position of array |
| any | A type which can hold any other type. |
| enum | A friendly datatype to define a set of numeric values with names |
| ReadonlyArray | Same as Array except works as a readonly property, once created, the values cannot be modified |
| Math |  |
| Date |  |
| RegExp |  |
| Union Types |  |

**Hands On**

1. **Interface Type**

Contracts within a code, blueprint

// Declaration

// With name

Interface IExample {

Var1: string,

Var2: number

}

// Without name

function(iobj : { x: number, y: number }): void {}

function(arg: string): { x: number, y: number } {}

// Optional properties using ‘?’

// readonly property (Only assigned when created)

interface IExample {

readonly var1: string

}

// Interface as function types

interface Method {

(var1: number, var2: string): boolean;

}

// Indexable types

interface IIndexable {

[index: number]: string

}

let array: IIndexable = [“hello”, “bob”];

array[0]; //hello

// interface inheritance

// Class inheriting interfaces

// interface work both as function and object

// Construct signature

**NOTE: There are two types of supported index signatures: string and number**

**NOTE: variable use const , while properties use readonly**

1. **Variable Declarations**

|  |  |
| --- | --- |
| **Declarator** | **Meaning** |
| var | * Declare without type specifier * Scope:   + Access within other functions (Closure Scope)   + Global scope   + Local scope   + Intermediate   function four() {  if (true) {  var a = 4;  }  alert(a); // alerts '4', not the global value of '1'  }  var declarations are accessible anywhere within their containing function, module, namespace, or global scope.  Multiple declaration of same variable is possible and refer to same object. |
| let | It has block scoping. Shadowing – Declaring same variable in different block is allowed and is consider a separate variable |
| const | They behave similar to let. The value cannot be changed once declared. |

* 1. **Destructuring**

**Hands On**

**let** input = [1, 2];

**let** [first, second] = input;

// Swap variables

[var1, var2] = [var2, var1];

// function parameter

function f([var1, var2] : [number, number]) {}

f([1,2])

// capture remaining items (…) three dots

let [var1, …vars] = [1, 2, 3, 4];

// Capture partial

let [var1] = [1, 2,3];

let [,var2, , var4] = [1, 2, 3, 4, 5];

// Destructure object

let objData = { x: 3, y: 4, name: “axis”};

let {x, y} = objData;

let {x, …rest} = objData;

rest.y, rest.name

// rename the properties

let { x1: x, y1: y} = objData;

This renaming the variables

* 1. **Spread**

**Hands On**

Opposite of Destructuring

// Spread an array into others

let array1 = [1, 2];

let array2 = [3, 4];

let array3 = [5, …array1, 10, …array2, 20];

// Spread of objects with defaults

let defaultData = { name: “bob”, price: 5};

let objData = { …defaultData, price: 9.5}; // overwrites the default price as it comes later, position is important

// It only includes an object’s properties and not methods

// generics not allowed

* 1. **Defaults**

**Hands On**

// function optional arguments

function({a: string, b?: number}): void {}

function({a, b = 0}): void {}

function({a, b = 0} = {a: “”}): void {}

1. **Index Signature**

// An interface can be defined to have unknown properties as

interface IWithUnknwon {

var1: string,

[varn: string]: any

}

let iobj: IWithUnknwon = {var1: “hello”, val1: 34, var3: “Next hello”};

1. **Class Type**

// Class structure

// Constructor

// inheritance

//Default access

// private modifier

// readonly

// getter and setter

// get but no set == readonly

// static

// abstract

// similarity of class and interface

// A class can extend another class

// A class can also implements another class(s) as interface

1. **Module Type**
2. **Namespace Type**
3. **Import Type**
4. **New Concept ES5 Type**

Decorators

1. **Import Type**
2. **Declaration File**