Build your app for production.

Consider AoT compilation

A typescript compiler compiles the .ts file

The TSC generates a JavaScript version of the **.ts** file passed to it. In other words, the TSC produces an equivalent JavaScript source code from the Typescript file given as an input to it. This process is termed as transpilation.

Npm install typescript -g

Install Microsoft visual studio code

Right click on the file and open with cmd in the editor.

Basic syntax :

var abc : string = ‘Mohan’;

after compilation it will create :

var abc = ‘Mohan’;

comment in typescript :

//

/\*

\*/

Object orientation:

class Animal{

show():void{

console.log(‘All is well’);

}

}

var ob = new Animal();

ob.show();

**Different types of 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 |

User-defined Types

User-defined types include Enumerations (enums), classes, interfaces, arrays, and tuple

Type assertion :

Here we are going to change the type of any variable :

**Decision making :**

var a: number = 5;

if (a > 0) {

console.log('Number is +ve')

}

**Switch**

var grade: string = 'A';

switch (grade) {

case 'A':

console.log('Very good ')

break;

case 'B':

console.log('Not Good')

default:

}

**Loop**

var i: number;

for (i = 0; i < 20; i++){

console.log('The value is '+i)

}

**Function**

Defining a function :

function <function\_name>(){}

Calling a function

<function\_name>();

Returning value from function :

function <function\_name>() : <return\_type>{

return value;

}

Parameters in function :

Call by value

This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument

Call by pointer

This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter

Define a function :

**Positional parameters**

function add(a:number, b:number){

console.log('The sum is '+(a+b))

}

add(12, 12);

Note : If we do not set any type then brdefault type will be <any>

## Optional Parameters

Optional parameters can be used when arguments need not be compulsorily passed for a function’s execution. A parameter can be marked optional by appending a question mark to its name. The optional parameter should be set as the last argument in a function

function add(a:number, b:number,c ?: number){

console.log(a+'--'+b+'--'+c)

}

add(12, 12);

**Rest parameters :**

function addNumbs(...nums:number[]){

var i: number ;

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

console.log(nums[i])

}

}

addNumbs(1,2,3,4,5)

Default parameters :

function add(a:number,b:number=12) {

console.log(a+'--'+b)

}

add(10)

Interface :

An interface is a syntactical contract that an entity should conform to. In other words, an interface defines the syntax that any entity must adhere to.

Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members. It often helps in providing a standard structure that the deriving classes would follow.

interface Person {

first\_name: string,

last\_name : string

}

var pr : Person {

first\_name: 'Mohan',

last\_name : 'Kumar'

}

console.log(pr)