Typescript

Drawback of JavaScript

* It is not a typesafe language, a variable can accept any kind of value, it might lead to unexpected result
* It is directly executed, which would not show error before execution, it shows the error only at the time execution

Typescript

* It is a typed JavaScript means it adds types to variables & functions
* It is a super set of JavaScript because it supports all the features of JavaScript and some extra features to improve the development experience
* It is more type safer than JavaScript
* It is Compiled first and then executed, the typescripts are converted to JavaScripts when you compile, you can execute the JavaScripts generated from typescripts
* You can work on typescripts without working in Javascript
* It follows the same syntax of JavaScript

In JavaScript

function add(x, y) { }

You can call add by add(20, 30); add(20, “hello”), add(true, false) and so on, this would lead to unexpected output and also we can’t stop programmers passing specific type of value

In TypeScript

function add(x: number, y: number) { } // x & y are type of number, now the add() accepts only numbers

add(20, 30); // no error

add(“hello”, 30); // compilation error

In JavaScript a function can return any type of value

function add(x, y) { return x + y; }

Here return x + y can’t always be number, it can be string also

In TypeScript you can mention what function must return

function add(x : number, y : number): number {   
 return (x + y)   
}

Here the add() function returns number, if you try to return a string or a boolean, typescript throws compilation error

Typescript has below types

* number
* string
* void
* boolean
* any
* []

Along with this it supports complex types and union types

Complex types: You can return a class type

Union types: Multiple types

function getUser() : User {  
  
}

The function getUser() returns User type

function getResult(): number | string {   
  
}

The function getResult() returns either number or string, this is union type

function getData(): any {  
  
}

The function getData() returns any type, means it can be anything

Does browser understand Typescript?

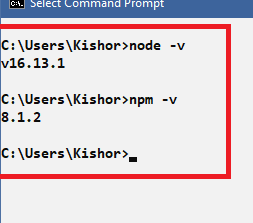
* NO

You must use Typescript to avoid development errors and convert it to Javascript through typescript compiler and run the javascript on the browser

Typescript compiler

It is a software to compile typescript files, which can be downloaded using npm command that comes from node.js

npm: Node package Manager, which is a command line tool to download the third party javascript libraries from the internet, these javascript libraries will be stored in node\_modules folder.



How to install typescript

>> npm install typescript

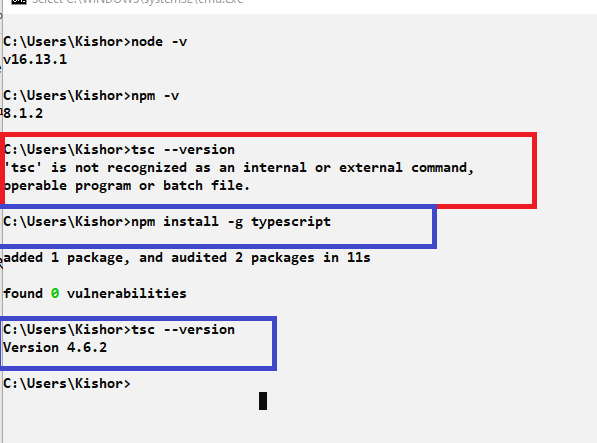
(or)

>> npm install -g typescript

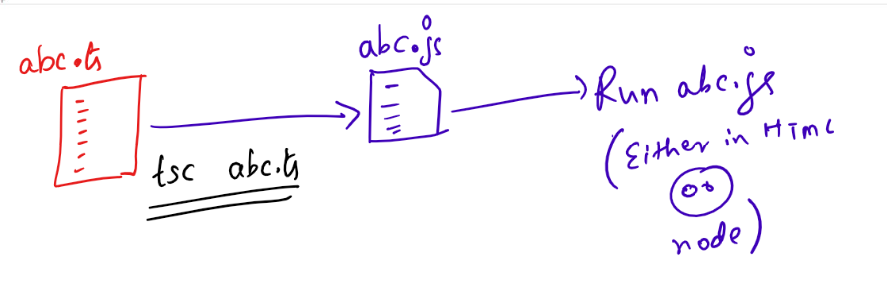
-g means, global installation so the typescript compiler will be available in any location of your machine, without -g, typescript compiler will be available only at the location where you entered npm install

Verify typescript installation

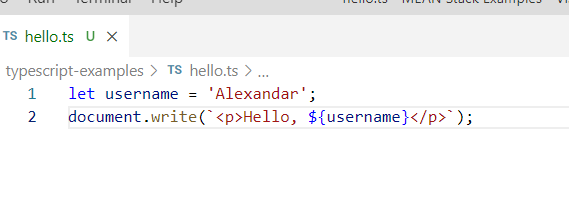
>> tsc --version



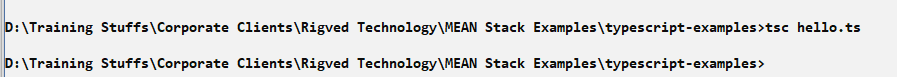
How typescript must be used



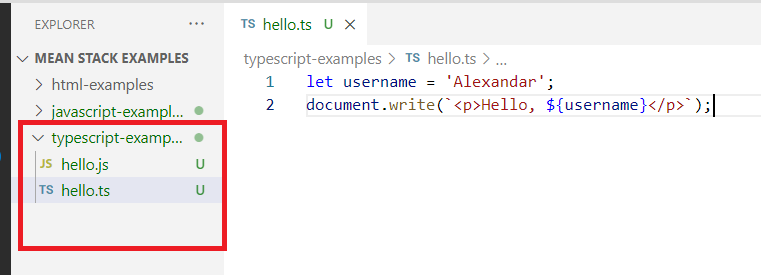
hello.ts



Compilation of hello.ts

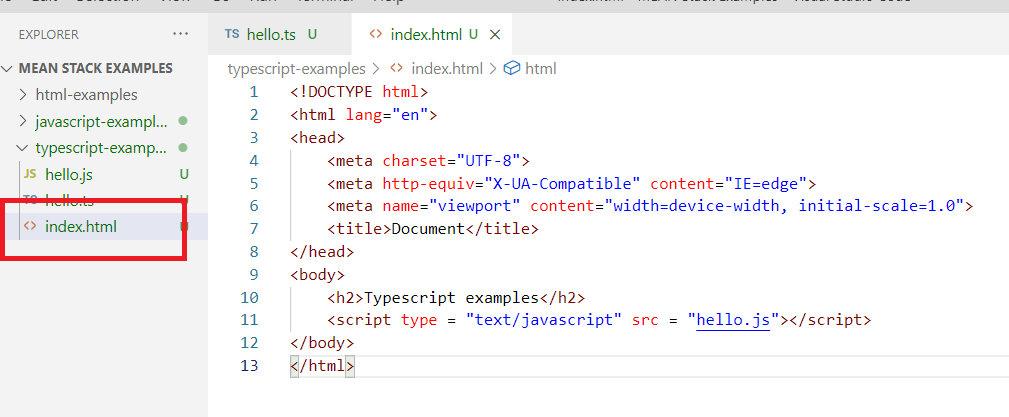


You can notice hello.js



Now you can include hello.js file in the HTML to see the output

index.html

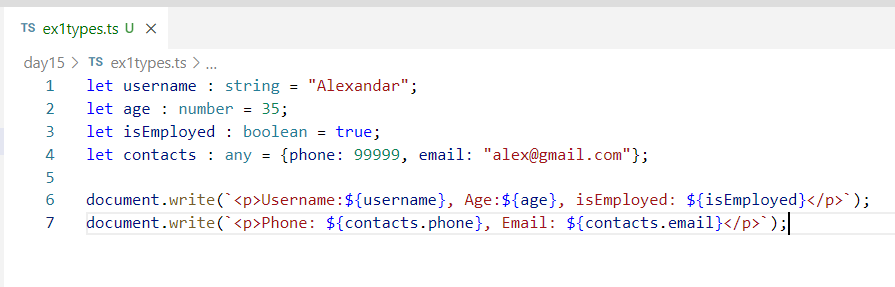


Output:

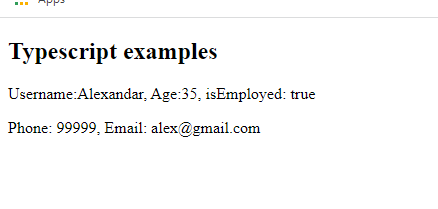


Adding some types to the script

ex1types.ts



Add the converted javascript to html and output would be



Tuples:

It allows you to express array with fixed set of values

let x : [string, number, string];

here x must have values that matches to the tuples

i.e., x = [“hello”, 30, “world”]; // this is ok

x = [“hello”, 20, “world”, “hi”]; // this leads to an error

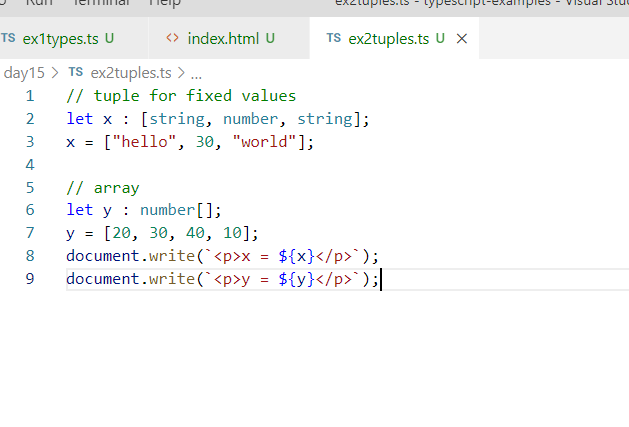
Functions with return types

This allows to mention the type of value a function would return

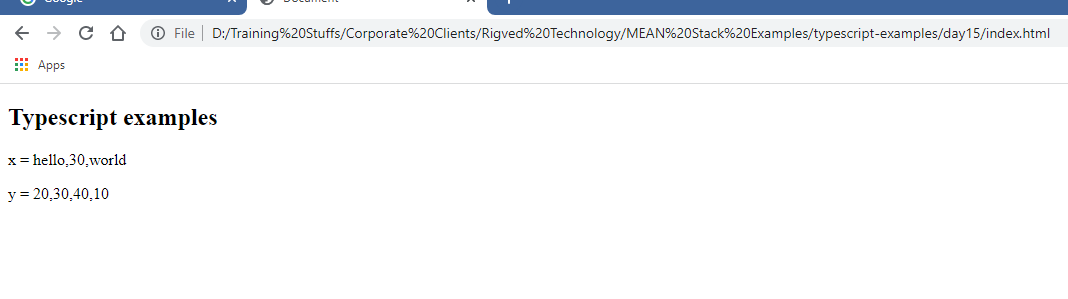
function fun\_name(): return\_type { }

return\_type can be number, string, any, boolean, void etc.

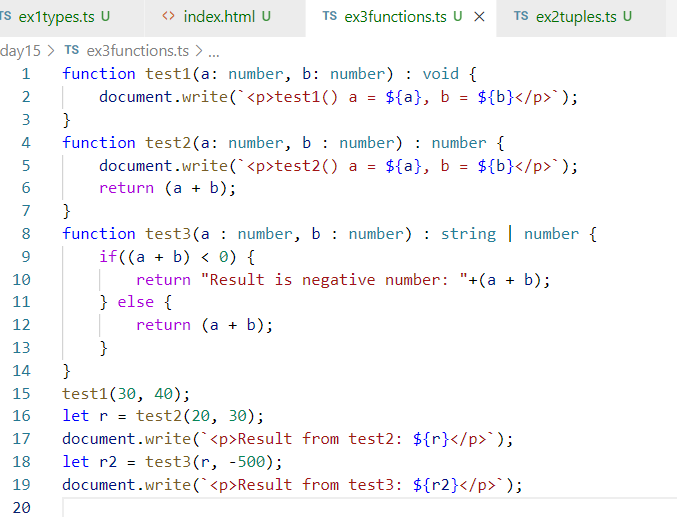
ex2tuples.ts

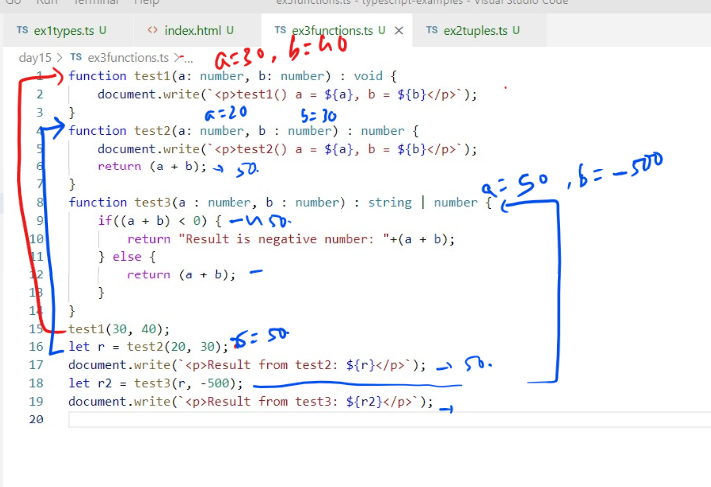


Output:

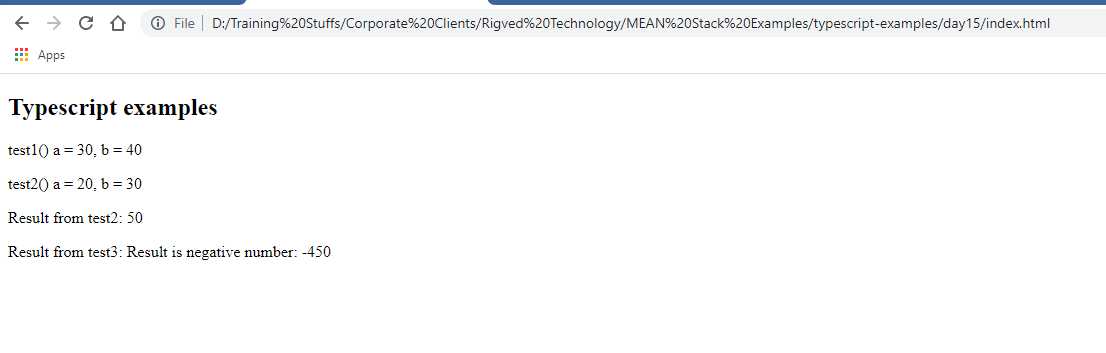


ex3functions.ts



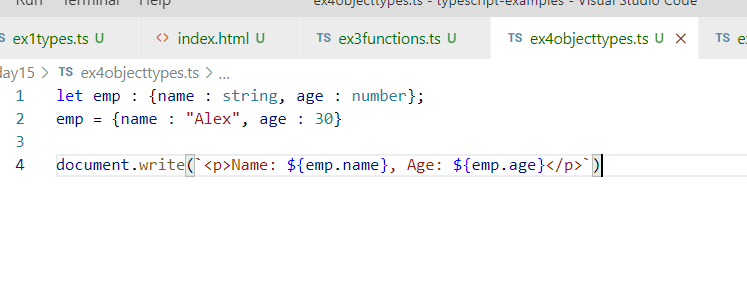


Output:



Specify types to the object

ex4types.ts



Output:



Typescript supports all the javascript features even the latest features of JavaScript’s like:-

let, const, class, constructor, super, extends, arrow, template, rest & spread, optional chain, exponential, generators, closures and so on.

With JavaScript you can create rest operator as below:

function sum(x, …y) {

…..  
 let value = // will have some result by iterating y  
 return value;  
}

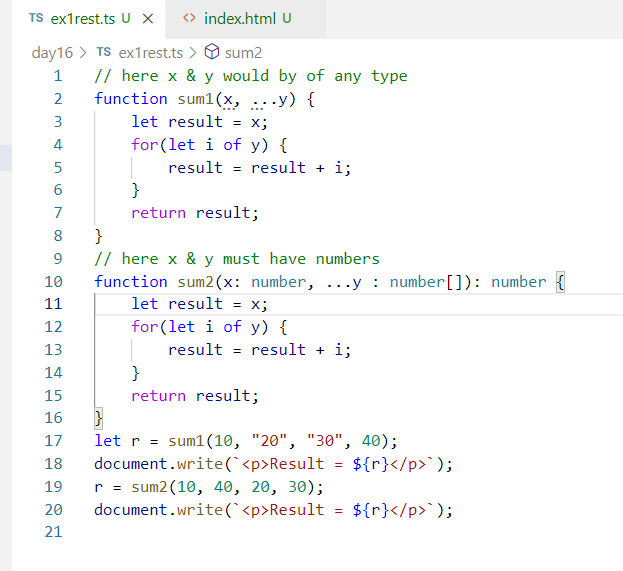
sum(10, 20, 30, 40, 50); // works

sum(10, 20, “hello”, “world”); // works without any problem, but you don’t get the proper result you may get NaN

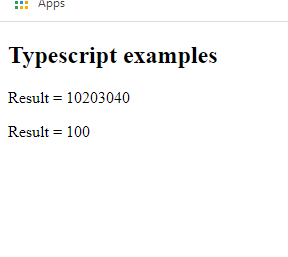
But in Typescript you can create rest operator can accept what kind of values

function sum(x: number, …y: number[]) {} // now the y variable can accept only list of numbers, other types if you pass typescript compiler gives error.

ex1rest.ts



Note: Compile this file & add the js file to the html and load that file



private, public, protected & Read-only variables

private: These are members accessible within the class, but not visible outside the class

public: These are members accessible within the class as well as outside the class

protected: These are members accessible within the class & to the sub classes

readonly: These are the variables that can’t be modified once its initialized in the constructor of the class

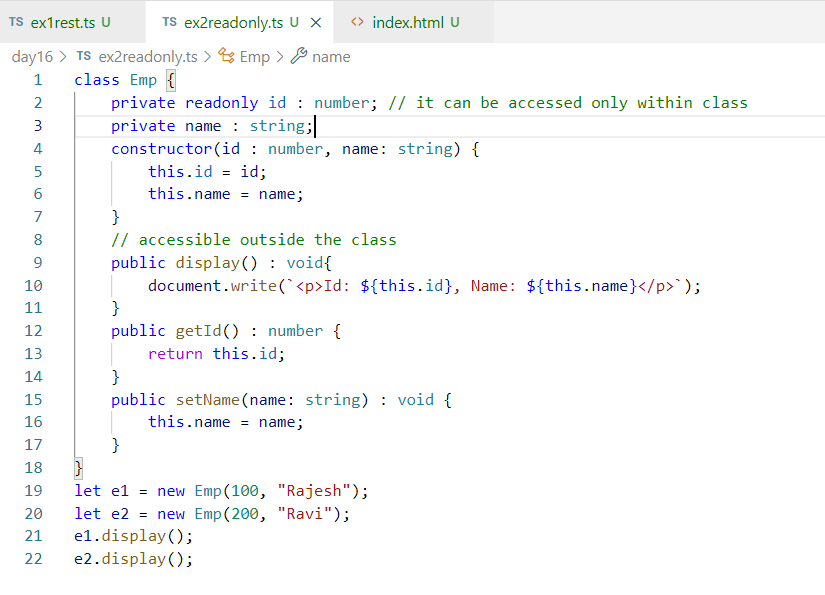
Note: const and read-only variables slightly differ, const variables must be initialized at the time declaration, whereas readonly variables must be initialized inside the constructor

class Emp {   
 private readonly name : string;  
 private readonly age: number;  
 private const x: number = 10; // it must be initialized at the time of declaration  
 constructor(name: string, age: number) {   
 this.name = name;  
 this.age = age;  
 }  
}

let e = new Emp(“Raj”, 35); // e.name & e.age can’t be modified

let e2 = new Emp(“Ravi”, 40); // e2.name & e2.age can’t be modified

ex2readonly.ts



Output:



Access Modifiers in Typescript

There are 3 access modifiers

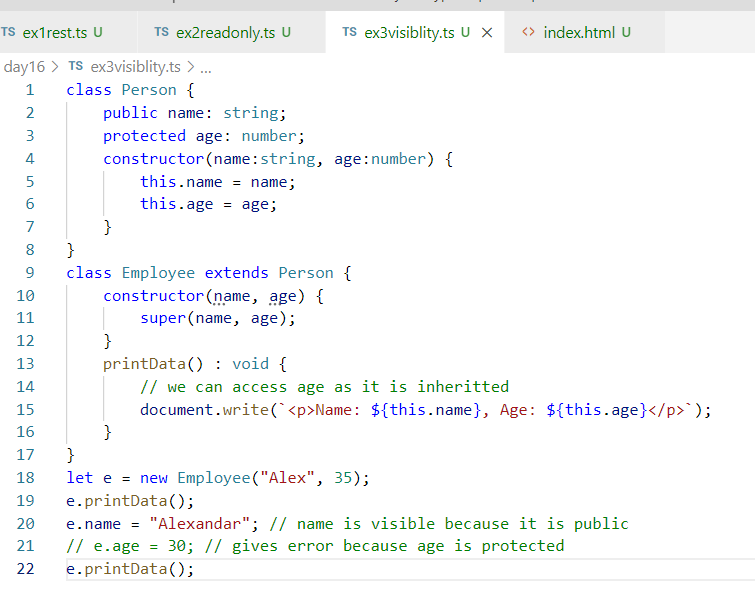
* private
* protected
* public

public: This is the default visible of the members, means all the members in a class are public by default, they are accessible from anywhere

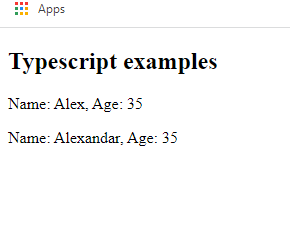
private: It is visible only within the class

protected: It is visible within the class & in the subclass

ex3visibility.ts



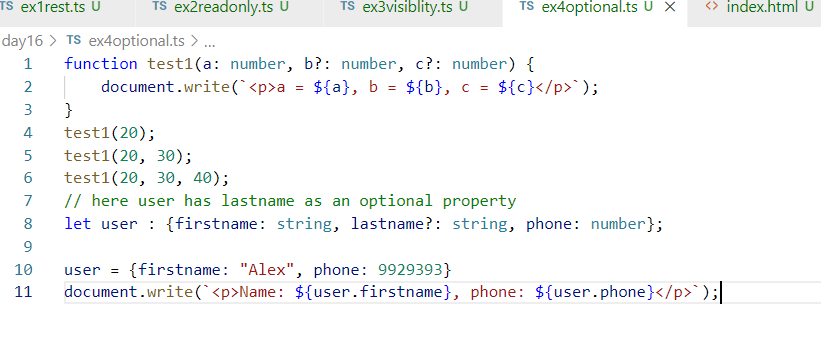
Output:



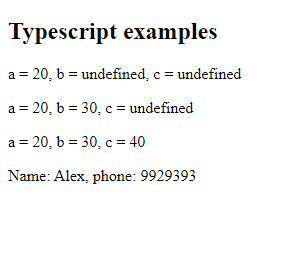
Optional properties & parameters(?):

Sometimes you may feel a function may or many not need parameters or an object may or may not need certain properties in such case you can make them optional

ex4optional.ts



Output:



Short-cut constructor initialization syntax:

This eliminates writing constructor with property initialization code, here the properties you want to initialize needn’t to be declared in the class and can used in the constructor parameter to act like the property but you need to use some access modifiers in the parameter of the constructor.

i.e., Emplyee without short-cut constructor initialization syntax;

class Employee {  
 private name: string;   
 private age : number;  
 constructor(name: string, age: number) {   
 this.name = name;   
 this.age = age;   
}  
}

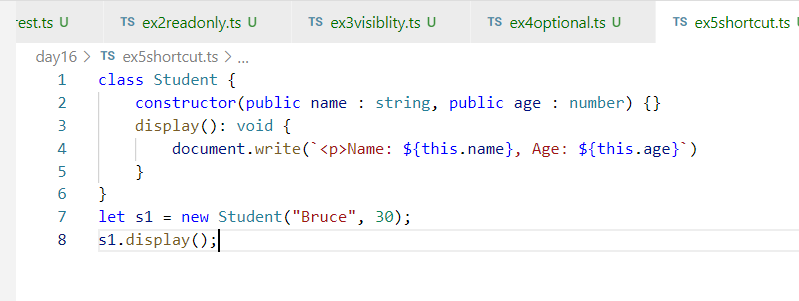
Same code with short-cut initialization of constructor

class Employee {   
 constructor(public name: string, public age: number) { }   
}

Typescript takes care of creating 2 properties in Employee name of string and age of number, it initializes the name & age with the parameters you pass while creating object.

name & age are available in the object which you can access with this keyword in the same class functions

ex5shortct.ts

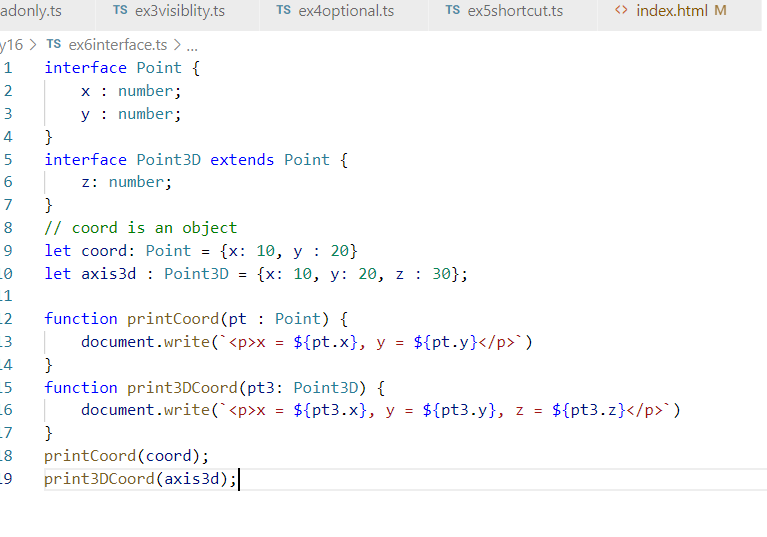


Output:

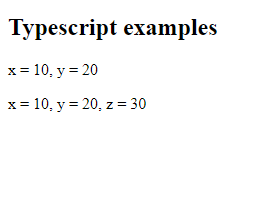


Interface:

It is a way to name an object types, it gives the structure of the object when you create, like how many properties an object must have and what are their data types



Output:



Intersection types:

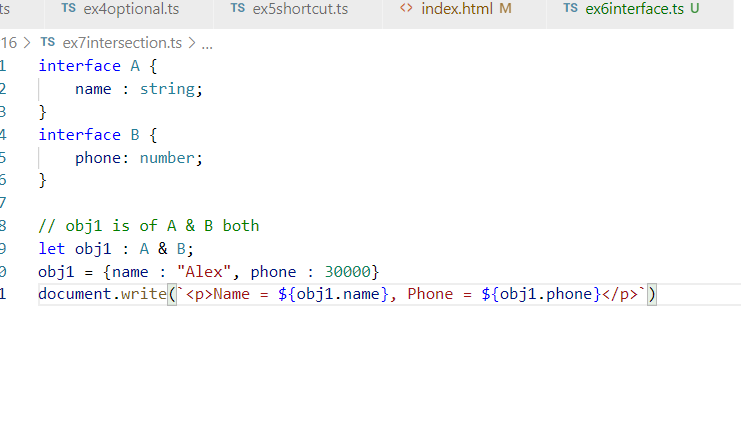
When you want a type to be created with multiple types and want that type to have all the features of the multiple types then you can use intersection types, like we used union with | to have a type that matches to any one type mentioned, in intersection you will use & to have a type that matches all the types mentioned.

interface A { }

interface B { }

let x : A & B; // x must have both A & B

let y : A | B; y can be either A or B type



Output:



Type aliases:

Usually you can create interface to create a type, but there’s another easier way to create a type i.e., type aliases, but its not extendable like interface

type Point = { x: number, y : number};

Here you are creating your own type, this allows you to combine some types to form a custom type

ex:

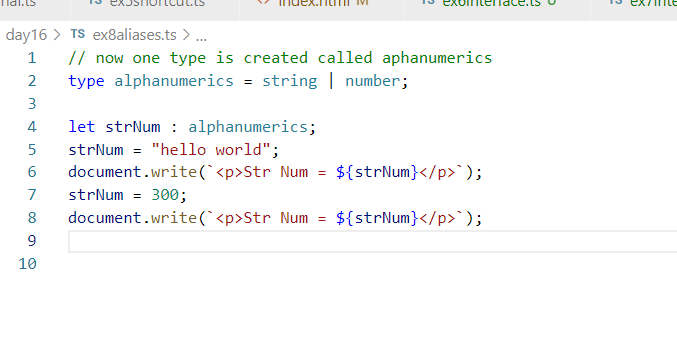
type alphanumerics = string | number;

let x : alphanumerics;

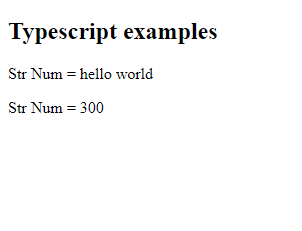
x = 10;

x = “hello”;

ex8aliases.ts



Output:



Enums:

It is to create defined set of named constants

enum Color { RED, GREEN, BLUE }

enum Direction { UP, DOWN, LEFT, RIGHT}

enum Gender { M, F}

let c : Color;

c = Color.RED; // ok

c = “RED”; // error

let x : Gender;

x value can be either Gender.M or Gender.F

If you assign x = “male”, it will be an error, because it must be Gender.F or Gender.M.

Modules

These are reusable variables, classes, functions written in a file that can be imported in another file

one.ts

export const items : string[] = [“apple”, “orange”, “grapes”];

export class User { }

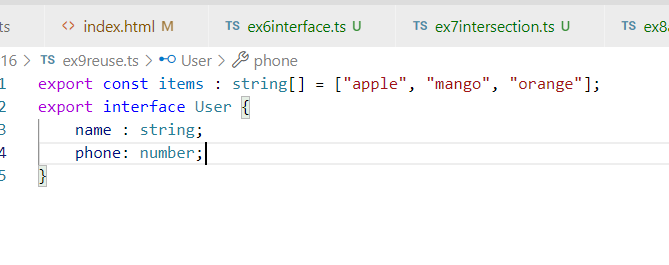
two.ts

import { items, User } from ‘./one’

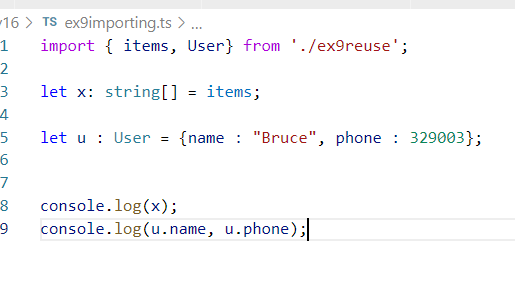
let x = items; value x will be apple, orange, grapes

let a : User; // now a can take only User object

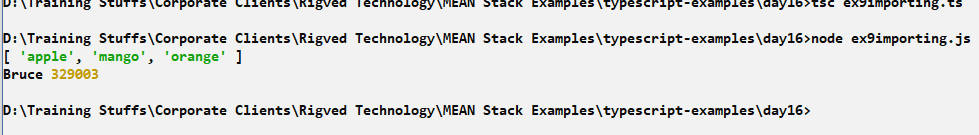
ex9reuse.ts



ex9importing.ts

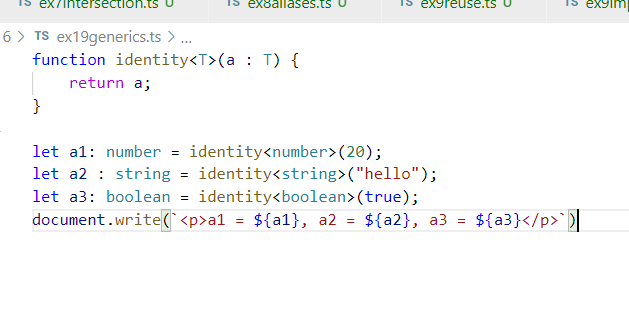


Note: In browser it may give exports error, but you can use node command to run the javascript



Generics:

It allows you to specify the datatypes at runtime, so that a variable can take any kind of datatypes at the time of creation.



Output:

