Static typing

Any variable, function argument or return value can have its type defined on initialization:

var burger: string = 'hamburger', // String

calories: number = 300, // Numeric

tasty: boolean = true; // Boolean

// Alternatively, you can omit the type declaration:

// var burger = 'hamburger';

// The function expects a string and an integer.

// It doesn't return anything so the type of the function itself is void.

function speak(food: string, energy: number): void {

console.log("Our " + food + " has " + energy + " calories.");

}

speak(burger, calories);

Javascript code

// JavaScript code from the above TS example.

var burger = 'hamburger',

calories = 300,

tasty = true;

function speak(food, energy) {

console.log("Our " + food + " has " + energy + " calories.");

}

speak(burger, calories);

If we try the belo , it throws an error

// The given type is boolean, the provided value is a string.

var tasty: boolean = "I haven't tried it yet";

Interface

Interfaces are used to type-check whether an object fits a certain structure.

// Here we define our Food interface, its properties, and their types.

interface Food {

name: string;

calories: number;

}

// We tell our function to expect an object that fulfills the Food interface.

// This way we know that the properties we need will always be available.

function speak(food: Food): void{

console.log("Our " + food.name + " has " + food.calories + " calories.");

}

// We define an object that has all of the properties the Food interface expects.

// Notice that types will be inferred automatically.

var ice\_cream = {

name: "ice cream",

calories: 200

}

speak(ice\_cream);

If something is missing, has the wrong type, or is named differently, the compiler will warn us.

interface Food {

name: string;

calories: number;

}

function speak(food: Food): void{

console.log("Our " + food.name + " has " + food.calories + " grams.");

}

// We've made a deliberate mistake and name is misspelled as nmae.

var ice\_cream = {

nmae: "ice cream",

calories: 200

}

speak(ice\_cream);

main.ts(16,7): error TS2345: Argument of type '{ nmae: string; calories: number; }

is not assignable to parameter of type 'Food'.

Property 'name' is missing in type '{ nmae: string; calories: number; }'.

Example 2:

interface IPerson {

firstName:string,

lastName:string,

sayHi: ()=>string

}

var customer:IPerson = {

firstName:"Tom",

lastName:"Hanks",

sayHi: ():string =>{return "Hi there"}

}

console.log("Customer Object ")

console.log(customer.firstName)

console.log(customer.lastName)

console.log(customer.sayHi())

var employee:IPerson = {

firstName:"Jim",

lastName:"Blakes",

sayHi: ():string =>{return "Hello!!!"}

}

console.log("Employee Object ")

console.log(employee.firstName) console.log(employee.lastName)

Classes in Typescript

class Menu {

// Our properties:

// By default they are public, but can also be private or protected.

items: Array<string>; // The items in the menu, an array of strings.

pages: number; // How many pages will the menu be, a number.

// A straightforward constructor.

constructor(item\_list: Array<string>, total\_pages: number) {

// The this keyword is mandatory.

this.items = item\_list;

this.pages = total\_pages;

}

// Methods

list(): void {

console.log("Our menu for today:");

for(var i=0; i<this.items.length; i++) {

console.log(this.items[i]);

}

}

}

// Create a new instance of the Menu class.

var sundayMenu = new Menu(["pancakes","waffles","orange juice"], 1);

// Call the list method.

sundayMenu.list();

Java classes

class HappyMeal extends Menu {

// Properties are inherited

// A new constructor has to be defined.

constructor(item\_list: Array<string>, total\_pages: number) {

// In this case we want the exact same constructor as the parent class (Menu),

// To automatically copy it we can call super() - a reference to the parent's constructor.

super(item\_list, total\_pages);

}

// Just like the properties, methods are inherited from the parent.

// However, we want to override the list() function so we redefine it.

list(): void{

console.log("Our special menu for children:");

for(var i=0; i<this.items.length; i++) {

console.log(this.items[i]);

}

}

}

// Create a new instance of the HappyMeal class.

var menu\_for\_children = new HappyMeal(["candy","drink","toy"], 1);

// This time the log message will begin with the special introduction.

menu\_for\_children.list();

Example 2:

class Car {

//field

engine:string;

//constructor

constructor(engine:string) {

this.engine = engine

}

//function

disp():void {

console.log("Function displays Engine is : "+this.engine)

}

}

//create an object

var obj = new Car("XXSY1")

//access the field

console.log("Reading attribute value Engine as : "+obj.engine)

//access the function

obj.disp()

Javascript code

//Generated by typescript 1.8.10

var Car = (function () {

//constructor

function Car(engine) {

this.engine = engine;

}

//function

Car.prototype.disp = function () {

console.log("Function displays Engine is : " + this.engine);

};

return Car;

}());

//create an object

var obj = new Car("XXSY1");

//access the field

console.log("Reading attribute value Engine as : " + obj.engine);

//access the function

obj.disp();

Example :3

class Student {

private firstName: string;

private lastName: string;

yearOfBirth: number; //Public scope by default

schoolName: string;

city: string;

//Constructor

constructor(firstName: string, lastName: string, schoolName: string, city: string, yearOfBirth: number) {

this.firstName = firstName;

this.lastName = lastName;

this.yearOfBirth = yearOfBirth;

this.city = city;

this.schoolName = schoolName;

}

//Function

age() {

return 2014 - this.yearOfBirth;

}

//Function

printStudentFullName(): void {

alert(this.lastName + ',' + this.firstName);

}

}

Inheritance

class Shape {

Area:number

constructor(a:number) {

this.Area = a

}

}

class Circle extends Shape {

disp():void {

console.log("Area of the circle: "+this.Area)

}

}

var obj = new Circle(223);

obj.disp()

On compiling, it will generate following JavaScript code.

//Generated by typescript 1.8.10

var \_\_extends = (this && this.\_\_extends) || function (d, b) {

for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];

function \_\_() { this.constructor = d; }

d.prototype = b === null ? Object.create(b) : (\_\_.prototype = b.prototype, new \_\_());

};

var Shape = (function () {

function Shape(a) {

this.Area = a;

}

return Shape;

}());

var Circle = (function (\_super) {

\_\_extends(Circle, \_super);

function Circle() {

\_super.apply(this, arguments);

}

Circle.prototype.disp = function () {

console.log("Area of the circle: " + this.Area);

};

return Circle;

}(Shape));

var obj = new Circle(223);

obj.disp();

Example 2:

//Interface

interface IStudent {

yearOfBirth: number;

age : () => number;

}

//Base Class

class College {

constructor(public name: string, public city: string) {

}

public Address(streetName: string) {

return ('College Name:' + this.name + ' City: ' + this.city + ' Street Name: ' + streetName);

}

}

//Child Class implements IStudent and inherits from College

class Student extends College implements IStudent {

firstName: string;

lastName: string;

yearOfBirth: number;

//private \_college: College;

//Constructor

constructor(firstName: string, lastName: string, name: string, city: string, yearOfBirth: number) {

super(name, city);

this.firstName = firstName;

this.lastName = lastName;

this.yearOfBirth = yearOfBirth;

}

age () {

return 2014 - this.yearOfBirth;

}

CollegeDetails() {

var y = super.Address('Maple Street');

alert(y);

}

printDetails(): void {

alert(this.firstName + ' ' + this.lastName + ' College is: ' + this.name);

}

}