UDACITY

Fullstack javascript nanodegree

Session 2

Agenda

- Intro to typescript
- Intro to unit testing
- Live collaboration
- Q & A

Intro to Typescript

Why TS

- Typescript is open source.
- Typescript simplifies JavaScript code, making it easier to read and debug.
- Typescript code can be **compiled as per ES5 and ES6 standards** to support the latest browser.
- Typescript can help us to avoid painful bugs that developers commonly run into when writing JavaScript by **type checking** the code.
- Typescript is nothing but **JavaScript with some additional features.**

Typescript features

- **Cross-platform:** typescript compiler can be installed on any operating system such as windows, mac os and Linux.
- **Object oriented language**
- **Static type-checking**: helps type checking at compile time. Thus, you can find errors while typing the code without running your script each tim
- **Optional static typing**: typescript also allows optional static typing if you would rather use JavaScript dynamic typing.
- **ES 6 features:** typescript includes most features of planned ECMAScript 2015 (ES 6, 7) such as class, interface, arrow functions etc.

Typescript data types

Variables can be declared using **Variable Declaration**
var(avoid using), let, const

- **String**
- **Number**
- **Boolean**
- **Array**
- **Tuple**
- **Enum**
- **Union**
- **Any (avoid using)**
- **Void**

Number, String and Boolean

```
//example for number,string and boolean

let normal:number = 1
let hexa:number = 0x111
let octa:number = 0o343
let binary:number = 0b10101

let str:string = "hello!" //Or use single quotes ''

let bool:boolean = true // or false
```

Array

```
//example arrays

let arr1:string[] =["hello","world"]

let arr2:Array<string> = ["hello" , "world"]
```

Tuple

```
//example tuple

let personName:string ="Ahmed"
let personAge:number = 25
let personHobbies:string[]=["swimming","reading"]

let personTuple : [string,number,string[]]=
["Ahmed",25,["swimming","reading"]]
```

Enums

```
enum colors{
    RED,
    YELLOW,
    BLUE
}
console.log(colors.BLUE)
//prints 2
```

You can also assign values to your enums (not common)

Union

```
//example union

let mixed :(string|number)

mixed = "hello" //ok
mixed = 32 //ok
mixed = true //error
```

Any

```
//example any

let mixed :any

mixed = "hello" //ok
mixed = 32 //ok
mixed = true //ok
//avoid using as you will lose type checking feature
```

Void

Used when you have a function with no return type

```
//example void

function voidFunction():void{
    console.log("void function, doesn't return")
}

function voidError():void{
    return 1
}
```

Type interference

Typescript will infer a type if you don't (not recommended)
Here, Typescript knows "a" is a number and "b" is a string

```
//example type interference

let a = 1
let b = "hello"
a=b
//Type 'string' is not assignable to type 'number'.
```

Type assertion (casting)

```
//example type assertion

let a:number = 123
//can be done using this syntax
let b:string = (a as unknown) as string
//or this
let c:string = <string>(<unknown> a)
//direct casting shows this error:
let d:string = <string> a
//Conversion of type 'number' to type 'string' may be
//a mistake because neither type sufficiently overlaps
//with the other. If this was intentional, convert
//the expression to 'unknown' first.
```

Functions

```
//functions
function example(num:number,optionalString?:string):number{
    console.log(num)
    console.log(optionalString)
    return num
}
example()//Expected 1-2 arguments, but got 0.ts(2554)
example(1)//ok
example(1,2)//Argument of type '2' is not assignable
// to parameter of type 'string | undefined'
example(1,"hello")//ok
example(1,"hello",2)//Expected 1-2 arguments, but got 3
```

interface

```
//interface
interface person{
    age:number,
    name:string,
    nickname?:string,
    readonly type:string
}
let p1:person = {age:20,name:"ahmed",notIncluded:1}//error
let p2:person = {age:20,name:"ahmed",nickname:"nickname",type:"human"}//ok
let p3:person = {age:20,name:"ahmed",type:"human"}//ok
p2.name = "Mohamed"//ok
p2.type="another type"//Cannot assign to 'type' because
//it is a read-only property.
```

type

```
//type
type person = {
    age:number,
    name:string,
    nickname?:string,
    readonly type:string
}
let p1:person = {age:20,name:"ahmed",notIncluded:1}//error
let p2:person = {age:20,name:"ahmed",nickname:"nickname",type:"human"}//ok
let p3:person = {age:20,name:"ahmed",type:"human"}//ok
p2.name = "Mohamed"//ok
p2.type="another type"//Cannot assign to 'type' because
//it is a read-only property.
```

Type or interface?

- You can use whatever you prefer but always notice the difference in syntax , there is a slight difference tho, you can check it here

```
//interface vs type extension

interface PartialPointX { x: number; }
type PartialPointY = { y: number }

//interfaces can extend a type
interface Point extends PartialPointX,PartialPointY{}

//types can extend an interface
type Point2 = PartialPointX & PartialPointY;

let p1:Point = {x:1,y:2}
let p2:Point2 = {x:1,y:2}
```

Intro to Unit-Testing

Unit-Testing

- Is a level of the software testing process where individual **units/components** of software/system are tested.

- It’s written and run by **software developers** to ensure that code meets it’s desgin and behaves as intended.

- The goal is to **isolate** each part of the program and show that individual parts are correct.

Unit-Testing Concerns

- Function correctness and completeness.
- Error Handling.
- Checking Input values.
- Correctness output data.
- Optimizing Performance.

Why Unit-Testing?

- Faster Debugging.
- Faster Development.
- Better Design.
- Reduce Future Costs.

Examples

```
it('calculate the sum of two numbers', () => {
  const result = calcSum(2, 3);

  expect(result).toEqual(5);
});
```

Examples

```
it('calculate the sum of two numbers, even with negative numbers', () => {
  const result = calcSum(2, -1);

  expect(result).toEqual(1);
});
```

Examples

```
describe('Testing the calcSum function', function() {
  it('calculate the sum of two numbers', () => {
    const result = calcSum(2, 3);

    expect(result).toEqual(5);
  });

  it('calculate the sum of two numbers, even with negative numbers', () => {
    const result = calcSum(2, -1);

    expect(result).toEqual(1);
  });

});
```

Matchers

- `toBe()`
- `toEqual()`
- `toBeDefined()`
- `toBeFalse()`
- `toBeNaN()`
- `toBeGreatherThan()`
- `toBeLessThan()`
- `toBeGreatherThanOrEqual()`

And a lot more you can find here.

Live Collaboration

Task

Description:

- You will work in individually.
- You should write the function that calculates the average for a group of numbers.

Notes:

- Every piece of your code as to be type annotated.

Setup:

- You must have a text editor installed on your machine.
- You can download the starter code from here, or you can just run the following commands:

```
$ git clone https://github.com/Elshafeay/ts-env.git
$ cd ts-env
$ npm install
$ git checkout ts-exercise
$ code .
// make your changes and then test your code using
$ npm test
```

Good Luck

Task

```
function calcAverage(){
  // write your code here
}

export { calcAverage };
```

Your Feedback is Appreciated

Remember that we are here to help you

All you have to do is **ASK!**

Thanks for attending

References

- Introduction To Typescript
- Static typing vs dynamic typing
- Is unit test worth it?
- Jasmine Matchers