Assignment 1

1.Write a brief description about unit testing and functional testing and its benefits in project, from a developer perspective?

Unit Testing :

- ➢ Unit testing is a type of testing where individual units or components of a software are tested
- ➢ The purpose is to validate that each unit of the software code performs as expected
- ➢ Unit Testing is done during the development (coding phase) of an application
- ➢ With this method of testing, both testers and developers can isolate each module
- ➢ It isolates a section of code and verifies its correctness.
- ➢ A unit may be an individual function, method, procedure, module, or object.

Benefits of unit testing :

- ➢ Unit tests help to fix bugs early in the development
- ➢ It helps the developers to understand the testing code base and enables them to make changes quickly
- ➢ Due to the modular nature of the unit testing, we can test parts of the project without waiting for others to be completed.
- ➢ Assures in simplifying the debugging process
- ➢ Improve the design of implementations.
- ➢ Add new features without breaking anything.

Functional Testing :

- ➢ It is a type of software testing which is used to verify the functionality of the software application
- ➢ The purpose of Functional tests is to test each function, by providing appropriate input, verifying the output against the Functional requirements.
- ➢ This testing is done to identify whether all the functions are working as expectations.
- ➢ It focuses on application specification rather than actual code.

Benefits of functional testing :

- ➢ It is performed from the perspective of the users, which allows the development team to create test scenarios that represent the real world use scenarios.
- ➢ Allows the team to meet the requirements of the user as well as the client.
- ➢ It helps improve actual system usage.
- ➢ Enhances the quality of the software product

2.Where and why do you need unit testing in your project, give me 10 examples and code snap?

Needed of unit testing :

-to validate addition

-to validate subtraction

-to validate title

app.component.html

```
<h1>{{ title }}</h1>
<h1>Adding inputBox Numbers</h1>
    <p>Num1: <input [(ngModel)]="num1"></p>
    <p>Num2: <input [(ngModel)]="num2"></p>
    <button (click)="add()">Add</button>
     <!---<p>Addition : {{ num1*1 - num2*1 }}</p>-->
     <button (click)="sub()">Sub</button>
<h3>{{ result }}</h3>
 <!---<p>Subtraction {{ num1*1  - num2*1 }}</p>-->
```

app.component.ts

```
import { Component } from '@angular/core';
import { FormBuilder } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
```

```
  title = 'UnitTesting1';
  num1!: number;
  num2!: number;
  result!: number;
  constructor(
  ) {}
  add() {
    this.result = this.num1 + this.num2;
  }
  sub() {
    this.result = this.num1 - this.num2;
  }
}
```

Test cases :

1.
```
//pass
it('should create the app', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  expect(app).toBeTruthy();
});
```

2.
```
//fail
it('should display original title', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  expect(h1.textContent).toContain(app.title);
});
```

3.
```
//pass
it(`should have as title 'UnitTesting1'`, () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  expect(app.title).toEqual('UnitTesting1');
});
```

4.
```
//fail
it('should render title', () => {
  const fixture = TestBed.createComponent(AppComponent);
  fixture.detectChanges();
  const compiled = fixture.nativeElement;
  expect(compiled.querySelector('.content span').textContent).toContain('UnitTesting1 app is running!');
});
```

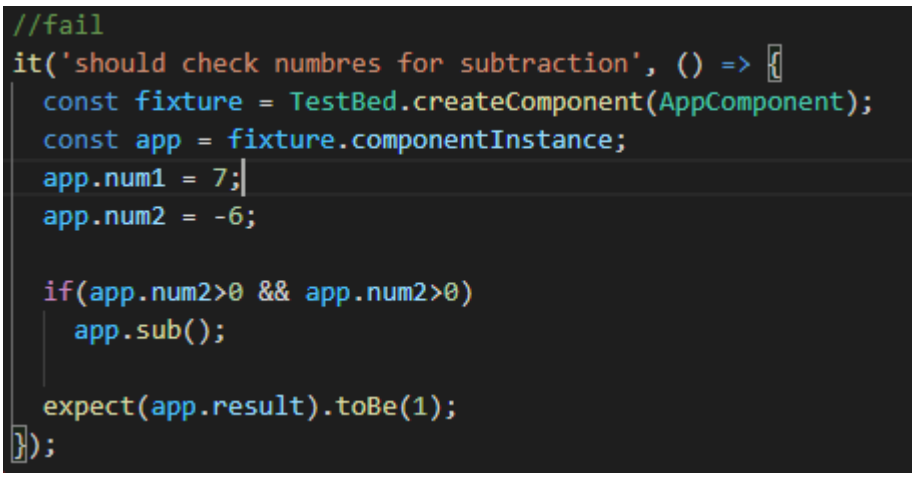
5.
```
//pass
it('should do addition', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  app.num1 = 5;
  app.num2 = 7;

  app.add();

  expect(app.result).toBe(12);
});
```

6.
```
//fail
it('should check number1', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  app.num1 = -5;
  app.num2 = 7;

  if(app.num1>0){
    app.add();
  }

  expect(app.result).toBe(12);
});
```

7.
```
//fail
it('should check number2', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  app.num1 = 5;
  app.num2 = -7;

  if(app.num2>0){
    app.add();
  }

  expect(app.result).toBe(12);
});
```

8.
```
//pass
it('should check both number', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  app.num1 = 5;
  app.num2 = 7;

  if(app.num1>0 && app.num2>0){
    app.add();
  }

  expect(app.result).toBe(12);
});
```

9.
```
//pass
it('should do subtraction', () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  app.num1 = 7;
  app.num2 = 6;

  app.sub();

  expect(app.result).toBe(1);
});
```

10.

//fail
it('should check numbres for subtraction', () => {
const fixture = TestBed.createComponent(AppComponent);
const app = fixture.componentInstance;
app.num1 = 7;
app.num2 = -6;
if(app.num2>0 && app.num2>0)
app.sub();
expect(app.result).toBe(1);
});