1. **Write a brief description about unit testing and functional testing and its benefits from a developer perspective.**

Unit Testing:

Unit Testing is a software testing technique by means of which individual units of software i.e., group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent module is tested to determine if there are any issue by the developer himself. It is correlated with functional correctness of the independent modules. Unit Testing of software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer.

The main objective of unit testing:

- To isolate a section of code.
- To verify the correctness of code.
- To test every function and procedure.
- To fix bug early in development cycle and to save costs.
- To help for code reuse.

Advantages of Unit Testing:

- Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
- Unit testing allows the programmer to refine code and make sure the module works properly.
- Unit testing enables to test parts of the project without waiting for others to be completed.

Functional Testing

Functional Testing is a type of Software Testing in which the system is tested against the functional requirements and specifications. Functional testing ensures that the requirements or specifications are properly satisfied by the application. This type of testing is particularly concerned with the result of processing. It focuses on simulation of actual system usage but does not develop any system structure assumptions. It is basically defined as a type of testing which verifies that each function of the software application works in conformance with the requirement and specification. This testing is not concerned about the source code of the application. Each functionality of the software application is tested by providing appropriate test input, expecting the output and comparing the actual output with the expected output. This testing focuses on checking of user interface, APIs, database, security, client or server application and functionality of the Application Under Test.

Functional Testing Process:

- Identify function that is to be performed.
- Create input data based on the specifications of function.

- Determine the output based on the specifications of function.
- Execute the test case.
- Compare the actual and expected output.

Advantages of Functional Testing

- It ensures to deliver a bug-free product.
- It ensures to deliver a high-quality product.
- No assumptions about the structure of the system.
- This testing is focused on the specifications as the customer usage.

2. Where and why, you need Unit Testing in your project give me 10 examples.

Where and Why:

- To isolate a section of code.
- To verify the correctness of code.
- To test every function and procedure.
- To fix bug early in development cycle and to save costs.
- To help for code reuse.

Unit tests are also especially useful when it comes to refactoring or re-writing a piece of code. If you have good unit tests coverage, you can refactor with confidence. Without unit tests, it is often hard to ensure that you didn't break anything.

Examples:

1.
```
// Arrange
let count = 0;// Act
const newCount = increment(count);// Assert
expect(newCount).toBe(1);
```

2.
```
describe('FileUploadComponent', () => {

 let component: FileUploadComponent;

 let fixture: ComponentFixture<FileUploadComponent>;

 beforeEach(() => {
  TestBed.configureTestingModule({
   declarations: [
    FileUploadComponent
   ]
  });

  fixture = TestBed.createComponent(FileUploadComponent);
  component = fixture.componentInstance;
```

```
    fixture.detectChanges();
  });

 it('should create file uploader component', () => {
    expect(component).toBeTruthy();
  });
});
```

3.
```
describe("ParagraphComponent", () => {
 let component: ParagraphComponent;
  let fixture: ComponentFixture<ParagraphComponent>;
  let dbgElement: DebugElement;
  let element: HTMLElement;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [ParagraphComponent]
    });
    fixture = TestBed.createComponent(ParagraphComponent);
    component = fixture.componentInstance;

    dbgElement = fixture.debugElement.query(By.css(".phone"));
    element = dbgElement.nativeElement;

    fixture.detectChanges();
  });

  it("paragraph should contain default message", () => {
    expect(element.innerText).toContain("not specified");
  });

  it("paragraph should contain phone number", () => {
```

```
    component.phone = "0021000111";

    fixture.detectChanges();

    expect(element.innerText).toContain("0021000111");
  });
});
```

4.
```
describe('MinutesToHoursPipe', () => {
  const pipe = new MinutesToHoursPipe();

  it('return same value if undefined', () => {
    expect(pipe.transform(undefined)).toBe(undefined);
  });

  it('transform minutes to hours and minutes string', () => {
    const testData = {
      1: '1 minute',
      30: '30 minutes',
      60: '1 hour',
      90: '1 hour and 30 minutes',
      97: '1 hour and 37 minutes',
      120: '2 hours',
      121: '2 hours and 1 minute',
      131: '2 hours and 11 minutes'
    };

    Object.keys(testData).forEach((key) => {
      expect(pipe.transform(key)).toBe(testData[key]);
    });
```

```
  });
});
```

5.
```
describe('Company.Model', () => {
        it('should create an instance', () => {
        expect(new Company.Model()).toBeTruthy();
        });
});
```

6.
```
it(`should have as title 'AngularApp'`, () => {
        const fixture = TestBed.createComponent(AppComponent);
        const app = fixture.componentInstance;
        expect(app.title).toEqual('AngularApp');
});
```

7.
```
it ('should check incremented value is greater than zero', ()=> {
        let counterComponent: CounterComponent = new CounterComponent();
        const curCounterValue = counterComponent.increaseCounter();
        expect(curCounterValue).toBeGreaterThan(0);
});
```

8.
```
it ('should check decremented value is less than zero', ()=> {
        let counterComponent: CounterComponent = new CounterComponent();
        const curCounterValue = counterComponent.decreaseCounter();
        expect(curCounterValue).toBeLessThan(0);
}) ;
```

9.
```
t ('should increment if input is positive', ()=> {
        const counterResult = counterParameter(1);
        expect(counterResult).toBe(2);
}) ;
```

10.
```
describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  });

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
```

```
    expect(app).toBeTruthy();
  });

  it(`should have as title 'my-first-app'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('my-first-app');
  });

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