Lecture slides - Week 14

Python - Unit Testing

Dr. Aamir Akbar

Director of both AWKUM AI Lab and AWKUM Robotics, Final Year Projects (FYPs) coordinator, and lecturer at the department of Computer Science
Abdul Wali Khan University, Mardan (AWKUM)

Contents

Software Testing

Unit Testing

The `unit` can be a function, a class, a method, or a module.

The purpose of `unit testing` is to test individual units or components of code to ensure they function correctly in isolation. The goal is to validate that each unit behaves as expected, according to its specifications.

The `unittest` module in Python provides a framework for creating and running unit tests.

Tests are typically written in separate Python files and use `assertions` to check whether the actual output of a unit matches the expected output.

`Unit testing` helps catch bugs early in the development process, ensures code reliability, facilitates code maintenance, and serves as documentation for how the code should behave.

How to test a single function as a `unit`

```python
import unittest

# Function to be tested
def add(a, b):
    return a + b

# Test case class
class TestAddFunction(unittest.TestCase):

    def test_add_positive_numbers(self):
        self.assertEqual(add(3, 5), 8)  # Assertion to check if 3 + 5 = 8

    def test_add_negative_numbers(self):
        self.assertEqual(add(-3, -7), -10)  # Assertion to check if -3 + -7 = -10

    def test_add_zero_to_number(self):
        self.assertEqual(add(10, 0), 10)  # Assertion to check if 10 + 0 = 10

unittest.main()
```

`unittest.TestCase` is the base class for all test cases. Each test is written as a method within a test case class, starting with the word `test`. Assertions like `self.assertEqual()` are used to verify if the actual output matches the expected output.

Case Study: Virtual Pet (with unit testing)

Virtual Pet

Activity: Create a class representing a virtual pet, capable of storing the pet's name and species information. The class must track the pet's hunger and energy levels. Feeding the pet will decrement its hunger level, while playing with the pet will reduce its energy level.

Virtual Pet

Activity: Create a class representing a virtual pet, capable of storing the pet's name and species information. The class must track the pet's hunger and energy levels. Feeding the pet will decrement its hunger level, while playing with the pet will reduce its energy level.

`Attributes` and `Methods:`

Create a class representing a virtual pet, capable of storing the pet's `name` and `species` information. The class must track the pet's `hunger` and `energy` levels. `Feeding` the pet will decrement its hunger level, while `playing` with the pet will reduce its energy level.

VirtualPet class

```
class VirtualPet:
    def __init__(self, name, species):
        self.name = name
        self.species = species
        self.hunger = 50
        self.energy = 50

    def feed(self):
        self.hunger -= 10
        if self.hunger < 0:
            self.hunger = 0

    def play(self):
        self.energy -= 20
        if self.energy < 0:
            self.energy = 0
```

TestVirtualPet class

This unit testing class will test each method of the VirtualPet class

```
import unittest
from virtualpet import VirtualPet

class TestVirtualPet(unittest.TestCase):

    def setUp(self):
        self.pet = VirtualPet("Fido", "Dog")

    def test_initial_values(self):
        self.assertEqual(self.pet.name, "Fido")
        self.assertEqual(self.pet.species, "Dog")
        self.assertEqual(self.pet.hunger, 50)
        self.assertEqual(self.pet.energy, 50)

    def test_feed(self):
        self.pet.feed()
        self.assertEqual(self.pet.hunger, 40)

    def test_play(self):
        self.pet.play()
        self.assertEqual(self.pet.energy, 30)
        self.pet.energy = 5  # Simulate low energy
        self.pet.play()
        self.assertEqual(self.pet.energy, 0)  # Check if energy doesn't go below 0
```

Test Driven Development

What is Test Driven Development (TDD)

`Test-Driven Development (TDD)` is a software development approach where tests are written before the actual code.

What is Test Driven Development (TDD)

`Test-Driven Development (TDD)` is a software development approach where tests are written before the actual code.

Write Test Cases: Initially, you create tests that define the expected behavior of the code you're about to write. These tests often fail since the code to satisfy them doesn't exist yet.

What is Test Driven Development (TDD)

`Test-Driven Development (TDD)` is a software development approach where tests are written before the actual code.

Write Test Cases: Initially, you create tests that define the expected behavior of the code you're about to write. These tests often fail since the code to satisfy them doesn't exist yet.

Write Code: Now, you create the minimum code required to pass those tests. The goal is to make the failing tests pass, not to create complete functionality.

What is Test Driven Development (TDD)

`Test-Driven Development (TDD)` is a software development approach where tests are written before the actual code.

Write Test Cases: Initially, you create tests that define the expected behavior of the code you're about to write. These tests often fail since the code to satisfy them doesn't exist yet.

Write Code: Now, you create the minimum code required to pass those tests. The goal is to make the failing tests pass, not to create complete functionality.

Refactor: Once the tests pass, you refactor the code to improve its structure, without changing its behavior. After refactoring, you rerun the tests to ensure everything still works as expected.