Gouthami Kanchukatla **PART-A** **ASU ID: 1218506822**

What is a unit testing framework?

Unit testing is a type of testing in which the individual components of the software code are tested by the developers during the coding phase. Unit testing framework is just like a software infrastructure which facilitates writing and executing of unit tests. It facilitates the automation of unit tests, setting up, shutting down code for tests and also supports test aggregation and test independence from the reporting framework. It provides methodology and set of guidelines for testing by which the individual units such as classes and methods are tested. When using a test automation framework it provides advantages like increased portability, reusability, and low maintenance script cost.

How can a developer utilize a framework?

Can utilize test runners to identify the tests in code both through (GUI) as well as console.

Can utilize to run tests automatically followed by the fail/pass status for every test.

Can utilize the framework to save money,time by fixing the bugs in the early development phase

Can work independently without pausing for the other code parts to be completed.

Can make their code more reliable and reusable as modules are isolated from each other in the framework.

What are the benefits of a framework?

Facilitates the agility in programming as well as safe refactoring during the development phase.

Improves code quality. In test driven development, it identifies edge cases,enabling us to write better code.

Provides a easy platform for code changes as well as helps in the continuous integration

Helps to reduce cyclomatic complexity as well as provides concrete proof of code implementation.

Provides indicators for documentation, helps to learn functionality thereby increasing development speed.

Examples of Unit Testing Frameworks: Pytest, Junit, XUnit, NUnit. PyUnit, Jasmine

Common capabilities of Pytest and Jasmine:

Both support execution of the testing code on the client side such as web browsers, other frontend components.

Both support testing the code behaviour on the server side such cas database and backend components.

Both facilitate fixtures, providing a specific environment for single testing.

Both allow organization of tests in groups.

Both are licensed under MIT.

Differences between Jasmine and Pytest	
Jasmine	Pytest
Uses javascript as a programming language.	Uses python as a programming language.
Supports End to End testing.	Does not support End to End testing.
Behaviour Driven Development framework.	Test Driven Development (all in one) framework.
Uses jasmine-ajax plugin for mock functionality.	Uses pytest-mock wrapper for mock functionality.
Does not support data generators and group fixtures.	supports data generators and group fixtures.

Code:

```
def heapify(arr, n, i):
    l = 2 * i + 1
    r = 2 * i + 2
    largest = i

    if l < n and arr[largest] < arr[l]:
        largest = l
    if r < n and arr[largest] < arr[r]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]  # swap
        heapify(arr, n, largest)

def heapSort(arr):
    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]  # swap
        heapify(arr, i, 0)

arr = [4,2,1,3]
heapSort(arr)
n = len(arr)
print("Sorted array is")
for i in range(n):
    print("%d" % arr[i]),
```

Test Cases:

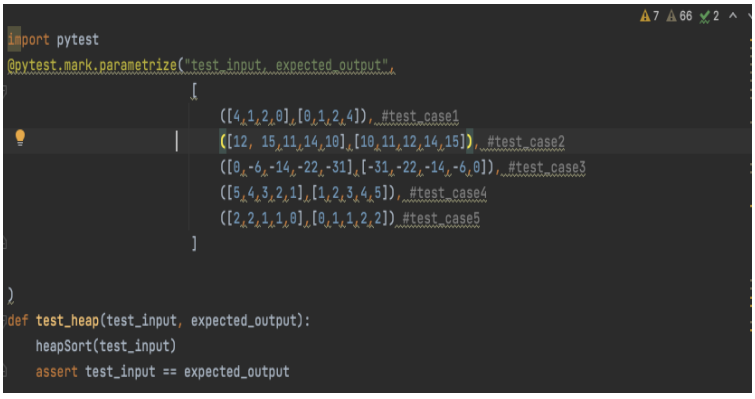
test_input	expected_output
[4,1,2,0]	[0,1,2,4]
[0,-6,-14,-22,-31]	[-31,-22,-14,-6,0]
[12, 15,11,14,10]	[10,11,12,14,15]
[5,4,3,2,1]	[1,2,3,4,5]
[2,2,1,1,0]	[0,1,1,2,2])

I have used pytest framework for unit testing

Here is the unittest code:

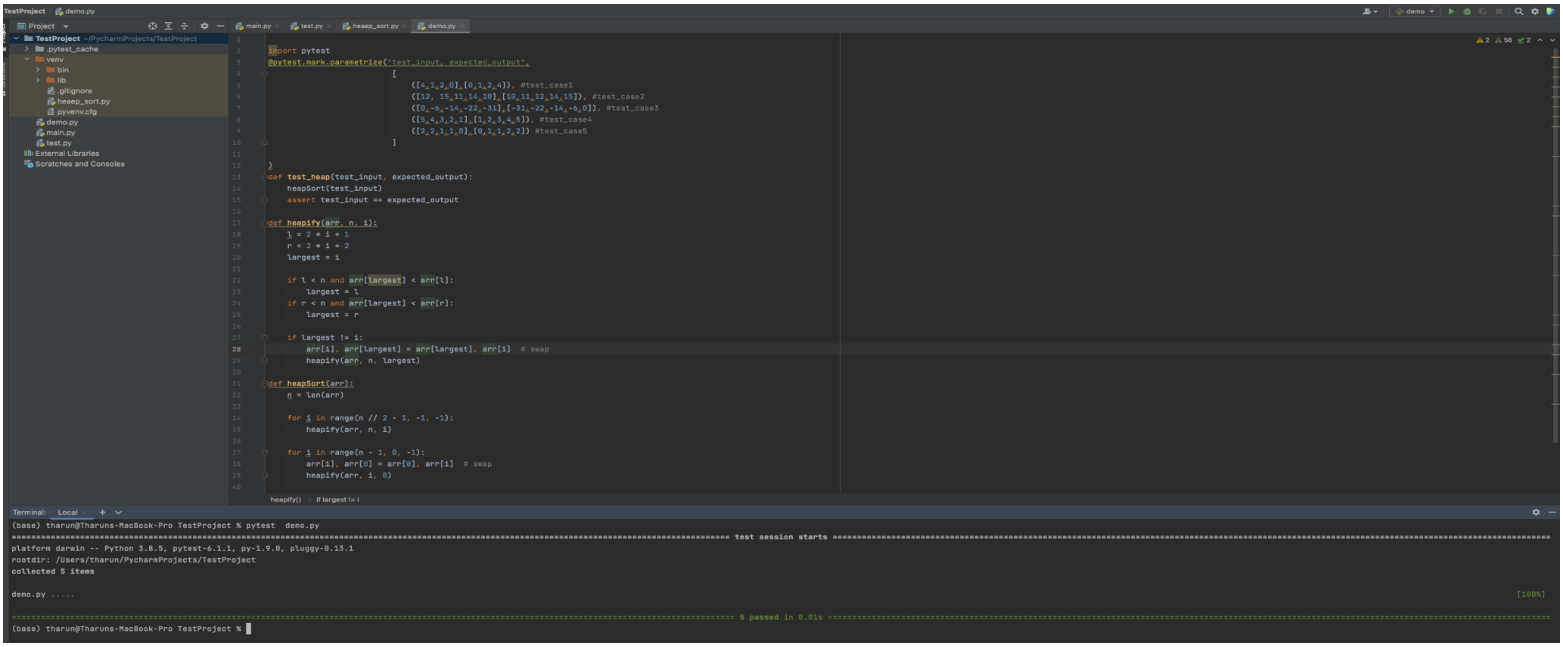
```
import pytest
@pytest.mark.parametrize("test_input, expected_output",
                         [
                            ([4,1,2,0],[0,1,2,4]), #test_case1
                            ([12, 15,11,14,10],[10,11,12,14,15]), #test_case2
                            ([0,-6,-14,-22,-31],[-31,-22,-14,-6,0]), #test_case3
                            ([5,4,3,2,1],[1,2,3,4,5]), #test_case4
                            ([2,2,1,1,0],[0,1,1,2,2]) #test_case5
                         ]

)
def test_heap(test_input, expected_output):
    heapSort(test_input)
    assert test_input == expected_output

def heapify(arr, n, i):
    l = 2 * i + 1
    r = 2 * i + 2
    largest = i

    if l < n and arr[largest] < arr[l]:
        largest = l
    if r < n and arr[largest] < arr[r]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]  # swap
        heapify(arr, n, largest)

def heapSort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]  # swap
        heapify(arr, i, 0)
```

Part1- test cases



Part2-output

Entire screen with test cases and output:



References:

https://www.guru99.com/unit-testing-guide.html

https://docs.python.org/3/library/unittest.html

https://www.upwork.com/resources/unit-testing

https://fortegrp.com/the-importance-of-unit-testing/

https://dzone.com/articles/top-8-benefits-of-unit-testing

https://www.softwaretestinghelp.com/python-testing-frameworks/

https://knapsackpro.com/testing_frameworks/difference_between/pytest/vs/jasmine

https://www.geeksforgeeks.org/heap-sort/#