**CS301-Software Engineering –Class Practice Sessions - 1**

**Time : Weekend Date : 24th March,2023**

**Name: Nariyam Likhith Roll.no:21BCS072**

**Theme : Create new cultural destination to celebrate the heritage of India and provide a platform for emerging Talents using Digital Technology solutions**

**Aim :**

* Creating doors for a first-of-its-kind, multi-disciplinary space for the Arts in cities
* Encourage Visual art space and captivating array of public art
* Bring together communities through a dynamic programming of epic theatricals , regional theatre, music , dance , spoken word etc.
* Major attraction is to provide a platform for emerging talent and showcases the vibrance of India’s heritage
* Generate source of income for the Art communities through collaborations, aggregators and accelerators investments

**Target audiences :**

* Home to Art, Artists, the audience from India and around the world.

**Assignment scope :**

1. Identify various requirements for the above program initiative that can be developed as a digital solutions
2. Use ChatGPT platform an generate code for the above requirements
   1. Generate code and run the program in Goggle Colab/Jupiter Notebook/Visual Code/PyCharm
   2. Perform integrated testing. Add integration testing code in the same program.
3. Modify the same program. Write APIs to access the data from the public domain and test the program for regression testing the same program

**Deliverables :**

Working Program with test scripts embedded in the same program.

CODE:

def art\_ecommerce\_platform(items):

"""

An e-commerce platform for artists to sell their work, including merchandise, artworks, and tickets for events.

Args:

- items: A list of dictionaries representing the items available for sale. Each dictionary should have the following keys:

- 'name': A string representing the name of the item.

- 'type': A string representing the type of the item ('artwork', 'merchandise', or 'ticket').

- 'description': A string representing the description of the item.

- 'price': A float representing the price of the item.

- 'quantity': An integer representing the quantity of the item available for sale.

Returns:

- A string representing the result of the transaction, including the total cost and any relevant information about the items purchased.

"""

total\_cost = 0

items\_purchased = []

# Print out the list of available items for sale

print("Available items:")

for item in items:

print(f"- {item['name']} ({item['type']}) - {item['price']} INR")

# Get the user's choice of item to purchase

while True:

choice = input("Enter the name of the item you would like to purchase, or 'done' to finish shopping: ")

if choice == 'done':

break

# Find the selected item in the list

selected\_item = None

for item in items:

if item['name'] == choice:

selected\_item = item

break

if selected\_item is None:

print("Sorry, that item is not available.")

continue

# Get the quantity of the item to purchase

while True:

quantity = int(input(f"How many {selected\_item['name']} would you like to purchase? "))

if quantity > selected\_item['quantity']:

print(f"Sorry, there are only {selected\_item['quantity']} {selected\_item['name']} available.")

else:

break

# Calculate the cost of the items purchased

cost = selected\_item['price'] \* quantity

total\_cost += cost

# Update the quantity of the item available for sale

selected\_item['quantity'] -= quantity

# Add the purchased items to the list

items\_purchased.append({'name': selected\_item['name'], 'quantity': quantity, 'cost': cost})

print(f"{quantity} {selected\_item['name']} added to your cart.")

# Print out the items purchased and the total cost

print("Items purchased:")

for item in items\_purchased:

print(f"- {item['quantity']} {item['name']} - {item['cost']} INR")

print(f"Total cost: {total\_cost} INR")

return f"Thank you for your purchase! Total cost: {total\_cost} INR. We hope you enjoy your items!"

UNIT TESTING:

import unittest

from my\_module import art\_ecommerce\_platform

class TestArtEcommercePlatform(unittest.TestCase):

def setUp(self):

self.items = [

{'name': 'Mona Lisa', 'type': 'artwork', 'description': 'Famous painting by Leonardo da Vinci', 'price': 1000, 'quantity': 5},

{'name': 'Van Gogh Starry Night Mug', 'type': 'merchandise', 'description': 'Ceramic mug with Van Gogh Starry Night print', 'price': 20, 'quantity': 10},

{'name': 'Concert Ticket', 'type': 'ticket', 'description': 'Ticket to see Adele in concert', 'price': 500, 'quantity': 2}

]

def test\_purchase\_items(self):

# Test purchasing one item

result = art\_ecommerce\_platform(self.items)

expected\_result = "Thank you for your purchase! Total cost: 1000 INR. We hope you enjoy your items!"

self.assertEqual(result, expected\_result)

# Test purchasing multiple items

self.items[0]['quantity'] = 3 # reduce quantity of Mona Lisa to test purchasing multiple items

result = art\_ecommerce\_platform(self.items)

expected\_result = "Thank you for your purchase! Total cost: 1060 INR. We hope you enjoy your items!"

self.assertEqual(result, expected\_result)

def test\_out\_of\_stock\_items(self):

# Test purchasing an out of stock item

self.items[0]['quantity'] = 0

result = art\_ecommerce\_platform(self.items)

expected\_result = "Sorry, there are only 0 Mona Lisa available."

self.assertIn(expected\_result, result)

# Test purchasing an item with a quantity greater than available

self.items[1]['quantity'] = 2

result = art\_ecommerce\_platform(self.items)

expected\_result = "Sorry, there are only 2 Van Gogh Starry Night Mug available."

self.assertIn(expected\_result, result)

def test\_invalid\_item(self):

# Test purchasing an invalid item

result = art\_ecommerce\_platform(self.items)

self.assertIn("Sorry, that item is not available.", result)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

------------------------------------------------------------------------------

This code defines a TestArtEcommercePlatform class with three test methods:

test\_purchase\_items: Tests purchasing one item and multiple items.

test\_out\_of\_stock\_items: Tests purchasing an out of stock item and an item with a quantity greater than available.

test\_invalid\_item: Tests purchasing an invalid item.

In each test method, we call the art\_ecommerce\_platform function with different inputs and assert that the result is what we expect. We can run this code by executing the file or by running python -m unittest <filename> in the command line.

\*\*\*\*\*\*\*THE END\*\*\*\*\*\*\*

-NARIYAM LIKHITH

-21BCS072.