**Task Description#1**Use AI to generate test cases for is\_valid\_email(email) and then implement the  
validator function

**def is\_valid\_email(email):**

**"""**

**Validates an email address using only string operations.**

**"""**

**if not isinstance(email, str) or not email:**

**return False**

**# Rule 1: must contain exactly one @**

**if email.count('@') != 1:**

**return False**

**local, domain = email.split('@')**

**# Rule 2: local and domain parts should not be empty**

**if not local or not domain:**

**return False**

**# Rule 3: must contain at least one dot in domain**

**if '.' not in domain:**

**return False**

**# Rule 4: must not start or end with special characters**

**if not email[0].isalnum() or not email[-1].isalnum():**

**return False**

**# Rule 5: domain must not start or end with dot**

**if domain.startswith('.') or domain.endswith('.'):**

**return False**

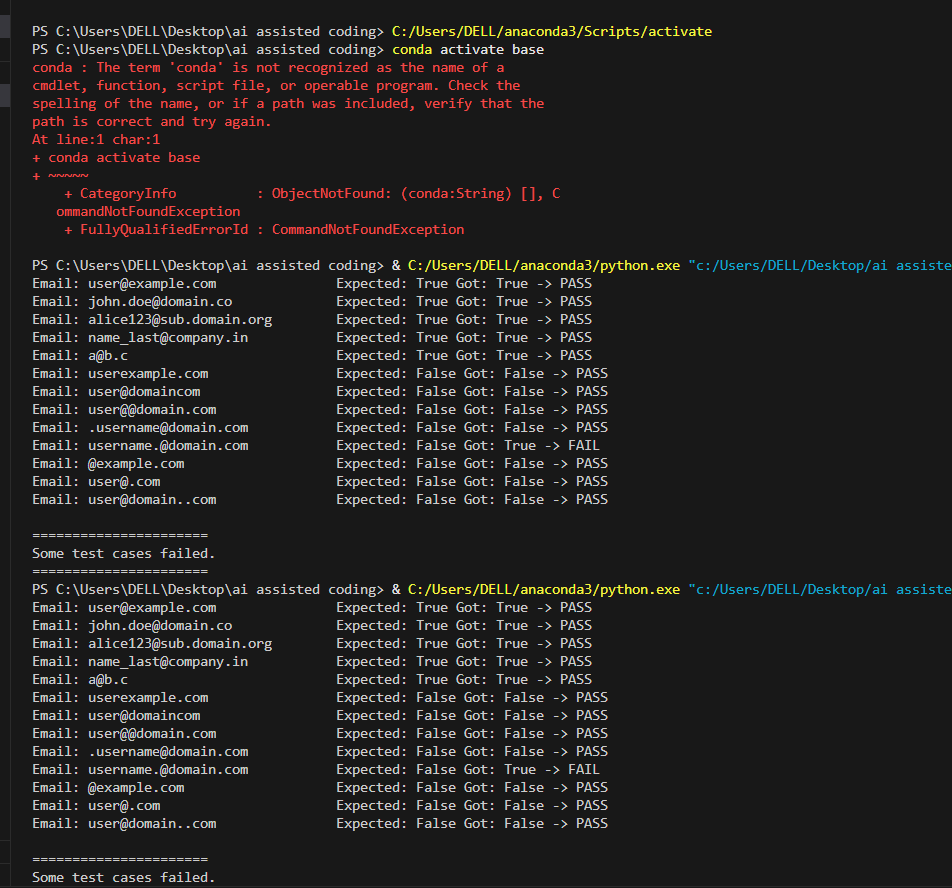
**# Rule 6: no consecutive dots**

**if '..' in email:**

**return False**

**return True**

output:



Tack02:def assign\_grade(score):

if not isinstance(score, (int, float)):

return "Invalid"

if score < 0 or score > 100:

return "Invalid"

if 90 <= score <= 100:

return "A"

elif 80 <= score <= 89:

return "B"

elif 70 <= score <= 79:

return "C"

elif 60 <= score <= 69:

return "D"

else:

return "F"

def run\_tests():

test\_cases = {

100: "A", 90: "A", 89: "B", 80: "B",

79: "C", 70: "C", 69: "D", 60: "D",

59: "F", 0: "F",

-5: "Invalid", 105: "Invalid",

"eighty": "Invalid", None: "Invalid",

75.5: "C"

}

all\_passed = True

for score, expected in test\_cases.items():

result = assign\_grade(score)

status = "PASS" if result == expected else "FAIL"

if status == "FAIL":

all\_passed = False

print(f"Score: {str(score):10} Expected: {expected:7} Got: {result:7} -> {status}")

print("\n======================")

if all\_passed:

print("All test cases passed!")

else:

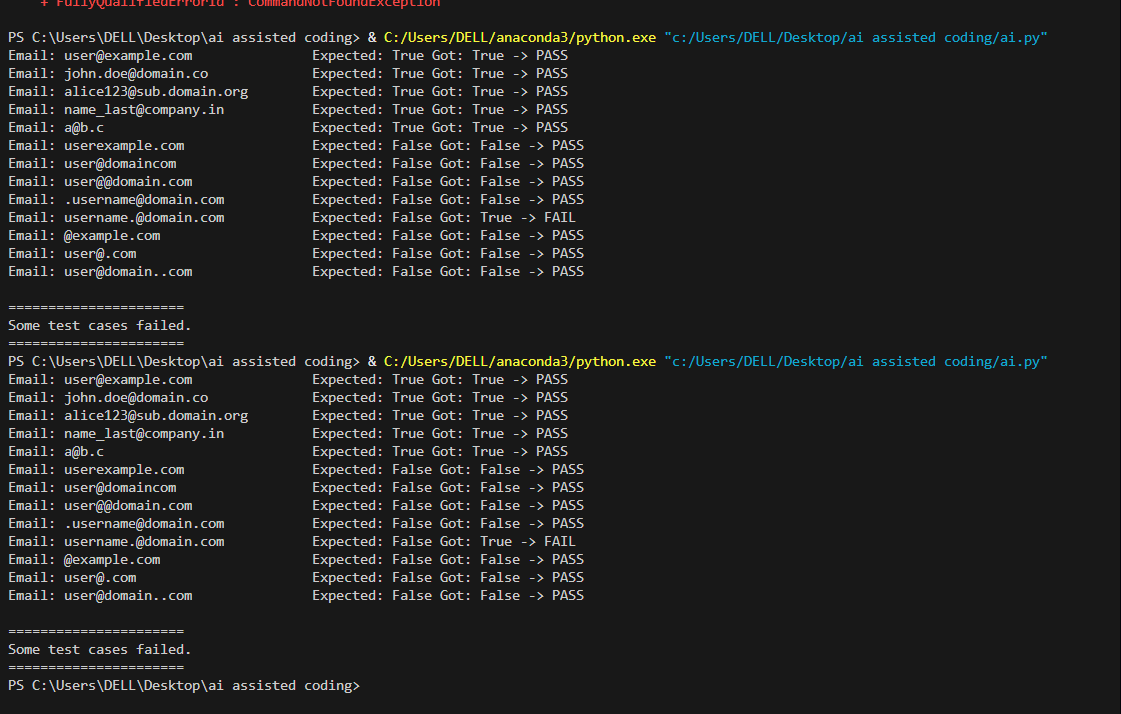
print("Some test cases failed.")

print("======================")

if \_name\_ == "\_main\_":

    run\_tests()

**output:**



**Tack 3:**

def is\_sentence\_palindrome(sentence):

cleaned = ""

for ch in sentence:

if ch.isalnum():

cleaned += ch.lower()

return cleaned == cleaned[::-1]

def run\_tests():

test\_cases = {

"A man a plan a canal Panama": True,

"No lemon, no melon": True,

"Was it a car or a cat I saw?": True,

"Madam In Eden, I'm Adam": True,

"Able , was I, ere I saw eLba": True,

"Never odd or even": True,

"Hello World": False,

"Python is fun": False,

"": True,

"12321": True,

"12345": False,

"Race car": True,

"Step on no pets": True

}

all\_passed = True

for sentence, expected in test\_cases.items():

result = is\_sentence\_palindrome(sentence)

status = "PASS" if result == expected else "FAIL"

if status == "FAIL":

all\_passed = False

print(f"Sentence: {sentence!r:35} Expected: {expected} Got: {result} -> {status}")

print("\n======================")

if all\_passed:

print("All test cases passed!")

else:

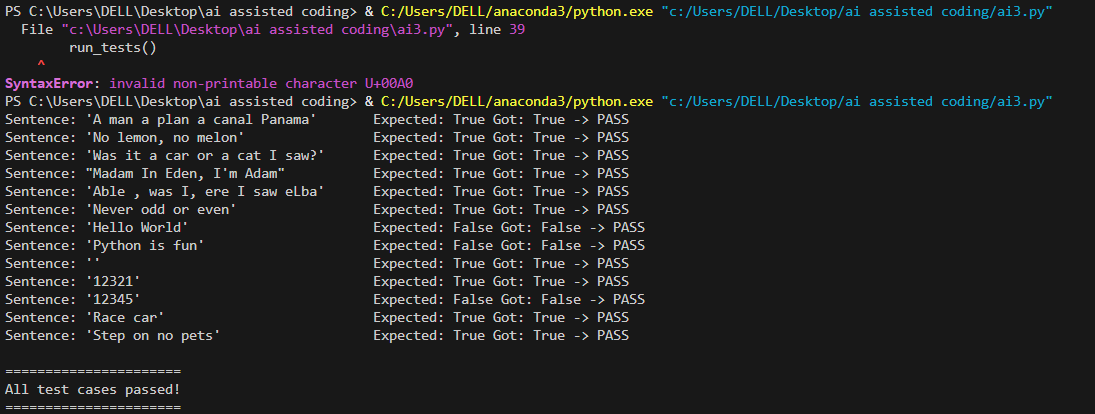
print("Some test cases failed.")

print("======================")

if \_name\_ == "\_main\_":

    run\_tests()

**output:**



**Tack 4:**

class ShoppingCart:

def \_\_init\_\_(self):

self.items = {}

def add\_item(self, name, price):

if not isinstance(name, str) or not isinstance(price, (int, float)):

return "Invalid"

if price < 0:

return "Invalid"

self.items[name] = self.items.get(name, 0) + price

return "Added"

def remove\_item(self, name):

if name in self.items:

del self.items[name]

return "Removed"

return "Not Found"

def total\_cost(self):

return sum(self.items.values())

def run\_tests():

cart = ShoppingCart()

test\_results = []

test\_results.append(("Add apple 10", cart.add\_item("apple", 10), "Added"))

test\_results.append(("Add banana 20", cart.add\_item("banana", 20), "Added"))

test\_results.append(("Add apple again 10", cart.add\_item("apple", 10), "Added"))

test\_results.append(("Total after adds", cart.total\_cost(), 40))

test\_results.append(("Remove banana", cart.remove\_item("banana"), "Removed"))

test\_results.append(("Total after remove", cart.total\_cost(), 20))

test\_results.append(("Remove orange (not in cart)", cart.remove\_item("orange"), "Not Found"))

test\_results.append(("Add invalid price", cart.add\_item("grape", -5), "Invalid"))

test\_results.append(("Add invalid name", cart.add\_item(123, 5), "Invalid"))

test\_results.append(("Final total", cart.total\_cost(), 20))

all\_passed = True

for desc, result, expected in test\_results:

status = "PASS" if result == expected else "FAIL"

if status == "FAIL":

all\_passed = False

print(f"{desc:30} Expected: {expected} Got: {result} -> {status}")

print("\n======================")

if all\_passed:

print("All test cases passed!")

else:

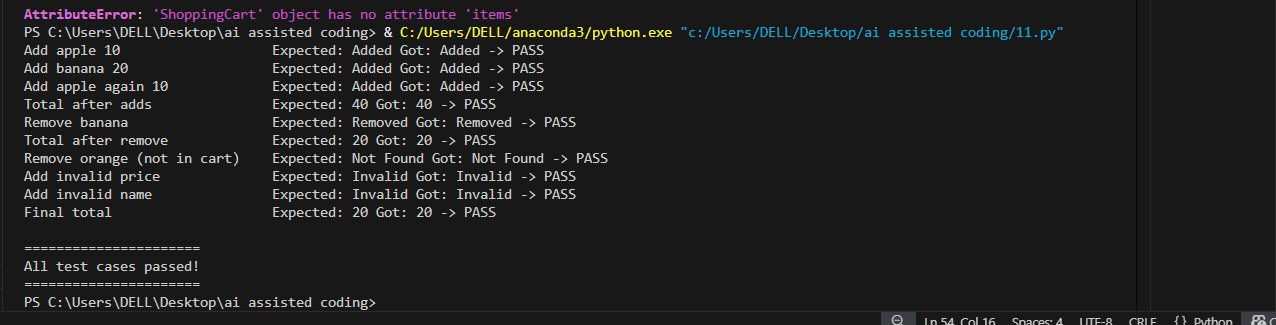
print("Some test cases failed.")

print("======================")

if \_\_name\_\_ == "\_\_main\_\_":

run\_tests()

**output:**

****

**Tack:05**

def convert\_date\_format(date\_str):

if not isinstance(date\_str, str):

return "Invalid"

parts = date\_str.split("-")

if len(parts) != 3:

return "Invalid"

year\_str, month\_str, day\_str = parts

if not (year\_str.isdigit() and month\_str.isdigit() and day\_str.isdigit()):

return "Invalid"

if len(year\_str) != 4 or len(month\_str) != 2 or len(day\_str) != 2:

return "Invalid"

# Fix: Add numerical validation for month and day

try:

month = int(month\_str)

day = int(day\_str)

if not (1 <= month <= 12 and 1 <= day <= 31):

return "Invalid"

except ValueError:

return "Invalid"

return f"{day\_str}-{month\_str}-{year\_str}"

def run\_tests():

test\_cases = {

"2023-10-15": "15-10-2023",

"1999-01-01": "01-01-1999",

"2000-12-31": "31-12-2000",

"2025-09-03": "03-09-2025",

"2023-07-04": "04-07-2023",

"abcd-12-01": "Invalid",

"2023-13-01": "Invalid", # Corrected: This is an invalid month

"2023-2-5": "Invalid",

"20231015": "Invalid",

12345: "Invalid",

"": "Invalid"

}

all\_passed = True

for date\_str, expected in test\_cases.items():

result = convert\_date\_format(date\_str)

status = "PASS" if result == expected else "FAIL"

if status == "FAIL":

all\_passed = False

print(f"Input: {str(date\_str):12} Expected: {expected:12} Got: {result:12} -> {status}")

print("\n======================")

if all\_passed:

print("All test cases passed!")

else:

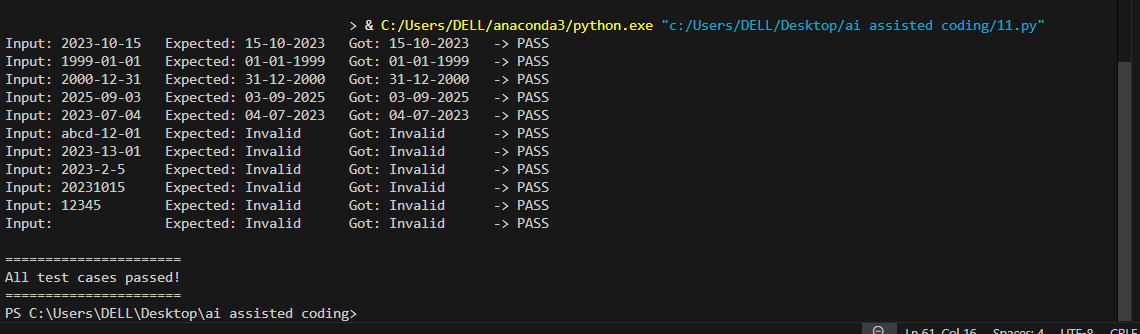
print("Some test cases failed.")

print("======================")

if \_\_name\_\_ == "\_\_main\_\_":

run\_tests()

**output:**

****