

		AI ASSISTED CODING	
		NAME: SIRIVELLA SANJANA ROLL NO:2403A510D4 ASSIGNMENT : 8.3	
		Task Description#1 Use AI to generate test cases for is_valid_email(email) and then implement the validator function. Requirements: <ul style="list-style-type: none">• Must contain @ and . characters.• Must not start or end with special characters.• Should not allow multiple @. Expected Output#1 <ul style="list-style-type: none">• Email validation logic passing all test cases PROMPT: write a python function using to generate gmailAdress code is_valid_email(email) and then implement the validator function. Requirements: <ul style="list-style-type: none">• Must contain @ and . characters.• Must not start or end with special characters.• Should not allow multiple @. CODE:	
1		<pre>task1.py > ... 1 import re 2 3 def is_valid_email(email): 4 # Check for exactly one '@' 5 if email.count('@') != 1: 6 return False 7 8 # Check for at least one '.' after '@' 9 if '.' not in email.split('@')[1]: 10 return False 11 12 # Regex pattern to enforce: 13 # - Starts with alphanumeric 14 # - No special char at start or end 15 # - One @ 16 # - At least one . in domain 17 pattern = r'^[a-zA-Z0-9][\w\.-]*[a-zA-Z0-9]@[a-zA-Z0-9]+\.[a-zA-Z]{2,}\$' 18 19 return re.match(pattern, email) is not None 20 21 # Dynamically get input from user 22 if __name__ == "__main__": 23 email_input = input("Enter your email: ") 24 25 if is_valid_email(email_input): 26 print("✅ Valid email.") 27 else: 28 print("❌ Invalid email. Make sure it:") 29 print("- Contains exactly one '@'") 30 print("- Contains at least one '.' after '@'") 31 print("- Does not start or end with special characters") 32</pre>	

OUTPUT:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\keerthi priya\Desktop\ai lab> &"C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai lab/task1.py"
Enter your email: jupakakeerthipriya@gmail.com
✔ Valid email.
PS C:\Users\keerthi priya\Desktop\ai lab> &"C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai lab/task1.py"
Enter your email: jupakakeerthipriya@gmail.com
✘ Invalid email. Make sure it:
- Contains exactly one '@'
- Contains at least one '.' after '@'
- Does not start or end with special characters
PS C:\Users\keerthi priya\Desktop\ai lab> &"C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai lab/task1.py"
Enter your email: jupakakeerthipriya@gmail.com
✘ Invalid email. Make sure it:
- Contains exactly one '@'
- Contains at least one '.' after '@'
- Does not start or end with special characters
PS C:\Users\keerthi priya\Desktop\ai lab>

```

Task Description#2 (Loops)

- Ask AI to generate test cases for `assign_grade(score)` function. Handle boundary and invalid inputs.

Requirements

- AI should generate test cases for `assign_grade(score)` where: 90-100: A, 80-89: B, 70-79: C, 60-69: D, <60: F
- Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

Expected Output#2

Grade assignment function passing test suite

PROMPT:

write a python code

for assign_grade(score) function. Handle boundary and invalid inputs.

Requirements

- AI should generate test cases for `assign_grade(score)` where: 90-100: A, 80-89: B, 70-79: C, 60-69: D, <60: F
- Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

CODE:

```

Welcome task1.py 1 task2.py
task2.py > ...
1 def assign_grade(score):
2     try:
3         # Check if input is None or empty string
4         if score is None or str(score).strip() == "":
5             return "Invalid input: score cannot be empty."
6
7         # Try converting to float
8         score = float(score)
9
10        # Check if score is within valid range
11        if score < 0 or score > 100:
12            return "Invalid score: must be between 0 and 100."
13        elif score >= 90:
14            return "A"
15        elif score >= 80:
16            return "B"
17        elif score >= 70:
18            return "C"
19        elif score >= 60:
20            return "D"
21        else:
22            return "F"
23    except (ValueError, TypeError):
24        return "Invalid input: score must be a number."
25
26    # Dynamic input from user
27    if __name__ == "__main__":
28        user_input = input("Enter your score: ")
29        result = assign_grade(user_input)
30        print(f"Grade: {result}")
31
32        print("\nRunning test cases...\n")
33
34        # Auto test cases including boundaries and invalid inputs
35        test_scores = [100, 90, 89, 80, 79, 70, 69, 60, 59, 0, -5, 105, "eighty", "", None]
36
37        for test in test_scores:

```

```

print("\nRunning test cases...\n")

# Auto test cases including boundaries and invalid inputs
test_scores = [100, 90, 89, 80, 79, 70, 69, 60, 59, 0, -5, 105, "eighty", "", None]

for test in test_scores:
    grade = assign_grade(test)
    print(f"Input: {repr(test):>9} → Grade: {grade}")

```

OUTPUT:

```
PROBLEMS ⓘ OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
.exe" "c:/Users/keerthi priya/Desktop/ai lab/task2.py"  
Enter your score: 80  
Grade: B  
  
Running test cases...  
  
Input:      100 → Grade: A  
Input:      90 → Grade: A  
Input:      89 → Grade: B  
Input:      80 → Grade: B  
Input:      79 → Grade: C  
Input:      70 → Grade: C  
Input:      69 → Grade: D  
Input:      60 → Grade: D  
Input:      59 → Grade: F  
Input:       0 → Grade: F  
Input:     -5 → Grade: Invalid score: must be between 0 and 100.  
Input:     59 → Grade: F  
Input:      0 → Grade: F  
Input:     -5 → Grade: Invalid score: must be between 0 and 100.  
Input:    -5 → Grade: Invalid score: must be between 0 and 100.  
Input:   105 → Grade: Invalid score: must be between 0 and 100.  
Input: 'eighty' → Grade: Invalid input: score must be a number.  
Input: '' → Grade: Invalid input: score cannot be empty.  
Input: 'eighty' → Grade: Invalid input: score must be a number.  
Input: '' → Grade: Invalid input: score cannot be empty.  
Input: None → Grade: Invalid input: score cannot be empty.  
PS C:\Users\keerthi priya\Desktop\ai lab>
```

Task Description#3

- Generate test cases using AI for `is_sentence_palindrome(sentence)`. Ignore case, punctuation, and spaces

Requirement

- Ask AI to create test cases for `is_sentence_palindrome(sentence)` (ignores case, spaces, and punctuation).
- Example:
"A man a plan a canal Panama" → True

Expected Output#3

- Function returns True/False for cleaned sentences
- Implement the function to pass AI-generated tests.

PROMPT:

Write a python code for `is_sentence_palindrome(sentence)`. Ignore case, punctuation, and spaces

Requirement

- Ask AI to create test cases for `is_sentence_palindrome(sentence)` (ignores case, spaces, and punctuation).

- Example:
"A man a plan a canal Panama" \rightarrow True.

CODE:

```
Welcome task1.py 1 task2.py task3.py X
task3.py > ...
1 import string
2
3 def is_sentence_palindrome(sentence):
4     # Remove punctuation and spaces, and convert to lowercase
5     cleaned = ''.join(
6         ch.lower() for ch in sentence if ch.isalnum()
7     )
8     return cleaned == cleaned[::-1]
9
10 # Dynamic input
11 if __name__ == "__main__":
12     user_input = input("Enter a sentence: ")
13     result = is_sentence_palindrome(user_input)
14     print(f"Is palindrome? {'✅ Yes' if result else '❌ No'}")
15
16     print("\nRunning test cases...\n")
17
18     test_cases = [
19         "A man a plan a canal Panama": True,
20         "No lemon, no melon": True,
21         "Was it a car or a cat I saw?": True,
22         "Madam, in Eden, I'm Adam": True,
23         "Hello World": False,
24         "": True, # Empty string is considered a palindrome
25         "12321": True,
26         "12345": False,
27         "Eva, can I see bees in a cave?": True,
28         "Not a palindrome": False,
29     ]
30
31     for sentence, expected in test_cases.items():
32         result = is_sentence_palindrome(sentence)
33         print(f"Input: {repr(sentence):40} → Expected: {expected} | Got: {result} | {'✅' if result == expected else '❌'}")
34
```

OUTPUT:

```
PROBLEMS Python DEBUG CONSOLE TERMINAL TASKS
.exe "c:/Users/keerthi priya/Desktop/ai lab/task3.py"
Enter a sentence: No lemon,no melon
Is palindrome? ✅ Yes

Running test cases...

Input: 'A man a plan a canal Panama' → Expected: True | Got: True | ✅
Input: 'No lemon, no melon' → Expected: True | Got: True | ✅
Input: 'Was it a car or a cat I saw?' → Expected: True | Got: True | ✅
Input: 'Madam, in Eden, I'm Adam' → Expected: True | Got: True | ✅
Input: 'Hello World' → Expected: False | Got: False | ✅
Input: '' → Expected: True | Got: True | ✅
Input: '12321' → Expected: True | Got: True | ✅
Input: '12345' → Expected: False | Got: False | ✅
Input: 'Eva, can I see bees in a cave?' → Expected: True | Got: True | ✅
Input: 'Not a palindrome' → Expected: False | Got: False | ✅
PS C:/Users/keerthi priya/Desktop/ai lab> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11
.exe" "c:/Users/keerthi priya/Desktop/ai lab/task3.py"
Enter a sentence: [ ]
```

Task Description#4

- Let AI fix itPrompt AI to generate test cases for a ShoppingCart class (add_item, remove_item, total_cost).

Methods:

Add_item(name,orice)
Remove_item(name)
Total_cost()

Expected Output#4

- Full class with tested functionalities

PROMPT:

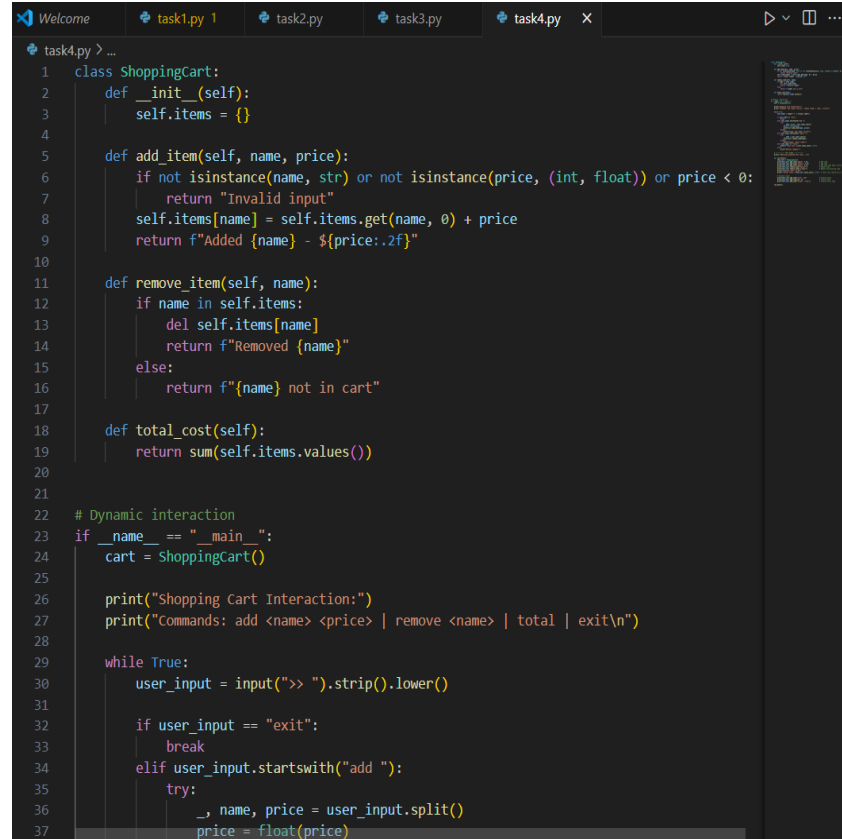
Write a python program to generate test cases for a ShoppingCart class (add_item, remove_item, total_cost).

Methods:

Add_item(name, orice)

Remove_item(name)

Total_cost() . give the code dynamically

CODE:

```
task4.py > ...
1 class ShoppingCart:
2     def __init__(self):
3         self.items = {}
4
5     def add_item(self, name, price):
6         if not isinstance(name, str) or not isinstance(price, (int, float)) or price < 0:
7             return "Invalid input"
8         self.items[name] = self.items.get(name, 0) + price
9         return f"Added {name} - ${price:.2f}"
10
11     def remove_item(self, name):
12         if name in self.items:
13             del self.items[name]
14             return f"Removed {name}"
15         else:
16             return f"{name} not in cart"
17
18     def total_cost(self):
19         return sum(self.items.values())
20
21
22 # Dynamic interaction
23 if __name__ == "__main__":
24     cart = ShoppingCart()
25
26     print("Shopping Cart Interaction:")
27     print("Commands: add <name> <price> | remove <name> | total | exit\n")
28
29     while True:
30         user_input = input(">> ").strip().lower()
31
32         if user_input == "exit":
33             break
34         elif user_input.startswith("add "):
35             try:
36                 _, name, price = user_input.split()
37                 price = float(price)
```

```
Welcome task1.py 1 task2.py task3.py task4.py X
task4.py > ...
35         try:
36             _, name, price = user_input.split()
37             price = float(price)
38             print(cart.add_item(name, price))
39         except:
40             print("Usage: add <name> <price>")
41     elif user_input.startswith("remove "):
42         try:
43             _, name = user_input.split()
44             print(cart.remove_item(name))
45         except:
46             print("Usage: remove <name>")
47     elif user_input == "total":
48         print(f"Total Cost: ${cart.total_cost():.2f}")
49     else:
50         print("Unknown command.")
51
52     # ----- TEST CASES -----
53     print("\nRunning automated test cases...\n")
54
55     def run_tests():
56         test_cart = ShoppingCart()
57         print(test_cart.add_item("apple", 1.5))           # Add item
58         print(test_cart.add_item("banana", 2.0))         # Add item
59         print(test_cart.add_item("apple", 0.5))          # Add same item again (price
60         print(test_cart.remove_item("banana"))           # Remove item
61         print(test_cart.remove_item("orange"))           # Remove non-existing item
62         print("Expected Total: $2.00")
63         print(f"Actual Total: ${test_cart.total_cost():.2f}") # Total cost should be 1.5
64
65         # Invalid inputs
66         print(test_cart.add_item("milk", -3))            # Invalid price
67         print(test_cart.add_item(123, 5))               # Invalid name
68         print(test_cart.add_item("bread", "free"))      # Invalid price type
69
70     run_tests()
71
```

OUTPUT:

```
>> add apple 1.5
Added apple - $1.50
>> add banana 2.5
Added banana - $2.50
Added banana - $2.50
>> remove apple
Removed apple
>> total
Total Cost: $2.50
>> total
Total Cost: $2.50
Total Cost: $2.50
>> exit
>> exit

Running automated test cases...

Added apple - $1.50

Running automated test cases...

Added apple - $1.50
Running automated test cases...

Added apple - $1.50

Added apple - $1.50
Added apple - $1.50
Added banana - $2.00
```

Task Description#5

- Use AI to write test cases for `convert_date_format(date_str)` to switch from "YYYY-MM-DD" to "DD-MM-YYYY".
Example: "2023-10-15" → "15-10-2023"

Expected Output#5

- Function converts input format correctly for all test cases

PROMPT:

Write a python program to generate `convert_date_format(date_str)` to switch from "YYYY-MM-DD" to "DD-MM-YYYY".

Example: "2023-10-15" → "15-10-2023". give code dynamically

CODE:

```
Welcome task1.py 1 task2.py task3.py task4.py task5.py
task5.py > ...
1 from datetime import datetime
2
3 def convert_date_format(date_str):
4     try:
5         # Parse input string as YYYY-MM-DD
6         date_obj = datetime.strptime(date_str, "%Y-%m-%d")
7         # Convert to DD-MM-YYYY format
8         return date_obj.strftime("%d-%m-%Y")
9     except ValueError:
10        return "❌ Invalid date format. Use YYYY-MM-DD."
11
12 # Dynamic user input
13 if __name__ == "__main__":
14     user_input = input("Enter a date (YYYY-MM-DD): ")
15     converted = convert_date_format(user_input)
16     print(f"Converted: {converted}")
17
18     print("\nRunning test cases...\n")
19
20     test_dates = [
21         "2023-10-15", # valid
22         "1999-01-01", # valid
23         "2020-02-29", # valid leap day
24         "2021-02-29", # invalid (non-leap year)
25         "15-10-2023", # invalid format
26         "2023/10/15", # invalid format
27         "", # empty
28         None # None input
29     ]
30
31     for test in test_dates:
32         try:
33             result = convert_date_format(test)
34         except Exception as e:
35             result = f"Error: {e}"
36         print(f"Input: {repr(test):>12} → Output: {result}")
37
```

OUTPUT:

