

# ASSIGNMENT- 8.3

Name: POOJITHA EDDE

HT.No: 2303A51356

Batch: 20

Task 1: Email Validation using TDD

Scenario

You are developing a user registration system that requires reliable email input validation.

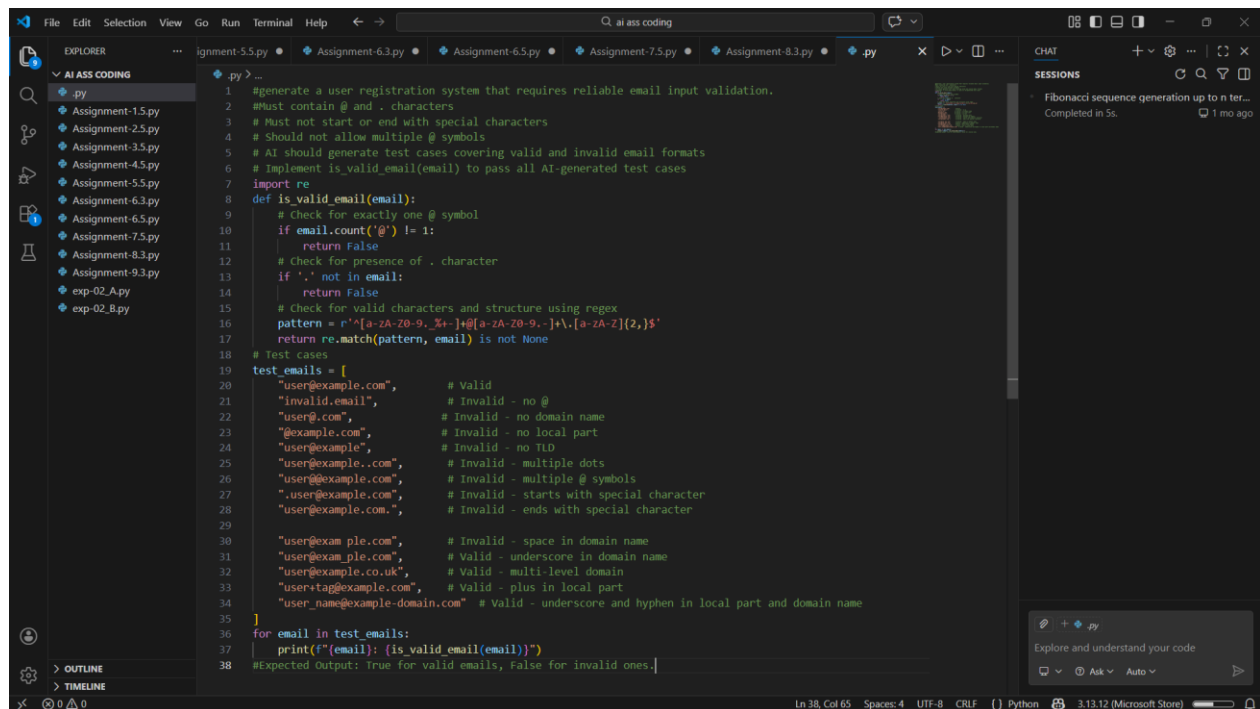
Requirements

- Must contain @ and . characters
- Must not start or end with special characters
- Should not allow multiple @ symbols
- AI should generate test cases covering valid and invalid email formats
- Implement is\_valid\_email(email) to pass all AI-generated test cases

Expected Output

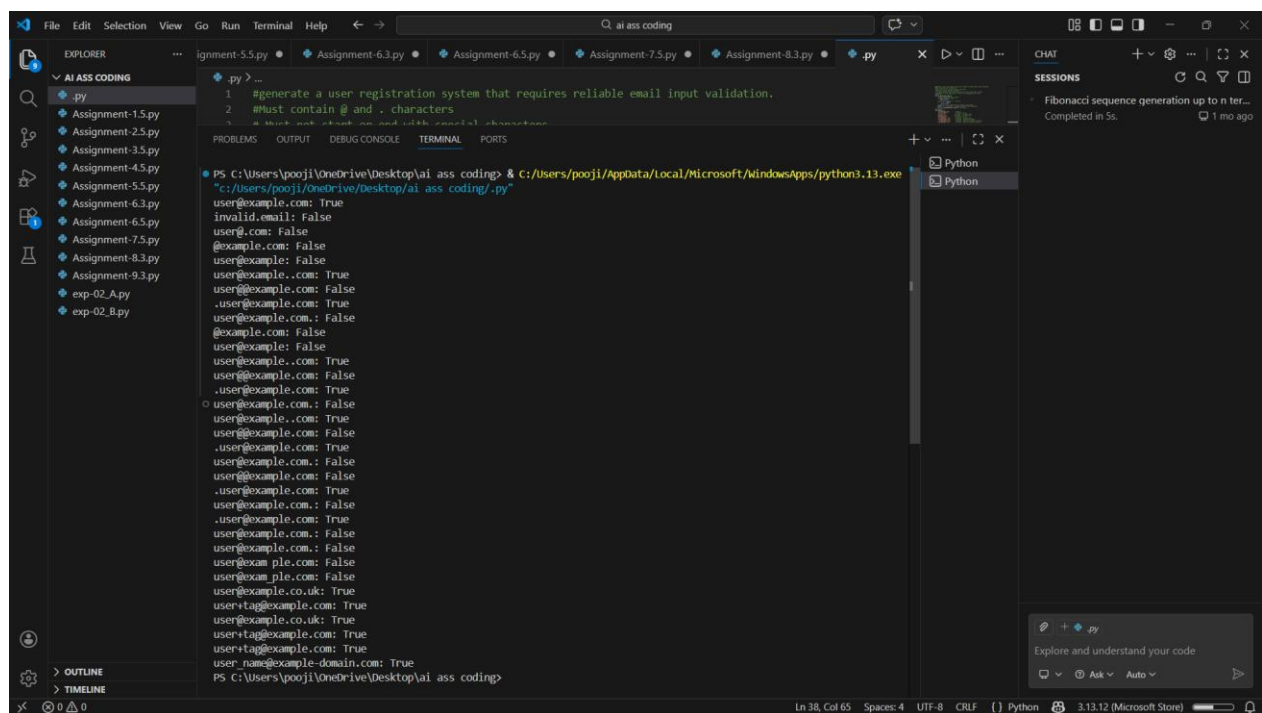
- Python function for email validation
- All AI-generated test cases pass successfully
- Invalid email formats are correctly rejected
- Valid email formats return True

CODE:



```
1 #generate a user registration system that requires reliable email input validation.
2 #Must contain @ and . characters
3 # Must not start or end with special characters
4 # Should not allow multiple @ symbols
5 # AI should generate test cases covering valid and invalid email formats
6 # Implement is_valid_email(email) to pass all AI-generated test cases
7 import re
8 def is_valid_email(email):
9     # Check for exactly one @ symbol
10    if email.count('@') != 1:
11        return False
12    # Check for presence of . character
13    if '.' not in email:
14        return False
15    # Check for valid characters and structure using regex
16    pattern = r'^[a-zA-Z0-9_+]+@[a-zA-Z0-9-]+\.[a-zA-Z]{2,}$'
17    return re.match(pattern, email) is not None
18 # Test cases
19 test_emails = [
20     "user@example.com",      # Valid
21     "invalid.email",        # Invalid - no @
22     "user@.com",            # Invalid - no domain name
23     "@example.com",         # Invalid - no local part
24     "user@example.",        # Invalid - no TLD
25     "user@example..com",    # Invalid - multiple dots
26     "user@example.com",     # Invalid - multiple @ symbols
27     ".user@example.com",    # Invalid - starts with special character
28     "user@example.com.",    # Invalid - ends with special character
29
30     "user@exam ple.com",    # Invalid - space in domain name
31     "user@exam_ple.com",    # Valid - underscore in domain name
32     "user@example.co.uk",   # Valid - multi-level domain
33     "user+tag@example.com", # Valid - plus in local part
34     "user_name@example-domain.com" # Valid - underscore and hyphen in local part and domain name
35 ]
36 for email in test_emails:
37     print(f"{email}: {is_valid_email(email)}")
38 #Expected Output: True for valid emails, False for invalid ones.
```

OUTPUT:



## Task 2: Grade Assignment using Loops

### Scenario

You are building an automated grading system for an online examination platform.

## Requirements

- AI should generate test cases for `assign_grade(score)` where:
  - 90–100 → A
  - 80–89 → B
  - 70–79 → C
  - 60–69 → D
  - Below 60 → F

- Include boundary values (60, 70, 80, 90)
- Include invalid inputs such as -5, 105, "eighty"
- Implement the function using a test-driven approach

### Expected Output

- Grade assignment function implemented in Python
- Boundary values handled correctly
- Invalid inputs handled gracefully
- All AI-generated test cases pass

CODE:

```
1 #generate automated grading system for an online examination platform.
2 # AI should generate test cases for assign_grade(score) where:
3 # 90-100 -> A
4 # 80-89 -> B
5 # 70-79 -> C
6 # 60-69 -> D
7 # Below 60 -> F
8 # Include boundary values (60, 70, 80, 90)
9 # Include invalid inputs such as -5, 105, "eighty"
10 # Implement the function using a test-driven approach
11 def assign_grade(score):
12     if isinstance(score, str) or score < 0 or score > 100:
13         return "Invalid input"
14     if score >= 90:
15         return 'A'
16     elif score >= 80:
17         return 'B'
18     elif score >= 70:
19         return 'C'
20     elif score >= 60:
21         return 'D'
22     else:
23         return 'F'
24 # Test cases
25 test_scores = [95, 85, 75, 65, 55, 60, 70, 80, 90, -5, 105, "eighty"]
26 for score in test_scores:
27     print(f"{score}: {assign_grade(score)}")
28 #Expected Output: Correct grade for valid scores, "Invalid input" for invalid ones.
29
```

OUTPUT:

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:\Users\pooji\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:\Users\pooji\OneDrive\Desktop\ai ass coding/.py"
95: A
85: B
75: C
65: D
55: F
60: D
70: C
80: B
90: A
-5: Invalid input
105: Invalid input
eighty: Invalid input
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

## Task 3: Sentence Palindrome Checker

### Scenario

You are developing a text-processing utility to analyze sentences.

### Requirements

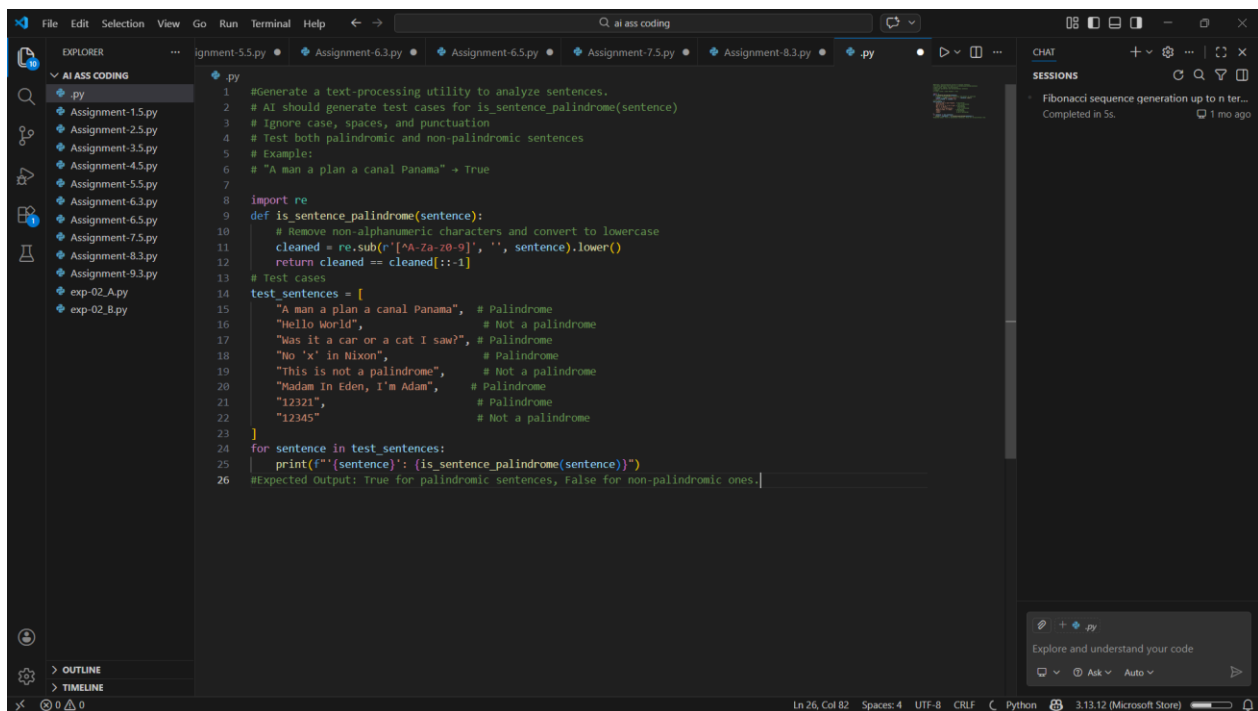
- AI should generate test cases for `is_sentence_palindrome(sentence)`

- Ignore case, spaces, and punctuation
- Test both palindromic and non-palindromic sentences
- Example:
  - "A man a plan a canal Panama" → True

#### Expected Output

- Function correctly identifies sentence palindromes
- Case and punctuation are ignored
- Returns True or False accurately
- All AI-generated test cases pass

#### CODE:



```

1 #Generate a text-processing utility to analyze sentences.
2 # AI should generate test cases for is_sentence_palindrome(sentence)
3 # Ignore case, spaces, and punctuation
4 # Test both palindromic and non-palindromic sentences
5 # Example:
6 # "A man a plan a canal Panama" → True
7
8 import re
9
10 def is_sentence_palindrome(sentence):
11     # Remove non-alphanumeric characters and convert to lowercase
12     cleaned = re.sub(r'[^A-Za-z0-9]', '', sentence).lower()
13     return cleaned == cleaned[::-1]
14
15 # Test cases
16 test_sentences = [
17     "A man a plan a canal Panama", # Palindrome
18     "Hello World", # Not a palindrome
19     "Was it a car or a cat I saw?", # Palindrome
20     "No 'x' in Nixon", # Palindrome
21     "This is not a palindrome", # Not a palindrome
22     "Madam In Eden, I'm Adam", # Palindrome
23     "12321", # Palindrome
24     "12345" # Not a palindrome
25 ]
26
27 for sentence in test_sentences:
28     print(f"{sentence}: {is_sentence_palindrome(sentence)}")
29
30 #Expected Output: True for palindromic sentences, False for non-palindromic ones.
  
```

#### OUTPUT:

The screenshot shows a VS Code editor with a Python file named `.py` open. The file contains a function `is_sentence_palindrome(sentence)` that cleans a sentence (removing non-alphanumeric characters and converting to lowercase) and checks if it is a palindrome. It includes several test cases in a list `test_sentences`.

```
1 #Generate a text-processing utility to analyze sentences.
2 # AI should generate test cases for is_sentence_palindrome(sentence)
3 # Ignore case, spaces, and punctuation
4 # Test both palindromic and non-palindromic sentences
5 # Example:
6 # "A man a plan a canal Panama" -> True
7
8 import re
9 def is_sentence_palindrome(sentence):
10     # Remove non-alphanumeric characters and convert to lowercase
11     cleaned = re.sub(r'[^A-Za-z0-9]', '', sentence).lower()
12     return cleaned == cleaned[::-1]
13
14 # Test cases
15 test_sentences = [
16     "A man a plan a canal Panama", # Palindrome
17     "Hello World", # Not a palindrome
18     "Was it a car or a cat I saw?", # Palindrome
19 ]
```

The terminal shows the command `python3.13.exe "c:/Users/pooji/o..."` and the output of the script:

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> python3.13.exe "c:/Users/pooji/o...
neDrive/Desktop/ai ass coding/.py"
'A man a plan a canal Panama': True
'Hello World': False
'Was it a car or a cat I saw?': True
'No 'x' in Nixon': True
'This is not a palindrome': False
'Madam In Eden, I'm Adam': True
'12321': True
'12345': False
```

## Task 4: ShoppingCart Class

### Scenario

You are designing a basic shopping cart module for an e-commerce application.

### Requirements

- AI should generate test cases for the ShoppingCart class
- Class must include the following methods:
  - `add_item(name, price)`
  - `remove_item(name)`
  - `total_cost()`
- Validate correct addition, removal, and cost calculation
- Handle empty cart scenarios

### Expected Output

- Fully implemented ShoppingCart class
- All methods pass AI-generated test cases
- Total cost is calculated accurately
- Items are added and removed correctly

CODE:

The screenshot shows the Visual Studio Code editor with a Python file named `.py` open. The file contains a `ShoppingCart` class and test cases. The class has methods `__init__`, `add_item`, `remove_item`, and `total_cost`. The test cases create a `ShoppingCart` object, add items, remove items, and calculate the total cost.

```
1 #Generate a basic shopping cart module for an e-commerce application.
2 # AI should generate test cases for the ShoppingCart class
3 # Class must include the following methods:
4 # add_item(name, price)
5 # remove_item(name)
6 # total_cost()
7 # Validate correct addition, removal, and cost calculation
8 # Handle empty cart scenarios
9
10 class ShoppingCart:
11     def __init__(self):
12         self.items = {}
13     def add_item(self, name, price):
14         self.items[name] = price
15     def remove_item(self, name):
16         if name in self.items:
17             del self.items[name]
18     def total_cost(self):
19         return sum(self.items.values())
20
21 # Test cases
22 cart = ShoppingCart()
23 cart.add_item("Book", 12.99)
24 cart.add_item("Pen", 1.50)
25 print(f"Total cost after adding items: {cart.total_cost()}") # Expected: 14.49
26 cart.remove_item("Pen")
27 print(f"Total cost after removing Pen: {cart.total_cost()}") # Expected: 12.99
28 cart.remove_item("Notebook") # Removing non-existing item
29 print(f"Total cost after trying to remove non-existing item: {cart.total_cost()}") # Expected: 12.99
30 cart.remove_item("Book")
31 print(f"Total cost after removing Book: {cart.total_cost()}") # Expected: 0.0
```

OUTPUT:

The screenshot shows the Visual Studio Code editor with the same Python file open. The output of the script is displayed in the terminal window at the bottom. The output shows the total cost after adding items, removing items, and calculating the total cost.

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:\Users\pooji\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:\Users\pooji\OneDrive\Desktop\ai ass coding\py"
Total cost after adding items: 14.49
Total cost after removing Pen: 12.99
Total cost after trying to remove non-existing item: 12.99
Total cost after removing Book: 0
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

## Task 5: Date Format Conversion

### Scenario

You are creating a utility function to convert date formats for reports.

### Requirements

- AI should generate test cases for `convert_date_format(date_str)`

- Input format must be "YYYY-MM-DD" • Output format must be "DD-MM-YYYY"
- Example:  
– "2023-10-15" → "15-10-2023"

Expected Output

- Date conversion function implemented in Python
- Correct format conversion for all valid inputs
- All AI-generated test cases pass successfully

CODE:

```

1 #Generate utility function to convert date formats for reports.
2 # AI should generate test cases for convert_date_format(date_str)
3 # Input format must be "YYYY-MM-DD"
4 # Output format must be "DD-MM-YYYY"
5 # Example:
6 # "2023-10-15" → "15-10-2023"
7 from datetime import datetime
8
9 def convert_date_format(date_str):
10     try:
11         date_obj = datetime.strptime(date_str, "%Y-%m-%d")
12         return date_obj.strftime("%d-%m-%Y")
13     except ValueError:
14         return "Invalid date format"
15
16
17 # Test cases
18 test_dates = [
19     "2023-10-15", # Valid date
20     "1990-01-01", # Valid date
21     "2023/10/15", # Invalid format
22
23     "15-10-2023", # Invalid format
24     "2023-13-01", # Invalid month
25     "2023-00-10", # Invalid month
26     "2023-10-32", # Invalid day
27     "2023-10-00" # Invalid day
28 ]
29
30 for date in test_dates:
31     print(f"{date}: {convert_date_format(date)}")
32
33 #Expected Output: Correctly converted date for valid inputs, "Invalid date format" for invalid ones.

```

OUTPUT:

