

ASSIGNMENT-8.1

HT NO:2303A51644

Batch No:28

Course:AI Assisted Coding

Task Description 1: (Password Strength Validator – Apply AI in Security Context)

• **Task:** Apply AI to generate at least 3 assert test cases for `is_strong_password(password)` and implement the validator function.

Prompt:

Generate at least 3 assert test cases for a Python function `is_strong_password(password)`. Requirements:

- Minimum 8 characters
 - Must contain uppercase, lowercase, digit, and special character
 - Must not contain spaces
- Then implement the function so all test cases pass.

1.General Testing

```
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\"':{}|<>]", password):
        return False
    return True

print(is_strong_password("Abcd@123"))
```

Output:

```
ant_Coding/task1(ass-8.1).py"
○ True
```

2. Assert testing

```
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\"':{}|<>]", password):
        return False
    return True

assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == False
💡
print("Assert tests passed")
```

Output:

```
○ ant_Coding/task1(ass-8.1).py"
Assert tests passed
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING C
```

3. Unit testing

```

import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\"':{}|<>]", password):
        return False
    return True

```

Create Next python file to run Test Cases:

```

import unittest
from task1 import is_strong_password

class TestPassword(unittest.TestCase):

    def test_valid(self):
        self.assertTrue(is_strong_password("Abcd@123"))

    def test_invalid(self):
        self.assertFalse(is_strong_password("abcd123"))

if __name__ == "__main__":
    unittest.main()

```

Output:

```
PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING C
URSES/AI_Assisant_Coding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python test_task1_unittest.py
..
-----
Ran 2 tests in 0.001s
OK
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> 
```

4. Pytest and Output:

```
!pip install pytest
```

```
Requirement already satisfied: pytest in /usr/local/lib/python3.12/dist-packages (8.4.2)
Requirement already satisfied: iniconfig>=1 in /usr/local/lib/python3.12/dist-packages (from pytest) (2.3.0)
Requirement already satisfied: packaging>=20 in /usr/local/lib/python3.12/dist-packages (from pytest) (26.0)
Requirement already satisfied: pluggy<2,>=1.5 in /usr/local/lib/python3.12/dist-packages (from pytest) (1.6.0)
Requirement already satisfied: pygments>=2.7.2 in /usr/local/lib/python3.12/dist-packages (from pytest) (2.19.2)
```

```
%%writefile task1.py
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\"':{}|<>]", password):
        return False
    return True
```

```
... Writing task1.py
```

```
▶ %%writefile test_task1.py
from task1 import is_strong_password

def test_valid():
    assert is_strong_password("Abcd@123") == True

def test_invalid():
    assert is_strong_password("abcd123") == False
```

*** Writing test_task1.py

▶ !pytest -v

```
*** ===== test session starts =====
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED [ 10%]
test_lab8.py::test_password_invalid PASSED [ 20%]
test_lab8.py::test_positive PASSED [ 30%]
test_lab8.py::test_negative PASSED [ 40%]
test_lab8.py::test_anagram_true PASSED [ 50%]
test_lab8.py::test_anagram_false PASSED [ 60%]
test_lab8.py::test_inventory PASSED [ 70%]
test_lab8.py::test_valid_date PASSED [ 80%]
test_task1.py::test_valid PASSED [ 90%]
test_task1.py::test_invalid PASSED [100%]

===== 10 passed in 0.03s =====
```

Justification:

This task applies AI in a **security context**.

- AI generates edge-case test scenarios.
- TDD approach ensures validation rules are clearly defined before coding.
- Improves reliability of password validation.
- Prevents weak or insecure passwords.
- Demonstrates input validation and regex handling.

- Shows importance of secure coding
- Encourages validation before implementation

Task Description 2: (Number Classification with Loops – Apply AI for Edge Case Handling)

- **Task:** Use AI to generate at least 3 assert test cases for a `classify_number(n)` function. Implement using loops.

Prompt:

Generate at least 3 assert test cases for a function `classify_number(n)` that:

- Returns "Positive", "Negative", or "Zero"
 - Handles invalid inputs like strings and None
 - Includes boundary values (-1, 0, 1)
- Then implement the function using loops so all tests pass.

1. General Testing and Output

```
task1(ass-8.1).py > ...
1  def classify_number(n):
2      if not isinstance(n, (int, float)):
3          return "Invalid Input"
4      if n > 0:
5          return "Positive"
6      elif n < 0:
7          return "Negative"
8      else:
9          return "Zero"
10
11  print(classify_number(10))
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python + v [] [X] ...

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/task1(ass-8.1).py"
Positive ...
```

2. Assert testing and Output

```

1  def classify_number(n):
2      if not isinstance(n, (int, float)):
3          return "Invalid Input"
4      if n > 0:
5          return "Positive"
6      elif n < 0:
7          return "Negative"
8      else:
9          return "Zero"
10
11
12  #generate assert statements code to test the function classify_number with different inputs.
13
14  assert classify_number(10) == "Positive"
15  assert classify_number(-5) == "Negative"
16  assert classify_number(0) == "Zero"
17  assert classify_number("abc") == "Invalid Input"
18  print("All test cases passed!")

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

Python + -

```

ant_Coding/task1(ass-8.1).py"
All test cases passed!
All test cases passed!
All test cases passed!
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>

```

3. Unit testing and Output

```

import re
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"

```

Create Next python file to run Test Cases:


```

3 import unittest
4 from task import classify_number
5 class TestNumber(unittest.TestCase):
6     def test_positive(self):
7         self.assertEqual(classify_number(10), "Positive")
8
9     def test_negative(self):
10        self.assertEqual(classify_number(-5), "Negative")
11
12    def test_zero(self):
13        self.assertEqual(classify_number(0), "Zero")
14
15    def test_invalid(self):
16        self.assertEqual(classify_number("abc"), "Invalid Input")
17
18 if __name__ == "__main__":
19     unittest.main()

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

Python + v

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COU
ant_Coding/test_task1_unittest.py"
```

....

Ran 4 tests in 0.001s

OK

4. Pytest and Output:

```

▶ def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"

```

```

def test_positive():
    assert classify_number(10) == "Positive"

def test_invalid():
    assert classify_number("abc") == "Invalid Input"

```



```

▶ lpytest -v

... ===== test session starts =====
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED [ 10%]
test_lab8.py::test_password_invalid PASSED [ 20%]
test_lab8.py::test_positive PASSED [ 30%]
test_lab8.py::test_negative PASSED [ 40%]
test_lab8.py::test_anagram_true PASSED [ 50%]
test_lab8.py::test_anagram_false PASSED [ 60%]
test_lab8.py::test_inventory PASSED [ 70%]
test_lab8.py::test_valid_date PASSED [ 80%]
test_task1.py::test_valid PASSED [ 90%]
test_task1.py::test_invalid PASSED [100%]

===== 10 passed in 0.02s =====

```

Justification:

This task focuses on **edge case handling**.

- AI helps identify boundary conditions.
- Ensures handling of invalid inputs.
- Demonstrates defensive programming.
- Reinforces loop usage and type checking.

→ Encourages robustness

→ Improves error handling

→ Demonstrates TDD validation logic

Task Description 3: (Anagram Checker – Apply AI for String

Analysis)

- **Task:** Use AI to generate at least 3 assert test cases for

`is_anagram(str1, str2)` and implement the function.

Prompt:

Generate at least 3 assert test cases for a function `is_anagram(str1, str2)` that:

- Ignores case
- Ignores spaces and punctuation

- Handles empty strings
Then implement the function so all tests pass.

1. General Testing and Output:

```

15 def is_anagram(str1, str2):
16     return sorted(str1.lower().replace(" ", "")) == sorted(str2.lower().replace(" ", ""))
17 # Example usage:
18 word1 = "Listen"
19 word2 = "Silent"
20 result = is_anagram(word1, word2)
21 print(result) # Output: True

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python + v [] [] ...

```

(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/task1(ass-8.1).py"
True

```

2. Assert testing and Output:

```

15 def is_anagram(str1, str2):
16     return sorted(str1.lower().replace(" ", "")) == sorted(str2.lower().replace(" ", ""))
17 assert is_anagram("listen", "silent") == True
18 assert is_anagram("hello", "world") == False
19 print("All test cases passed!")

```

esktop\LEARNING COURSES\AI_Assisant_Coding\task.py

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python + v [] [] ...

```

(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/task1(ass-8.1).py"
All test cases passed!
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>

```

3. Unit testing and Output:

```
import re
def is_anagram(str1, str2):
    return sorted(str1.lower().replace(" ", "")) == sorted(str2.lower().replace(" ", ""))
```

Create Next python file to run Test Cases:

```
10 > import unittest...
12 class TestAnagram(unittest.TestCase):
13     def test_true(self):
14         self.assertTrue(is_anagram("listen", "silent"))
15     def test_false(self):
16         self.assertFalse(is_anagram("hello", "world"))
17 if __name__ == "__main__":
18     unittest.main()
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python + v

(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python test_task_unittest.py

Ran 2 tests in 0.001s

OK

4. Pytest and Output:

```
def is_anagram(str1, str2):
    return sorted(str1.lower().replace(" ", "")) == sorted(str2.lower().replace(" ", ""))
```

```
def test_anagram():
    assert is_anagram("listen", "silent") == True
```

+ Code + Text

```

▶ !pytest -v

... ===== test session starts =====
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED [ 10%]
test_lab8.py::test_password_invalid PASSED [ 20%]
test_lab8.py::test_positive PASSED [ 30%]
test_lab8.py::test_negative PASSED [ 40%]
test_lab8.py::test_anagram_true PASSED [ 50%]
test_lab8.py::test_anagram_false PASSED [ 60%]
test_lab8.py::test_inventory PASSED [ 70%]
test_lab8.py::test_valid_date PASSED [ 80%]
test_task1.py::test_valid PASSED [ 90%]
test_task1.py::test_invalid PASSED [100%]

===== 10 passed in 0.02s =====

```

Justification:

This task applies AI for **string analysis**.

- AI helps detect tricky test cases.
- Validates normalization (case folding, filtering).
- Ensures string preprocessing before comparison.
- Covers empty and identical cases.

→ Improves text processing logic
 → Demonstrates data cleaning
 → Shows importance of normalization

Task Description 4: (Inventory Class – Apply AI to Simulate Real-

World Inventory System)

- **Task:** Ask AI to generate at least 3 assert-based tests for an

Inventory class with stock management.

Prompt:

Generate at least 3 assert-based test cases for an Inventory class that supports:

- `add_item(name, quantity)`
 - `remove_item(name, quantity)`
 - `get_stock(name)`
- Then implement the class so all assertions pass.

1. General Testing and Output:

```
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False
    def get_stock(self, item):
        return self.stock.get(item, 0)
```

```
# Example usage
inventory = Inventory()
inventory.add_item("apple", 10)
inventory.add_item("banana", 5)
print(inventory.get_stock("apple")) # Output: 10
print(inventory.get_stock("banana")) # Output: 5
print(inventory.remove_item("apple", 3)) # Output: True
print(inventory.get_stock("apple")) # Output: 7
```

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding\task1(ass-8.1).py"
10
5
True
○ 5
True
7
7
7
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> █
```

2.Assert testing and Output:

```
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False
    def get_stock(self, item):
        return self.stock.get(item, 0)

inv = Inventory()
inv.add_item("Pen",10)
assert inv.get_stock("Pen") == 10
inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5
inv.remove_item("Pen", 10)
assert inv.get_stock("Pen") == 5
print("All tests passed!")
```

All tests passed!

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> █
```


3. Unit testing and Output:

```
import re
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False
    def get_stock(self, item):
        return self.stock.get(item, 0)
```

Create Next python file to run Test Cases:

```
test_task_unittest.py > TestInventory > test_remove

10 import unittest
11 from task import Inventory
12 class TestInventory(unittest.TestCase):
13     def test_add(self):
14         inv = Inventory()
15         inv.add_item("Pen",10)
16         self.assertEqual(inv.get_stock("Pen"),10)
17     def test_remove(self):
18         inv = Inventory()
19         inv.add_item("Pen",10)
20         inv.remove_item("Pen", 5)
21         self.assertEqual(inv.get_stock("Pen"),5)
22         inv.remove_item("Pen", 10)
23         self.assertEqual(inv.get_stock("Pen"),5)
```

..

Ran 2 tests in 0.000s

OK

4. Pytest and Output:

```
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False
    def get_stock(self, item):
        return self.stock.get(item, 0)
```

```
▶ def test_inventory():
    inv = Inventory()
    inv.add_item("Pen", 10)
    assert inv.get_stock("Pen") == 10
    assert inv.remove_item("Pen", 5) == True
```

```
!pytest -v
```

```
===== test session starts =====
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED [ 10%]
test_lab8.py::test_password_invalid PASSED [ 20%]
test_lab8.py::test_positive PASSED [ 30%]
test_lab8.py::test_negative PASSED [ 40%]
test_lab8.py::test_anagram_true PASSED [ 50%]
test_lab8.py::test_anagram_false PASSED [ 60%]
test_lab8.py::test_inventory PASSED [ 70%]
test_lab8.py::test_valid_date PASSED [ 80%]
test_task1.py::test_valid PASSED [ 90%]
test_task1.py::test_invalid PASSED [100%]

===== 10 passed in 0.03s =====
```

Justification:

This task simulates a **real-world system**.

- AI generates realistic stock management tests.
- Encourages object-oriented design.
- Validates state updates inside a class.
- Demonstrates data storage using dictionaries.

→ Applies OOP concepts
→ Simulates real inventory system
→ Encourages state validation

Task Description 5 (Date Validation & Formatting – Apply AI for

Data Validation)

• **Task:** Use AI to generate at least 3 assert test cases for `validate_and_format_date(date_str)` to check and convert dates.

Prompt:

Generate at least 3 assert test cases for a function `validate_and_format_date(date_str)` that:

- Validates "MM/DD/YYYY" format
 - Handles invalid dates
 - Converts valid dates to "YYYY-MM-DD"
- Then implement the function so all test cases pass.

1. General Testing and Output:

```

10 from datetime import datetime
11
12 def validate_and_format_date(date_str):
13     try:
14         d = datetime.strptime(date_str, "%m/%d/%Y")
15         return d.strftime("%Y-%m-%d")
16     except:
17         return "Invalid Date"
18 # Test cases
19 print(validate_and_format_date("12/31/2020")) # Output: "2020-12-31"
20 print(validate_and_format_date("31/12/2020")) # Output: "Invalid Date"

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python + v [] []

RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI
 ant_Coding/task1(ass-8.1).py"
 2020-12-31
 Invalid Date
 (.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> []

2. Assert testing and Output:

```

10 from datetime import datetime
11
12 def validate_and_format_date(date_str):
13     try:
14         d = datetime.strptime(date_str, "%m/%d/%Y")
15         return d.strftime("%Y-%m-%d")
16     except:
17         return "Invalid Date"
18 assert validate_and_format_date("10/15/2023") == "2023-10-15"
19 assert validate_and_format_date("02/30/2023") == "Invalid Date"
20 assert validate_and_format_date("13/01/2023") == "Invalid Date"
21 assert validate_and_format_date("12/31/2023") == "2023-12-31"
22 print("All test cases passed!")

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python + v [] []

(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/task1(ass-8.1).py"
 ...
 (.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assisant_Coding/task1(ass-8.1).py"
 All test cases passed!

3. Unit testing

```
import re
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        d = datetime.strptime(date_str, "%m/%d/%Y")
        return d.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"
```

Create Next python file to run Test Cases:

```

10 import unittest
11 from task import validate_and_format_date
12 class TestDate(unittest.TestCase):
13     def test_valid(self):
14         self.assertEqual(validate_and_format_date("10/15/2023"), "2023-10-15")
15     def test_invalid_day(self):
16         self.assertEqual(validate_and_format_date("02/30/2023"), "Invalid Date")
17     def test_invalid_month(self):
18         self.assertEqual(validate_and_format_date("13/01/2023"), "Invalid Date")
19     def test_valid_end_of_year(self):
20         self.assertEqual(validate_and_format_date("12/31/2023"), "2023-12-31")
21
22 if __name__ == "__main__":
23     unittest.main()

```

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python + v []
```

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python test_task_unittest.py
....
-----
Ran 4 tests in 0.006s
```

4. Pytest and Output:

```
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        d = datetime.strptime(date_str, "%m/%d/%Y")
        return d.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"
```

```
def test_date():
    assert validate_and_format_date("10/15/2023") == "2023-10-15"
```

```
▶ !pytest -v

... ===== test session starts =====
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED [ 10%]
test_lab8.py::test_password_invalid PASSED [ 20%]
test_lab8.py::test_positive PASSED [ 30%]
test_lab8.py::test_negative PASSED [ 40%]
test_lab8.py::test_anagram_true PASSED [ 50%]
test_lab8.py::test_anagram_false PASSED [ 60%]
test_lab8.py::test_inventory PASSED [ 70%]
test_lab8.py::test_valid_date PASSED [ 80%]
test_task1.py::test_valid PASSED [ 90%]
test_task1.py::test_invalid PASSED [100%]

===== 10 passed in 0.02s =====
```

Justification:

This task focuses on **data validation**.

- AI generates invalid date cases (e.g., Feb 30).
- Ensures strict format validation.
- Demonstrates exception handling.
- Converts data into standardized format.

→ Reinforces error handling

→ Demonstrates datetime usage

→ Encourages clean data formatting