
Lab Assignment # 08

Course Title : AI Assistant Coding

Name of Student : V.SUSHMITHA

Enrollment No. : 2303A54055

Batch No. : 47-b

Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases

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.

• **Requirements:**

- o Password must have at least 8 characters.
- o Must include uppercase, lowercase, digit, and special character.
- o Must not contain spaces.

Example Assert Test Cases:

```
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == True
```

Expected Output #1: • Password validation logic passing all AI-generated test cases.

Output Screenshot:

The top screenshot shows the implementation of the `is_strong_password` function in `task01.py`. The function is designed to validate passwords based on several requirements: at least 8 characters, containing at least one uppercase letter, one lowercase letter, one digit, and one special character, and not containing spaces or leading/trailing spaces. The function uses a series of conditional checks to ensure these requirements are met.

```

1 # Generate at least 5 assert-based test cases for a function:
2 # def is_strong_password(password):
3 # Requirements:
4 # - At least 8 characters
5 # - Must contain uppercase letter
6 # - Must contain lowercase letter
7 # - Must contain digit
8 # - Must contain special character
9 # - Must not contain spaces
10 # Include edge cases such as:
11 # - Empty string
12 # - Only digits
13 # - Only uppercase
14 # - Password with spaces
15 # - Exactly 8 characters valid case
16 # Return only assert statements.
17 # Now implement the Python function:
18 # def is_strong_password(password):
19 # Make sure it passes all previously generated test cases.
20 # Use clean logic and proper condition checking.
21 # Do not use external libraries.
22 def is_strong_password(password):
23     if not isinstance(password, str):
24         return False
25
26     if len(password) < 8:
27         return False
28
29     if ' ' in password:
30         return False
31
32     if not any(char.isupper() for char in password):
33         return False
34
35     if not any(char.islower() for char in password):
36         return False
37
38     if not any(char.isdigit() for char in password):
39         return False
40
41     if not any(char in "!@#$%^&*()-_+[]{}|;'\",.<>/" for char in password):
42         return False
43
44     return True

```

The bottom screenshot shows the same file with test cases added to the `task01.py` file. The test cases use the `assert` statement to verify the function's behavior for various inputs, including valid passwords, empty strings, and passwords that fail one or more requirements. A print statement at the end of the test cases confirms that all tests passed successfully.

```

47 # Test Cases
48 assert is_strong_password("Password123!") == True
49 assert is_strong_password("") == False
50 assert is_strong_password("12345678") == False
51 assert is_strong_password("PASSWORD") == False
52 assert is_strong_password("Pass word!") == False
53 assert is_strong_password("Passw1!") == False
54 assert is_strong_password("Passw0rd!") == True
55 assert is_strong_password("P@ssw0rd") == True
56 assert is_strong_password("P@ss w0rd") == False
57 assert is_strong_password("P@ssw0r") == False
58
59 print("All test cases passed successfully 🎉")
60
61 # Analyze the password validation function.
62 # Suggest improvements for:
63 # - Readability
64 # - Security
65 # - Edge case handling
66 # - Performance
67
68 # Explain briefly.
69 # The function is_strong_password is straightforward and checks all the required conditions for a strong password.
70 # Improvements:

```

The bottom screenshot also shows the terminal output, which confirms that all test cases passed successfully.

```

Python 3.9.6
/usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab8.1/task01.py
anumandlarithika@ANUMANDLAS-MacBook-Air AI_Assisted_Lab % /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab8.1/task01.py
All test cases passed successfully 🎉
anumandlarithika@ANUMANDLAS-MacBook-Air AI_Assisted_Lab %

```

Explanation: This checks if a password is truly “strong” by making sure it has the right mix of characters and no spaces.

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.

• **Requirements:**

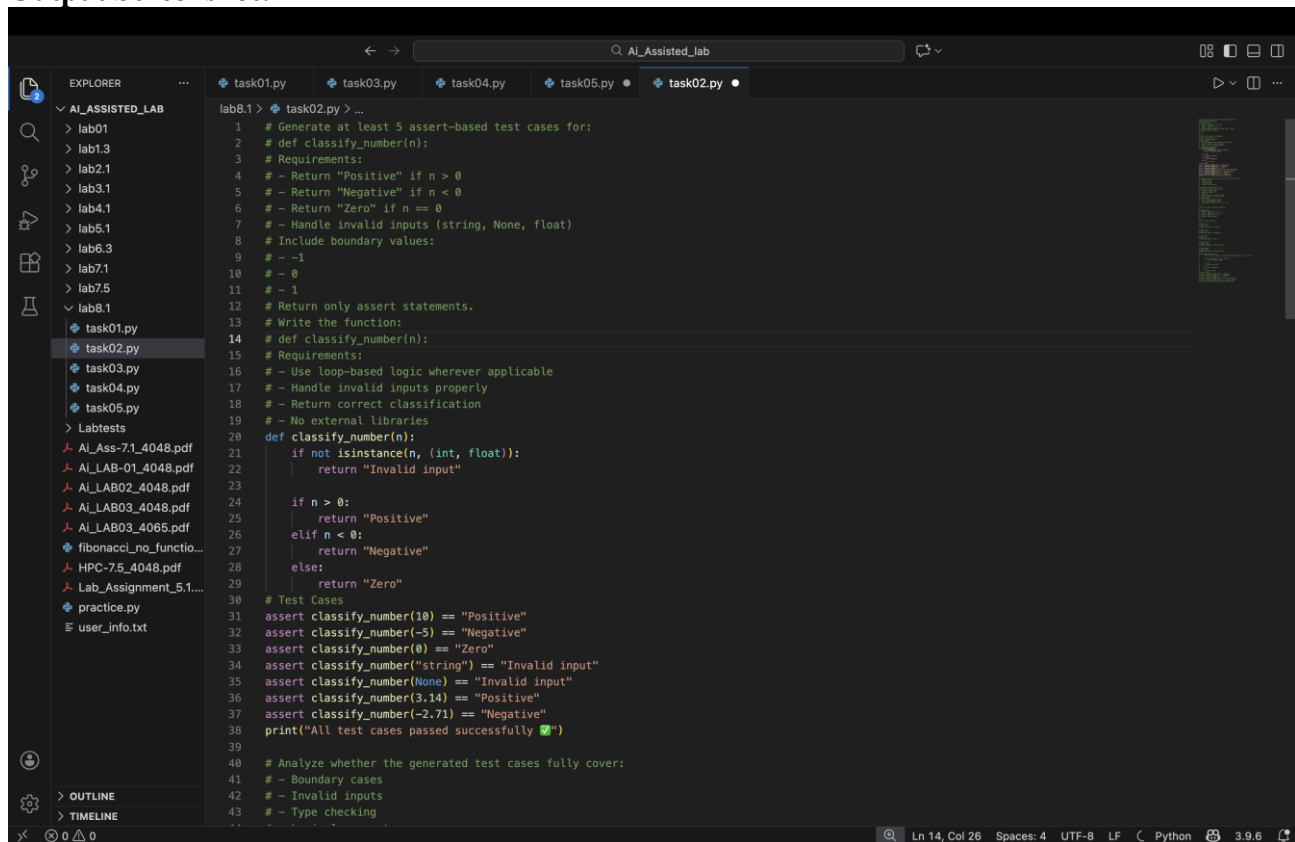
- o Classify numbers as Positive, Negative, or Zero.
- o Handle invalid inputs like strings and None.
- o Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:

```
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
```

Expected Output #2: • Classification logic passing all assert tests.

Output Screenshot:



The screenshot shows a VS Code editor window with a file explorer on the left and a code editor in the center. The file explorer shows a project named 'AI_ASSISTED_LAB' with several files and folders. The code editor displays a Python script named 'task02.py' with the following content:

```
lab8.1 > task02.py > ...
1 # Generate at least 5 assert-based test cases for:
2 # def classify_number(n):
3 # Requirements:
4 # - Return "Positive" if n > 0
5 # - Return "Negative" if n < 0
6 # - Return "Zero" if n == 0
7 # - Handle invalid inputs (string, None, float)
8 # Include boundary values:
9 # - -1
10 # - 0
11 # - 1
12 # Return only assert statements.
13 # Write the function:
14 # def classify_number(n):
15 # Requirements:
16 # - Use loop-based logic wherever applicable
17 # - Handle invalid inputs properly
18 # - Return correct classification
19 # - No external libraries
20 def classify_number(n):
21     if not isinstance(n, (int, float)):
22         return "Invalid input"
23
24     if n > 0:
25         return "Positive"
26     elif n < 0:
27         return "Negative"
28     else:
29         return "Zero"
30
31 # Test Cases
32 assert classify_number(10) == "Positive"
33 assert classify_number(-5) == "Negative"
34 assert classify_number(0) == "Zero"
35 assert classify_number("string") == "Invalid input"
36 assert classify_number(None) == "Invalid input"
37 assert classify_number(3.14) == "Positive"
38 assert classify_number(-2.71) == "Negative"
39 print("All test cases passed successfully 🎉")
40
41 # Analyze whether the generated test cases fully cover:
42 # - Boundary cases
43 # - Invalid inputs
44 # - Type checking
```

```
lab8.1 > task02.py > ...
1 # Generate at least 5 assert-based test cases for:
2 # def classify_number(n):
3 # Requirements:
4 # - Return "Positive" if n > 0
5 # - Return "Negative" if n < 0
6 # - Return "Zero" if n == 0
7 # - Handle invalid inputs (string, None, float)
8 # Include boundary values:
9 # - -1
10 # - 0
11 # - 1
12 # Return only assert statements.
13 # Write the function:
14 # def classify_number(n):
15 # Requirements:
16 # - Use loop-based logic wherever applicable
17 # - Handle invalid inputs properly
18 # - Return correct classification
19 # - No external libraries
20 def classify_number(n):
21     if not isinstance(n, (int, float)):
22         return "Invalid input"
23
24     if n > 0:
25         return "Positive"
26     elif n < 0:
27         return "Negative"
28     else:
29         return "Zero"
30
31 # Generate 5 test cases
32 test_cases = [-1, 0, 1, "hello", None]
33 for i, test_case in enumerate(test_cases):
34     assert classify_number(test_case) in ["Positive", "Negative", "Zero", "Invalid input"], f"Test case {i+1} failed"
35
36 # All test cases passed successfully
```

Explanation: This sorts numbers into positive, negative, or zero, while politely rejecting anything that isn't a number.

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.

- **Requirements:**
 - o Ignore case, spaces, and punctuation.
 - o Handle edge cases (empty strings, identical words).

Example Assert Test Cases:
`assert is_anagram("listen", "silent") == True`
`assert is_anagram("hello", "world") == False`
`assert is_anagram("Dormitory", "Dirty Room") == True`

Expected Output #3: • Function correctly identifying anagrams and passing all AI-generated tests.

Output Screenshot:

```
lab8.1 > task03.py > ...
1 # Generate at least 6 assert-based test cases for:
2 # def is_anagram(str1, str2):
3 # Requirements:
4 # - Ignore case
5 # - Ignore spaces
6 # - Ignore punctuation
7 # - Handle empty strings
8 # - Handle identical words
9 # Include both positive and negative cases.
10 # Return only assert statements.
11 # Write a clean Python implementation for:
12 # def is_anagram(str1, str2):
13 # Requirements:
14 # - Ignore case differences
15 # - Remove spaces
16 # - Remove punctuation
17 # - Handle empty input safely
18 # - No external libraries
19 # Ensure all test cases pass.
20 def is_anagram(str1, str2):
21     def clean_string(s):
22         return ''.join(char.lower() for char in s if char.isalnum())
23
24     cleaned_str1 = clean_string(str1)
25     cleaned_str2 = clean_string(str2)
26
27     return sorted(cleaned_str1) == sorted(cleaned_str2)
28
29 # Test Cases
30 assert is_anagram("Listen", "Silent") == True
31 assert is_anagram("Triangle", "Integral") == True
32 assert is_anagram("Dormitory", "Dirty Room") == True
33 assert is_anagram("Hello", "World") == False
34 assert is_anagram("", "") == True
35 assert is_anagram("A", "a") == True
36 assert is_anagram("Anagram", "Nag a ram") == True
37 assert is_anagram("Test", "Best") == False
38 print("All test cases passed successfully 🎉")
39
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

/usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab8.1/task03.py
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_lab/lab8.1/task03.py
All test cases passed successfully 🎉
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab %

Explanation: This spots whether two words or phrases are made of the same letters, ignoring case, spaces, and punctuation.

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.

• **Methods:**

- o add_item(name, quantity)
- o remove_item(name, quantity)
- o get_stock(name)

Example Assert Test Cases:

```
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.add_item("Book", 3)
assert inv.get_stock("Book") == 3
```

Expected Output #4: • Fully functional class passing all assertions.

Output Screenshot:

```
lab8.1 > task04.py > ...
1 # Generate at least 6 assert-based test cases for an Inventory class with:
2 # Methods:
3 # - add_item(name, quantity)
4 # - remove_item(name, quantity)
5 # - get_stock(name)
6 # Test:
7 # - Adding new item
8 # - Removing items
9 # - Removing more than stock
10 # - Getting stock of non-existing item
11 # - Adding duplicate items
12 # - Negative quantity handling
13 # Return only assert statements.
14 # Write a clean and robust Python class:
15 # class Inventory:
16 # Include:
17 # - Internal dictionary for storage
18 # - Proper validation
19 # - Error handling for invalid operations
20 # - Prevent negative stock
21 # Ensure it passes all tests.
22 class Inventory:
23     def __init__(self):
24         self.stock = {}
25
26     def add_item(self, name, quantity):
27         if not isinstance(name, str) or not isinstance(quantity, int):
28             raise TypeError("Invalid input type")
29
30         if quantity <= 0:
31             raise ValueError("Quantity must be positive")
32
33         if name in self.stock:
34             self.stock[name] += quantity
35         else:
36             self.stock[name] = quantity
37
38     def remove_item(self, name, quantity):
39         if name not in self.stock:
40             raise KeyError("Item does not exist")
41
42         if quantity <= 0:
43             raise ValueError("Quantity must be positive")
44
```

```
lab8.1 > task04.py > ...
52
53
54 # TestCases :
55 inventory = Inventory()
56 # Add items
57 inventory.add_item("Pen", 10)
58 assert inventory.get_stock("Pen") == 10
59 inventory.add_item("Pen", 5)
60 assert inventory.get_stock("Pen") == 15
61 inventory.add_item("Book", 3)
62 assert inventory.get_stock("Book") == 3
63 # Remove items
64 inventory.remove_item("Pen", 5)
65 assert inventory.get_stock("Pen") == 10
66 # Remove more than available
67 try:
68     inventory.remove_item("Pen", 20)
69 except ValueError as e:
70     assert e.args[0] == "Insufficient stock"
71 # Remove non-existing item
72 try:
73     inventory.remove_item("Banana", 1)
74 except KeyError as e:
75     assert e.args[0] == "Item does not exist"
76 # Negative quantity add
77 try:
78     inventory.add_item("Pencil", -5)
79 except ValueError as e:
80     assert e.args[0] == "Quantity must be positive"
81 # Negative quantity remove
82 try:
83     inventory.remove_item("Pen", -2)
84 except ValueError as e:
85     assert e.args[0] == "Quantity must be positive"
86 print("All Inventory test cases passed successfully 🎉")
87
88
89 # Suggest improvements for this Inventory class in terms of:
90
```

Explanation: This simulates a mini store system where you can add, remove, and check stock for items.

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.

- **Requirements:**

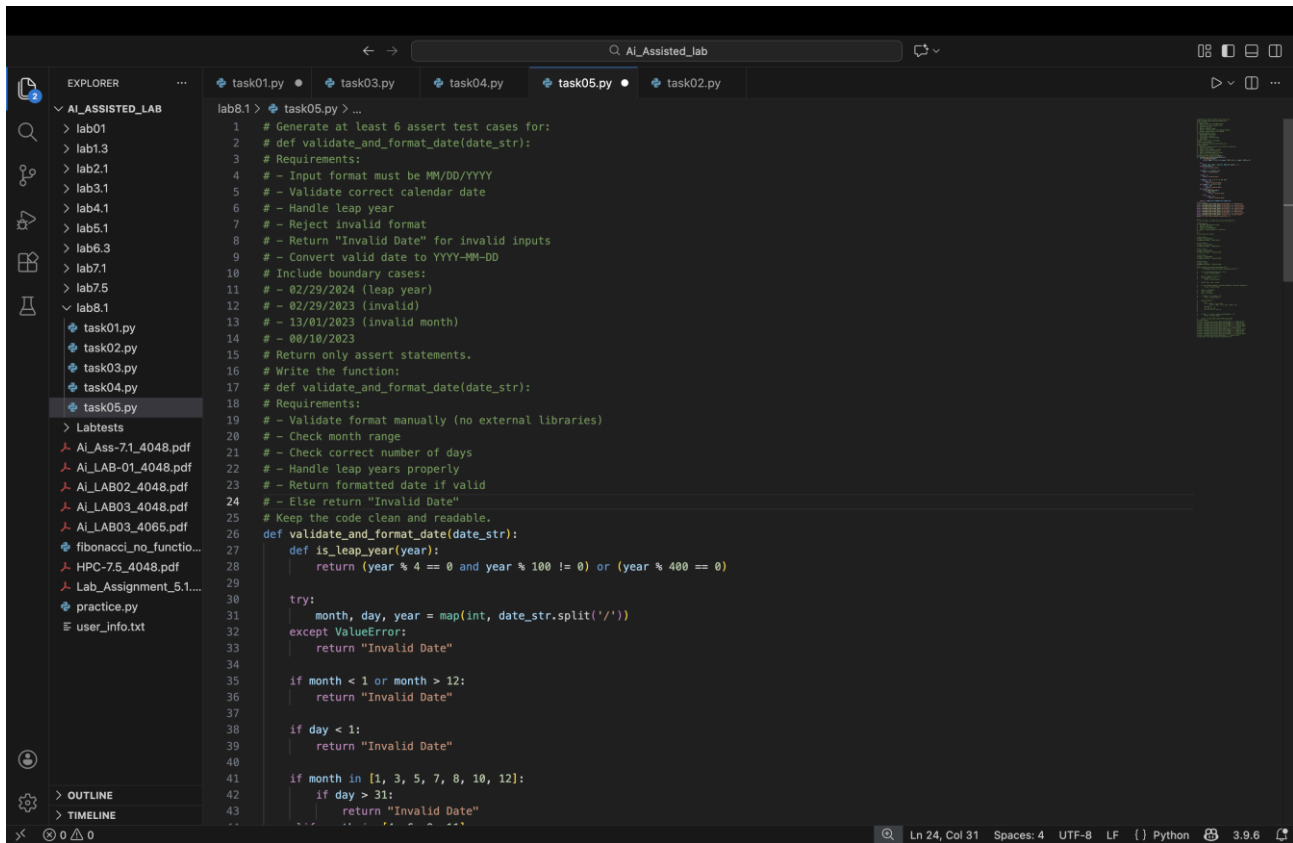
- o Validate "MM/DD/YYYY" format.
- o Handle invalid dates.
- o Convert valid dates to "YYYY-MM-DD".

Example Assert Test Cases:

```
assert validate_and_format_date("10/15/2023") == "2023-10-15"  
assert validate_and_format_date("02/30/2023") == "Invalid Date"  
assert validate_and_format_date("01/01/2024") == "2024-01-01"
```

Expected Output #5: • Function passes all AI-generated assertions and handles edge cases.

Output Screenshot:



```
lab8.1 > task05.py > ...  
1 # Generate at least 6 assert test cases for:  
2 # def validate_and_format_date(date_str):  
3 # Requirements:  
4 # - Input format must be MM/DD/YYYY  
5 # - Validate correct calendar date  
6 # - Handle leap year  
7 # - Reject invalid format  
8 # - Return "Invalid Date" for invalid inputs  
9 # - Convert valid date to YYYY-MM-DD  
10 # Include boundary cases:  
11 # - 02/29/2024 (leap year)  
12 # - 02/29/2023 (invalid)  
13 # - 13/01/2023 (invalid month)  
14 # - 00/10/2023  
15 # Return only assert statements.  
16 # Write the function:  
17 # def validate_and_format_date(date_str):  
18 # Requirements:  
19 # - Validate format manually (no external libraries)  
20 # - Check month range  
21 # - Check correct number of days  
22 # - Handle leap years properly  
23 # - Return formatted date if valid  
24 # - Else return "Invalid Date"  
25 # Keep the code clean and readable.  
26 def validate_and_format_date(date_str):  
27     def is_leap_year(year):  
28         return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)  
29  
30     try:  
31         month, day, year = map(int, date_str.split('/'))  
32     except ValueError:  
33         return "Invalid Date"  
34  
35     if month < 1 or month > 12:  
36         return "Invalid Date"  
37  
38     if day < 1:  
39         return "Invalid Date"  
40  
41     if month in [1, 3, 5, 7, 8, 10, 12]:  
42         if day > 31:  
43             return "Invalid Date"  
44  
45     if month in [4, 6, 9, 11]:  
46         if day > 30:  
47             return "Invalid Date"  
48  
49     if not is_leap_year(year) and month == 2 and day > 28:  
50         return "Invalid Date"  
51  
52     return f"{year}-{month:02d}-{day:02d}"
```

The screenshot displays a VS Code editor window with a Python script named `task05.py` open. The script is located in a workspace named `AI_Assisted_Lab`. The script defines a function `validate_and_format_date` that takes a date string and returns a formatted date string or "Invalid Date". The function uses conditional logic to check the month, day, and year, including a leap year check for February. Below the function, there are several test cases using `assert` statements to verify the function's behavior for various dates, including leap years and invalid dates. The script concludes with a `print` statement indicating that all test cases passed successfully.

```
lab8.1 > task05.py > ...
26 def validate_and_format_date(date_str):
44     elif month in [4, 6, 9, 11]:
45         if day > 30:
46             return "Invalid Date"
47     elif month == 2:
48         if is_leap_year(year):
49             if day > 29:
50                 return "Invalid Date"
51         else:
52             if day > 28:
53                 return "Invalid Date"
54     return f"{year:04d}-{month:02d}-{day:02d}"
55 # Test Cases
56 assert validate_and_format_date("02/29/2024") == "2024-02-29"
57 assert validate_and_format_date("02/29/2023") == "Invalid Date"
58 assert validate_and_format_date("13/01/2023") == "Invalid Date"
59 assert validate_and_format_date("00/10/2023") == "Invalid Date"
60 assert validate_and_format_date("12/31/2023") == "2023-12-31"
61 assert validate_and_format_date("01/01/2023") == "2023-01-01"
62 assert validate_and_format_date("02/30/2024") == "Invalid Date"
63 assert validate_and_format_date("invalid") == "Invalid Date"
64 print("All test cases passed successfully 🎉")
65
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + Python 3.9.6
```

Explanation: This ensures a date is valid in “MM/DD/YYYY” format and neatly converts it into “YYYY-MM-DD”.