AI ASSISTED CODING

LAB ASSIGNMENT 8.1

TEST DRIVEN DEVELOPMENT WITH AI

2303A51165

BATCH – 18

Task 1: Password Strength Validator

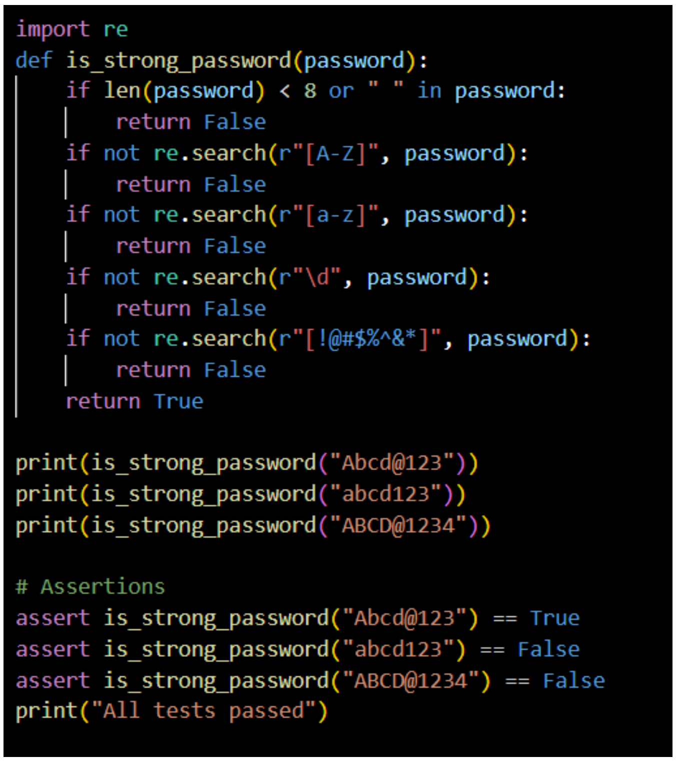
Question

Generate assert test cases and implement password validator.

PROMPT

Implement function ensuring password meets security rules.

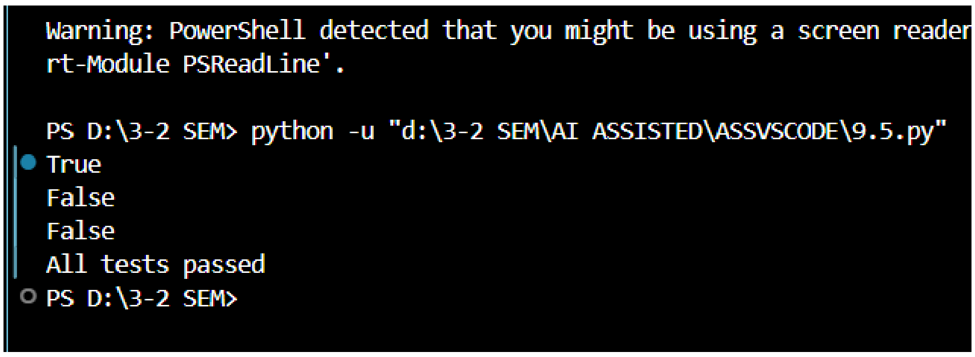
CODE



```python
import re
def is_strong_password(password):
    if len(password) < 8 or " " 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"\d", password):
        return False
    if not re.search(r"[!@#$%^&*]", password):
        return False
    return True

print(is_strong_password("Abcd@123"))
print(is_strong_password("abcd123"))
print(is_strong_password("ABCD@1234"))

# Assertions
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == False
print("All tests passed")
```

```
import re
def is_strong_password(password):
    if len(password) < 8 or " " 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"\d", password):
        return False
    if not re.search(r"[!@#$%^&*]", password):
        return False
    return True
print(is_strong_password("Abcd@123"))
print(is_strong_password("abcd123"))
print(is_strong_password("ABCD@1234"))
# Assertions
assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == False
print("All tests passed")
```

OUTPUT



```
Warning: PowerShell detected that you might be using a screen reader
rt-Module PSReadLine'.

PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED\ASSVSCODE\9.5.py"
True
False
False
All tests passed
PS D:\3-2 SEM>
```

EXPLANATION

Regular expressions validate each rule ensuring secure password.

TASK 2: NUMBER CLASSIFICATION

QUESTION

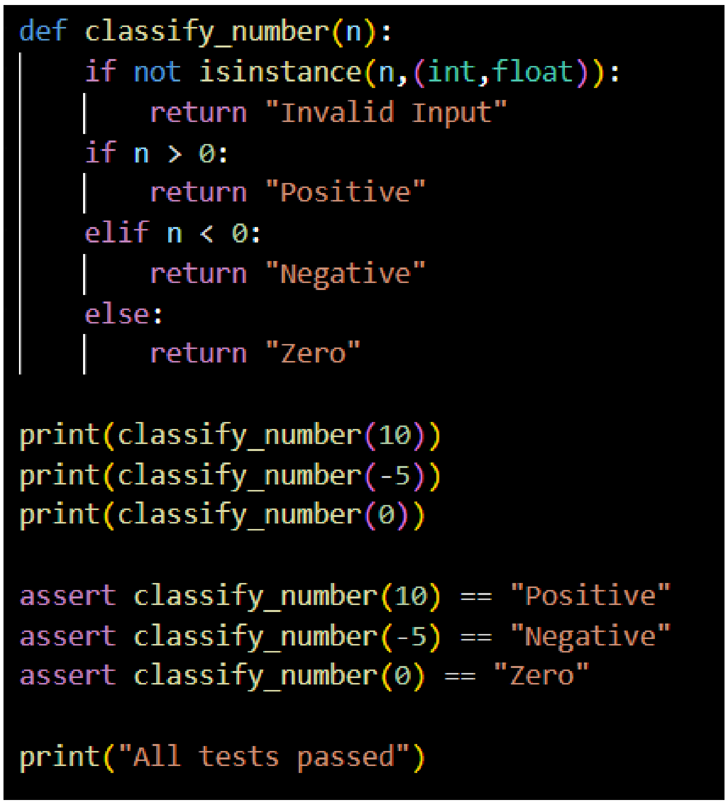
Classify numbers as Positive, Negative, or Zero.

PROMPT

Handle invalid inputs and edge cases.

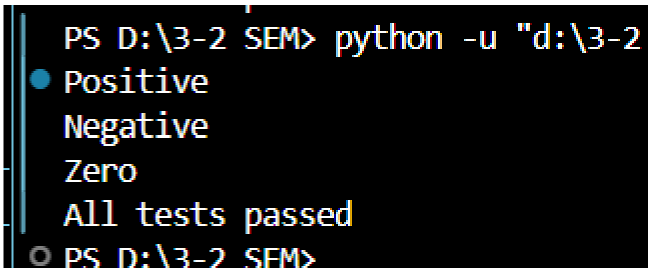
CODE

```
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"
print(classify_number(10))
print(classify_number(-5))
print(classify_number(0))
assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
print("All tests passed")
```



```python
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"

print(classify_number(10))
print(classify_number(-5))
print(classify_number(0))

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

print("All tests passed")
```

OUTPUT



```
PS D:\3-2 SEM> python -u "d:\3-2
Positive
Negative
Zero
All tests passed
PS D:\3-2 SEM>
```

EXPLANATION

Function checks type then classifies value.

TASK 3: ANAGRAM CHECKER

QUESTION

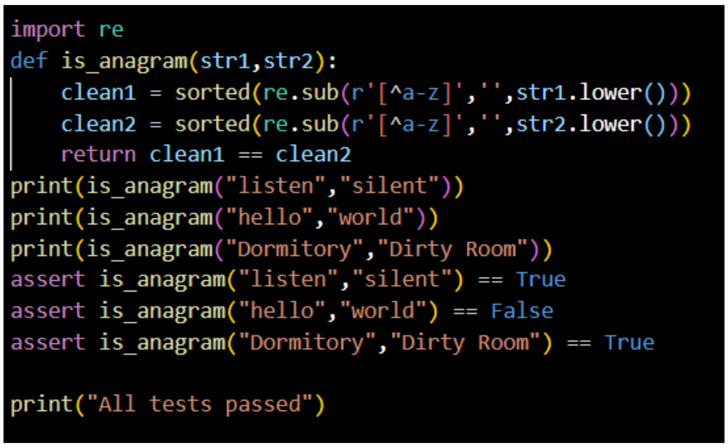
Implement anagram checker ignoring spaces and case.

PROMPT

Generate tests and implement logic.

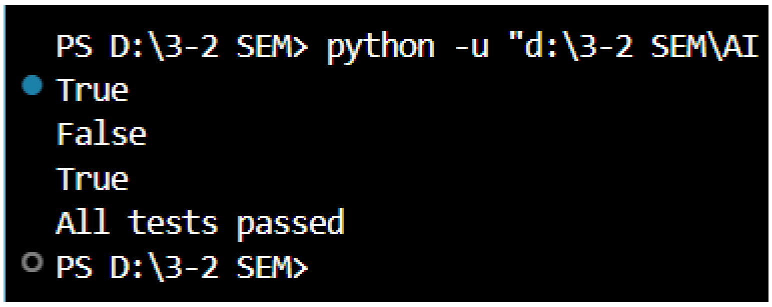
CODE

```
import re
def is_anagram(str1,str2):
    clean1 = sorted(re.sub(r'[^a-z]','',str1.lower()))
    clean2 = sorted(re.sub(r'[^a-z]','',str2.lower()))
    return clean1 == clean2
print(is_anagram("listen","silent"))
print(is_anagram("hello","world"))
print(is_anagram("Dormitory","Dirty Room"))
assert is_anagram("listen","silent") == True
assert is_anagram("hello","world") == False
assert is_anagram("Dormitory","Dirty Room") == True
print("All tests passed")
```



```
import re
def is_anagram(str1,str2):
    clean1 = sorted(re.sub(r'[^a-z]','',str1.lower()))
    clean2 = sorted(re.sub(r'[^a-z]','',str2.lower()))
    return clean1 == clean2
print(is_anagram("listen","silent"))
print(is_anagram("hello","world"))
print(is_anagram("Dormitory","Dirty Room"))
assert is_anagram("listen","silent") == True
assert is_anagram("hello","world") == False
assert is_anagram("Dormitory","Dirty Room") == True

print("All tests passed")
```

OUTPUT



```
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI
True
False
True
All tests passed
PS D:\3-2 SEM>
```

EXPLANATION

Cleaning removes spaces and punctuation before comparison.

Task 4:INVENTORY CLASS

QUESTION

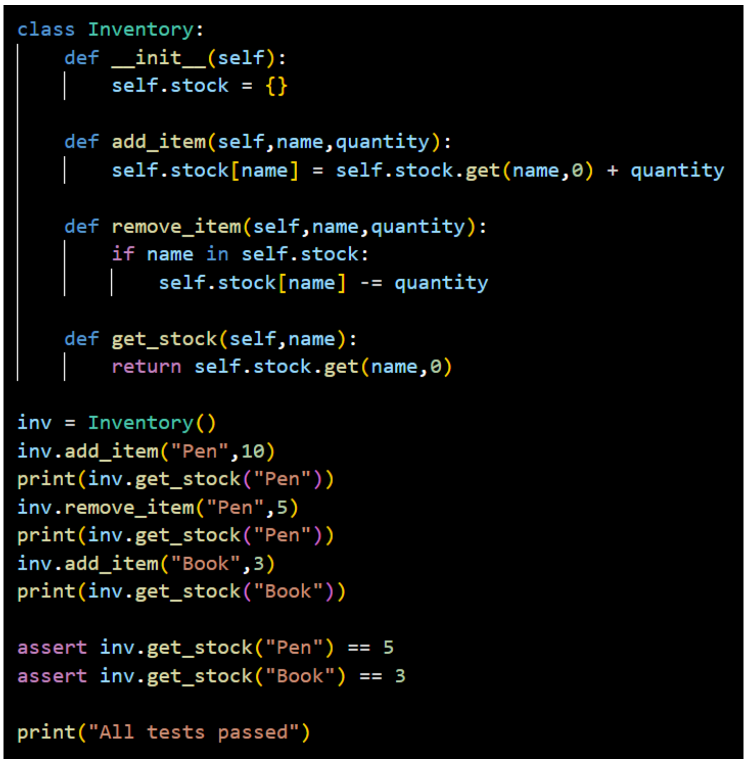
Implement inventory system with test cases.

PROMPT

Create class managing stock.

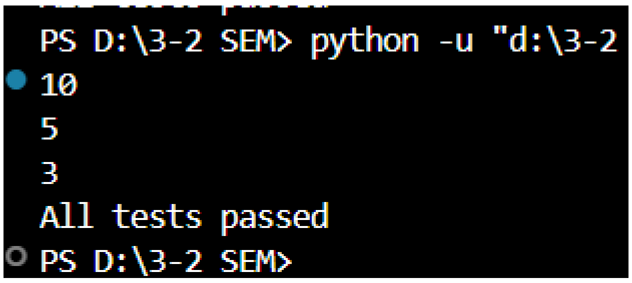
Code

```
class Inventory:
   def __init__(self):
      self.stock = {}
   def add_item(self,name,quantity):
      self.stock[name] = self.stock.get(name,0) + quantity
   def remove_item(self,name,quantity):
      if name in self.stock:
         self.stock[name] -= quantity
   def get_stock(self,name):
      return self.stock.get(name,0)
inv = Inventory()
inv.add_item("Pen",10)
print(inv.get_stock("Pen"))
inv.remove_item("Pen",5)
print(inv.get_stock("Pen"))
inv.add_item("Book",3)
print(inv.get_stock("Book"))
assert inv.get_stock("Pen") == 5
assert inv.get_stock("Book") == 3
print("All tests passed")
```



```python
class Inventory:
    def __init__(self):
        self.stock = {}

    def add_item(self,name,quantity):
        self.stock[name] = self.stock.get(name,0) + quantity

    def remove_item(self,name,quantity):
        if name in self.stock:
            self.stock[name] -= quantity

    def get_stock(self,name):
        return self.stock.get(name,0)

inv = Inventory()
inv.add_item("Pen",10)
print(inv.get_stock("Pen"))
inv.remove_item("Pen",5)
print(inv.get_stock("Pen"))
inv.add_item("Book",3)
print(inv.get_stock("Book"))

assert inv.get_stock("Pen") == 5
assert inv.get_stock("Book") == 3

print("All tests passed")
```

OUTPUT



```
PS D:\3-2 SEM> python -u "d:\3-2
10
5
3
All tests passed
PS D:\3-2 SEM>
```

Assertions passed

EXPLANATION

Class uses dictionary to manage stock quantities.

Task 5: DATE VALIDATION & FORMATTING

QUESTION

Validate date format and convert to YYYY-MM-DD.

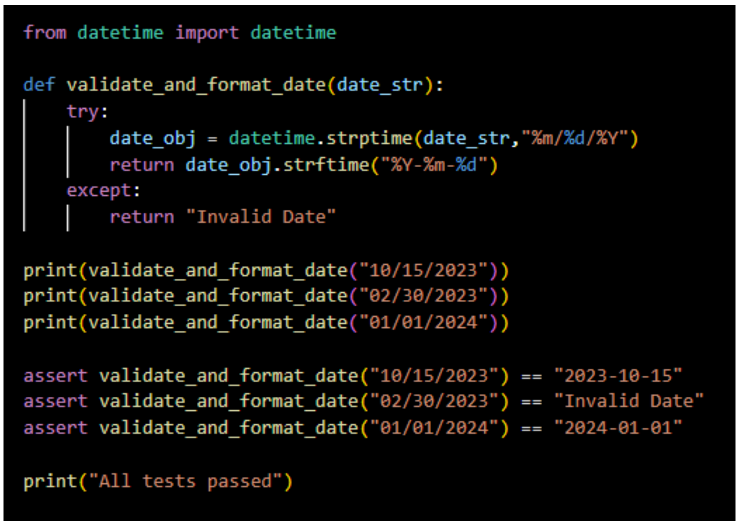
PROMPT

Handle invalid dates and format conversion.

Code

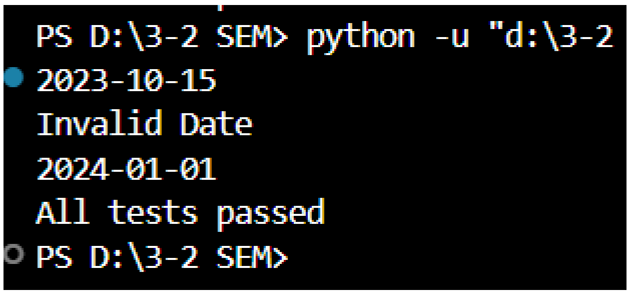
from datetime import datetime

```
def validate_and_format_date(date_str):
    try:
        date_obj = datetime.strptime(date_str,"%m/%d/%Y")
        return date_obj.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"
print(validate_and_format_date("10/15/2023"))
print(validate_and_format_date("02/30/2023"))
print(validate_and_format_date("01/01/2024"))
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"
print("All tests passed")
```



```
from datetime import datetime

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

print(validate_and_format_date("10/15/2023"))
print(validate_and_format_date("02/30/2023"))
print(validate_and_format_date("01/01/2024"))

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"

print("All tests passed")
```

OUTPUT



```
PS D:\3-2 SEM> python -u "d:\3-2
2023-10-15
Invalid Date
2024-01-01
All tests passed
PS D:\3-2 SEM>
```

Assertions passed

EXPLANATION

Datetime module validates date and formats correctly.