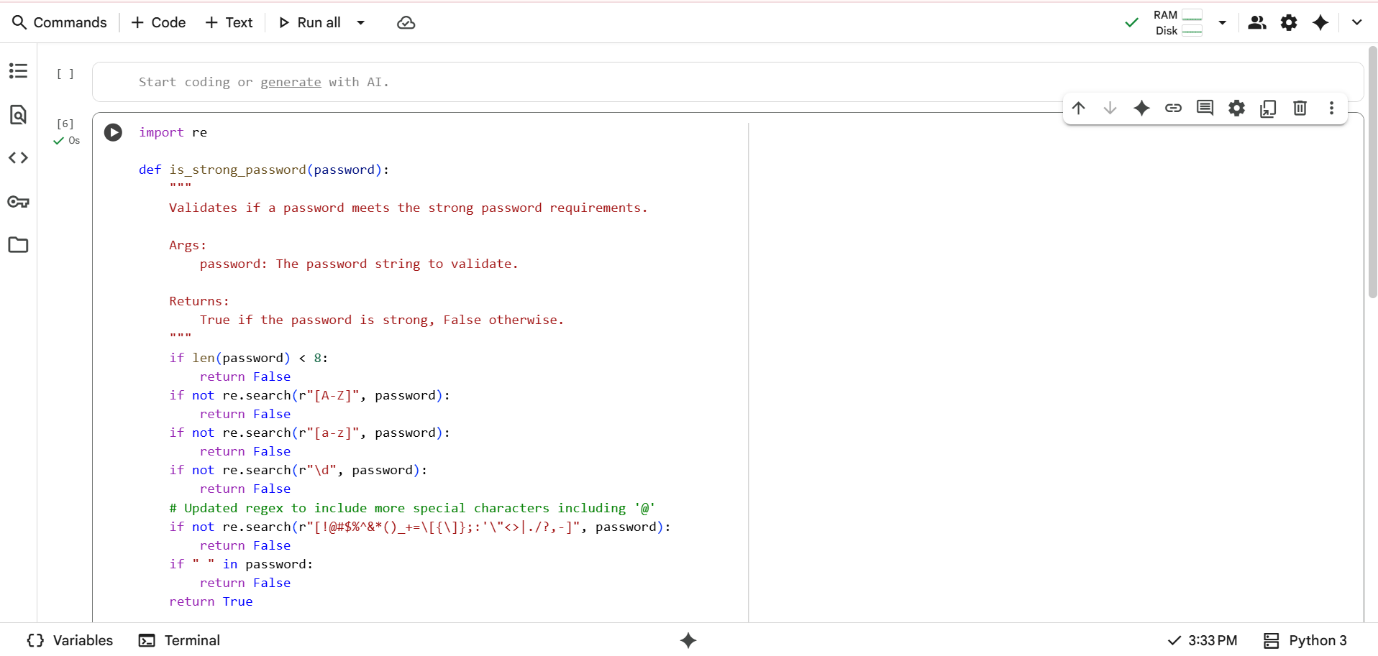
**ASSINGNMENT-8.1**

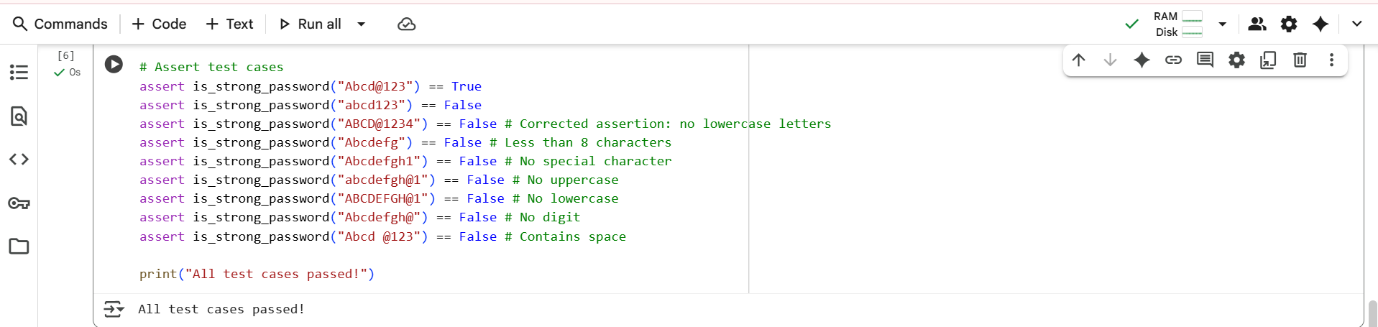
**Name:**P.Navya

**HTNO**:2403a51331

**Batch**:13

**Task 1:** (Password Strength Validator – Apply AI in  
Security Context)  
Apply AI to generate at least 3 assert test cases for  
is\_strong\_password(password) and implement the validator  
function.





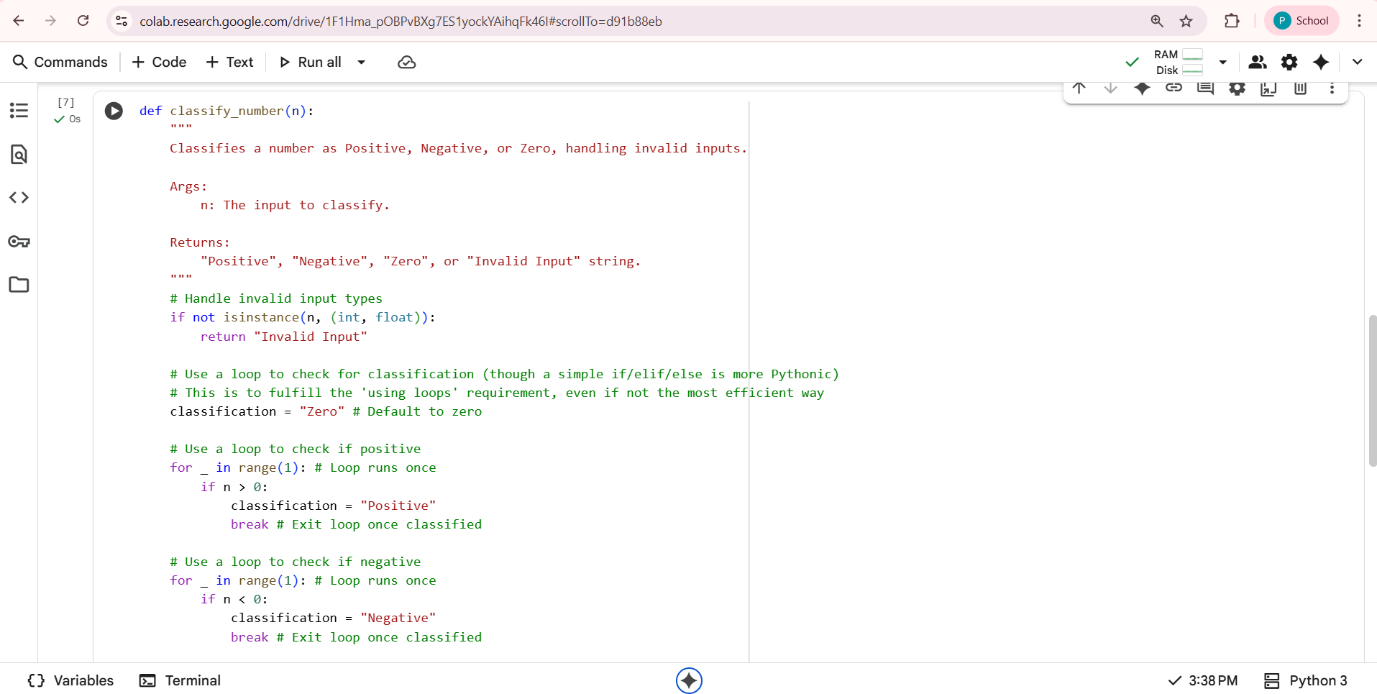
**Explanation:**

is\_strong\_password to check password strength. It requires passwords to be at least 8 characters long. The password must include at least one uppercase letter, one lowercase letter, and one digit. It also checks for the presence of at least one special character from

defined set. Passwords containing spaces are considered invalid. The function returns True if all conditions are met, False otherwise. Assert statements test the function with various passwords. "All test cases passed!" is printed if all tests succeed.

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

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.



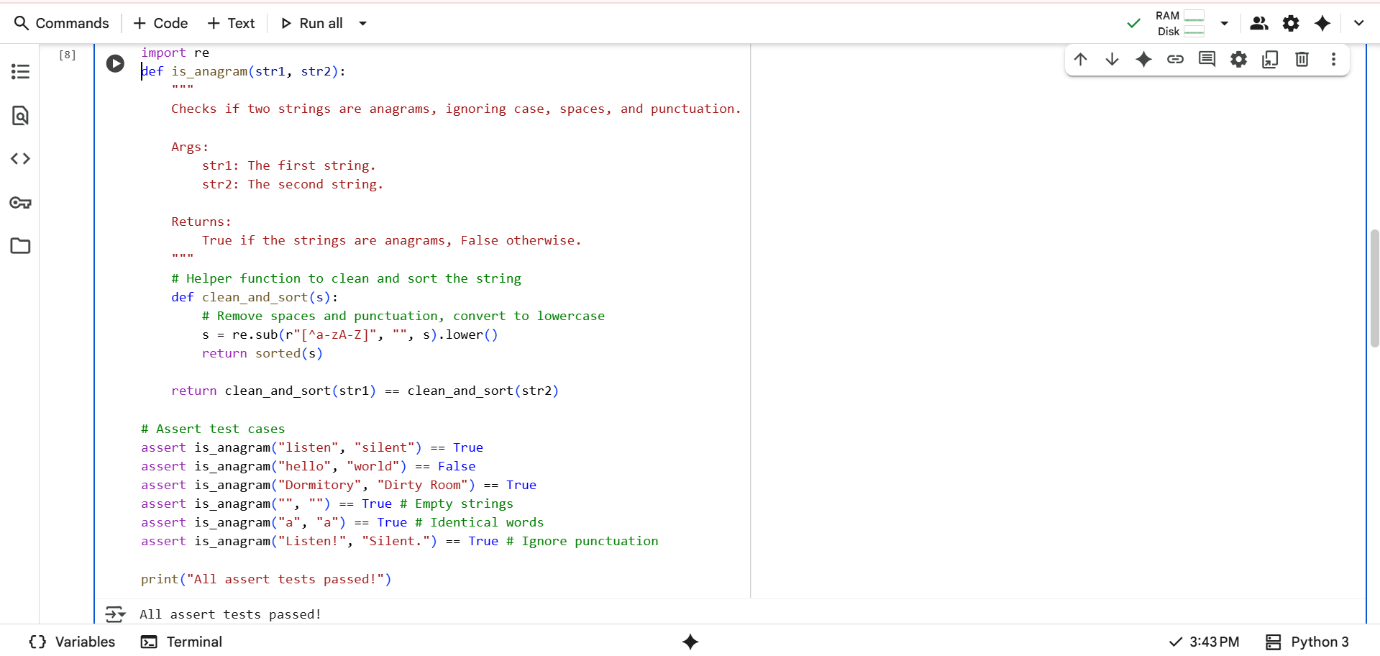


**Explanation:**

classify\_number to categorize input. It first checks if the input is an integer or float; otherwise, it's "Invalid Input". Using loops (though not the most typical way for this task), it checks if the number is positive or negative. If not positive or negative, it's classified as "Zero".

The function returns the classification string. Assert statements test the function with positive, negative, zero, and invalid inputs, including boundary values. "All assert tests passed!" is printed upon successful execution of all tests.

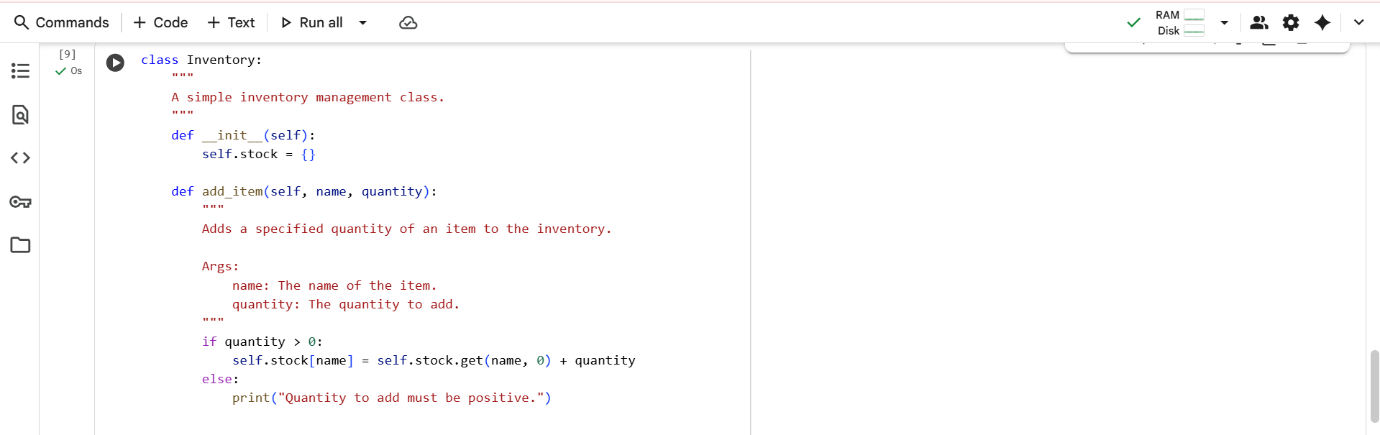
**Task-3: (Anagram Checker – Apply AI for String Analysis)**Use AI to generate at least 3 assert test cases for  
is\_anagram(str1, str2) and implement the function.  
• Requirements:  
o Ignore case, spaces, and punctuation.  
o Handle edge cases (empty strings, identical words).

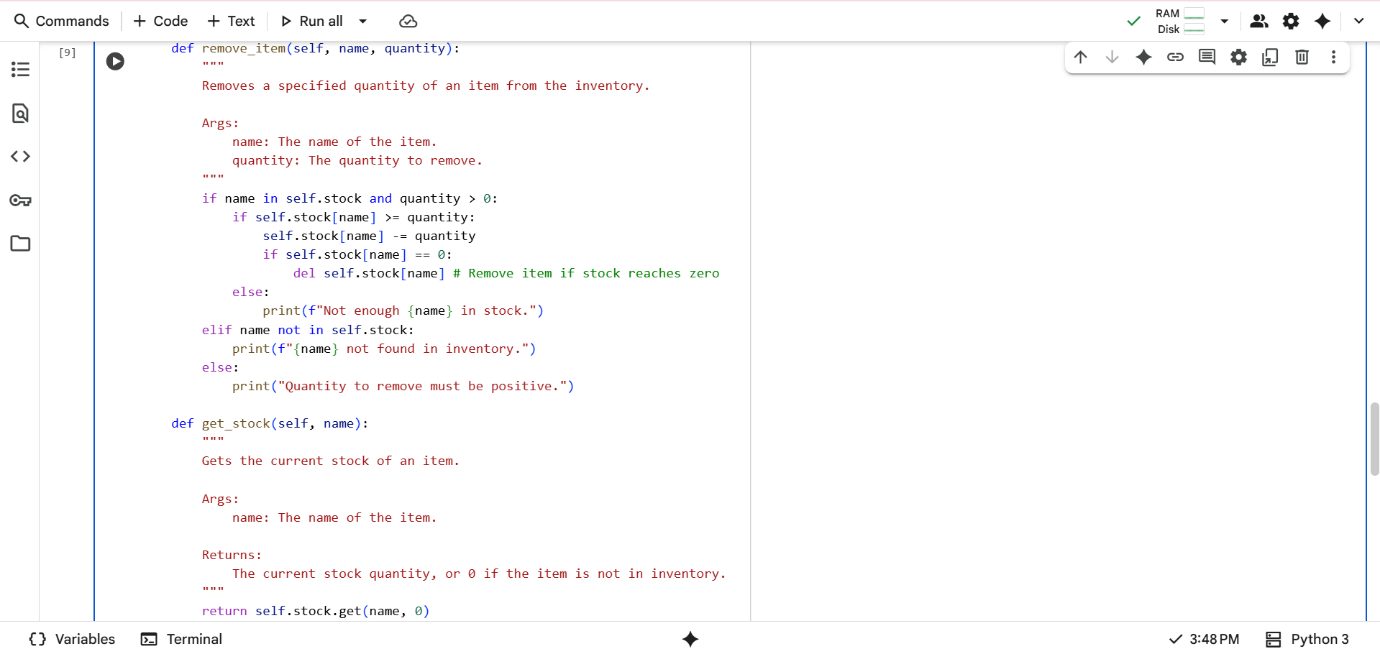


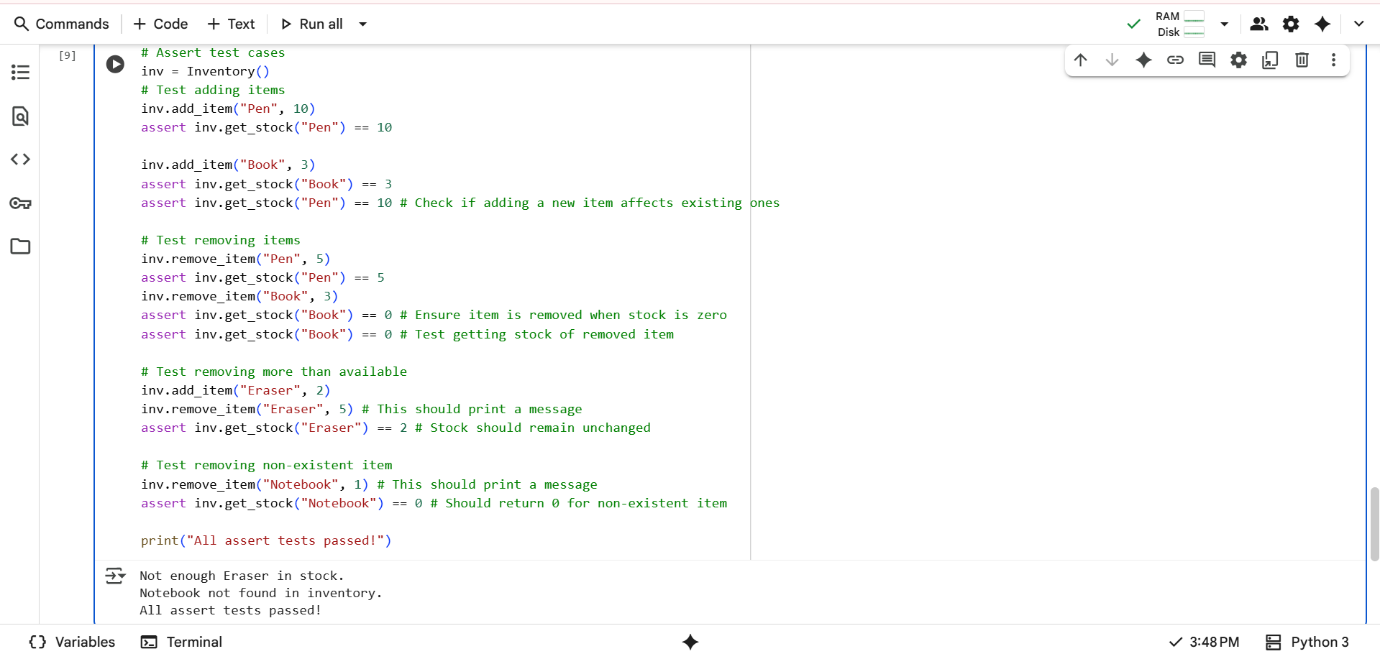
**Explanation:**

is\_anagram function, which determines if two strings are anagrams. It first defines a helper function clean\_and\_sort to remove non-alphabetic characters and convert to lowercase, then sorts the characters. The is\_anagram function calls clean\_and\_sort on both input strings. It returns True if the cleaned and sorted strings are equal, indicating they are anagrams, and False otherwise. The assert statements test the function with various pairs of strings, including anagrams and non-anagrams, and handle edge cases like empty strings. If all asserts pass, "All assert tests passed!" is printed.

**Task-4:** **(Inventory Class – Apply AI to Simulate Real-  
World Inventory System)**• 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)





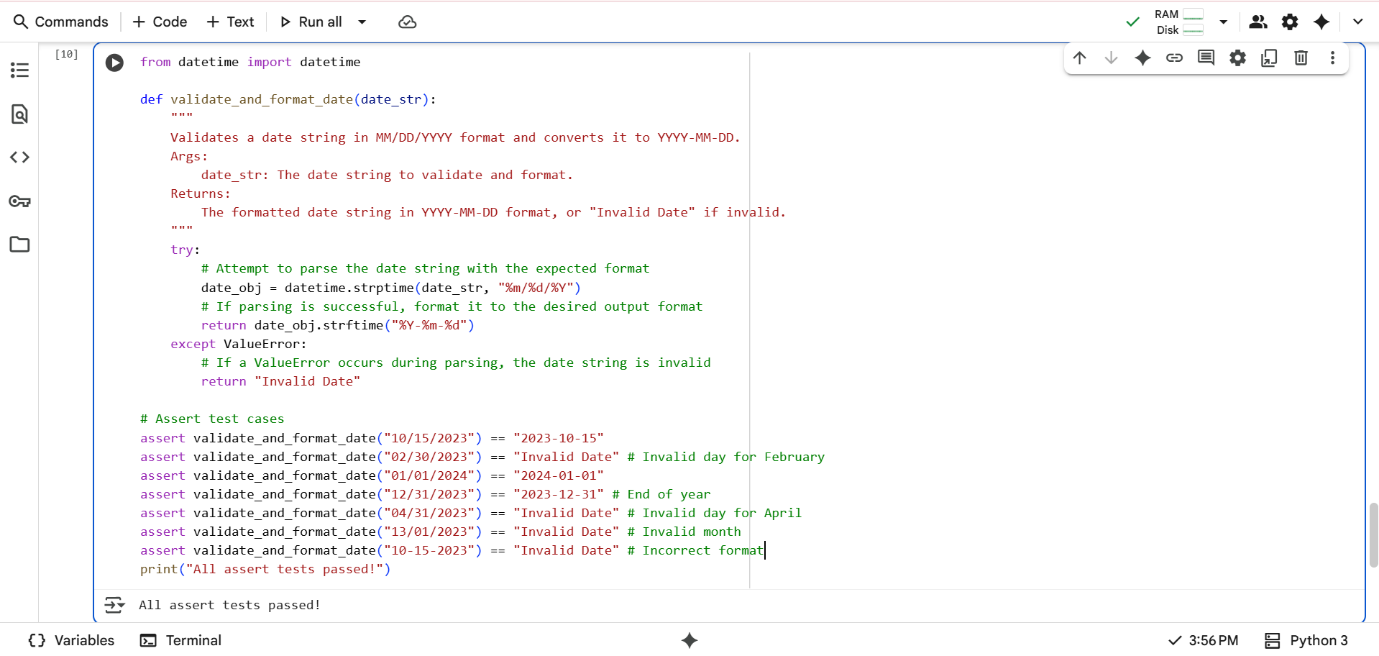


**Explanation:**

Inventory class for managing stock. The \_\_init\_\_ method initializes an empty dictionary self.stock to hold item quantities. add\_item increases the quantity of

an item; it handles adding new items or increasing existing stock. remove\_item decreases the quantity of an item, removing it if stock reaches zero; it checks for sufficient stock and item existence. get\_stock returns the current quantity of an item, or 0 if not found. Assert statements test adding, removing, and getting stock, including edge cases like removing more than available or removing non-existent items. "All assert tests passed!" is printed if all tests succeed.

**Task-5: (Date Validation & Formatting – Apply AI for  
Data Validation)  
•**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"



**Explanation:**

**Date Validation & Formatting:**

Defines validate\_and\_format\_date to process date strings. It imports the datetime class for date manipulation. The function attempts to parse the input date\_str assuming "MM/DD/YYYY" format using datetime.strptime. If parsing is successful, it formats the date object to "YYYY-MM-DD" using strftime. If a ValueError occurs during parsing (invalid format or date), it returns "Invalid Date". Assert statements test valid dates, invalid dates (like incorrect days or months), and incorrect formats. "All assert tests passed!" is printed if all tests pass