***AI ASSISTED CODING***

***NAME: Bhawwana shre***

***HT NO: 2403a52311***

***BATCH NO: 01***

***ASSIGMENT 8.2***

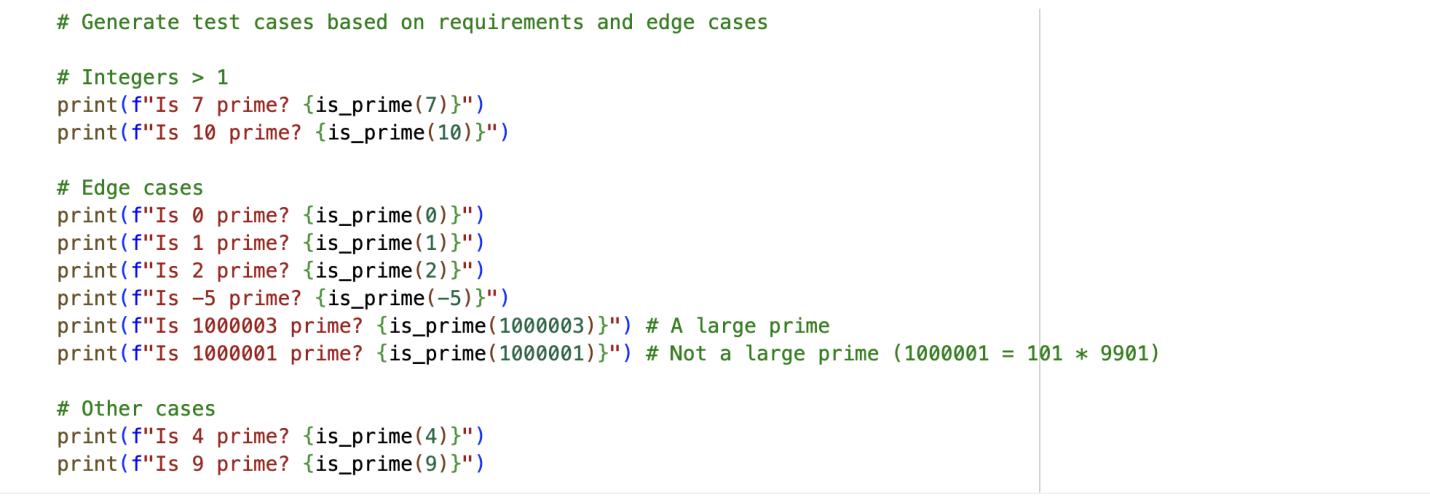
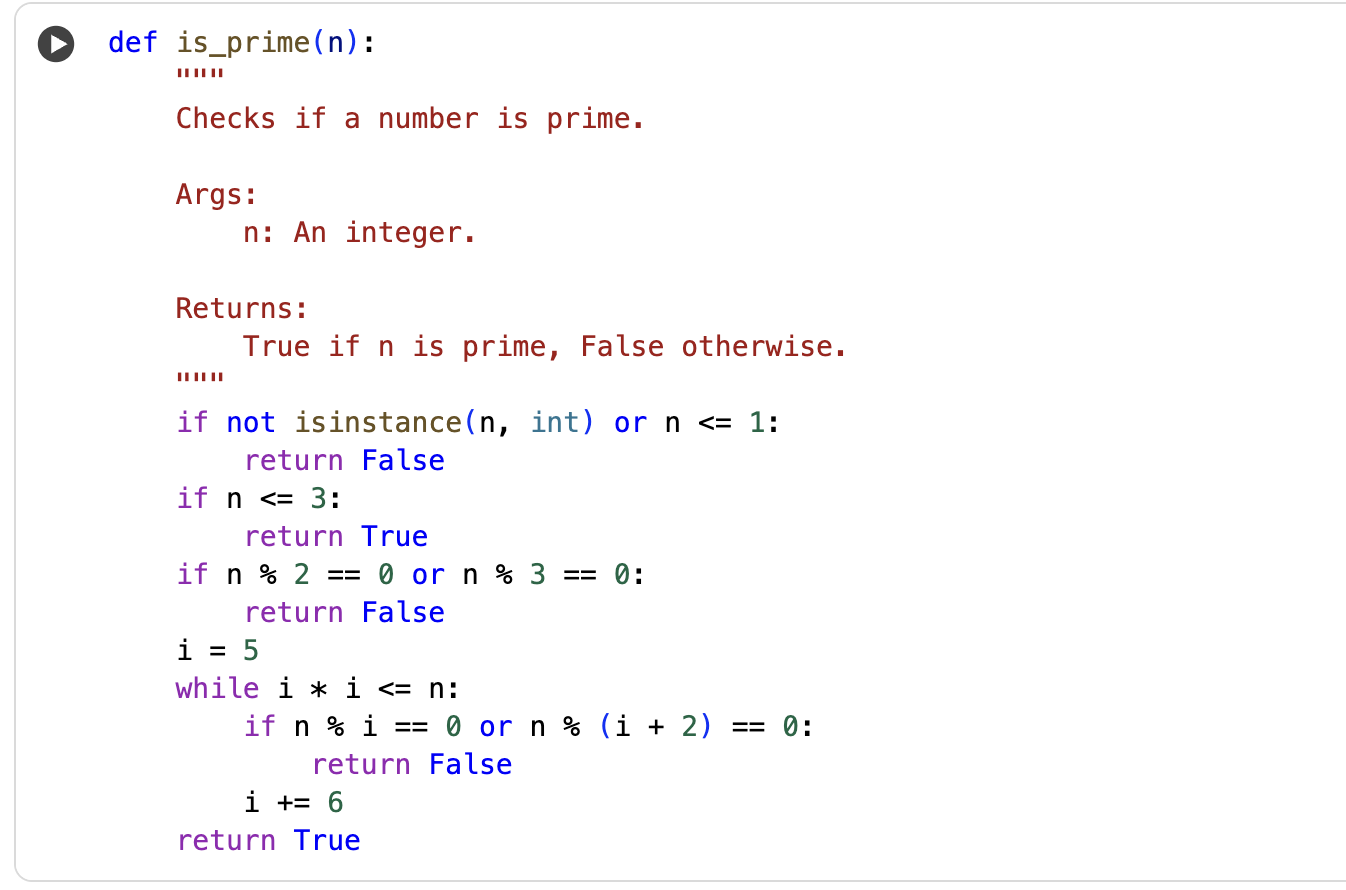
***TASK 1~***

*Use AI to generate test cases for a function is\_prime(n) and then implement the  
function.  
Requirements:  
• Only integers > 1 can be prime.  
• Check edge cases: 0, 1, 2, negative numbers, and large primes.*

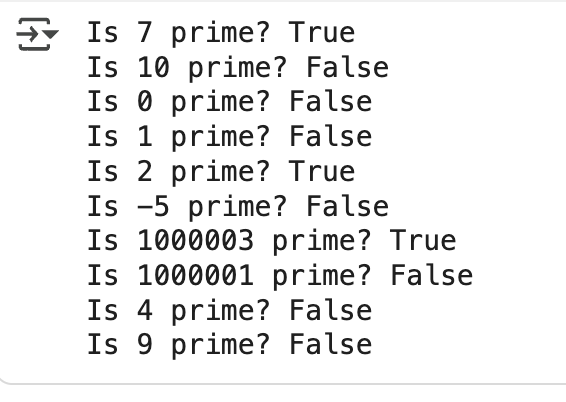
*PROMPT:*

*Generate test cases for a function is\_prime(n) and then implement the function. The function should return True if the input integer n is prime and False otherwise. Only integers greater than 1 can be prime. Include test cases for edge cases: 0, 1, 2, negative numbers, and large primes.*

*CODE:*

**

*OUTPUT:*

**

*EXPLANATION:*

*The first code cell defines the is\_prime function, which efficiently checks if an integer is a prime number. The second code cell provides various test cases to verify the is\_prime function's correctness, including edge cases like 0, 1, and negative numbers, as well as large prime and non-prime numbers.*

***TASK 2~***

*Ask AI to generate test cases for celsius\_to\_fahrenheit(c) and fahrenheit\_to\_celsius(f).  
Requirements  
• Validate known pairs: 0°C = 32°F, 100°C = 212°F.  
• Include decimals and invalid inputs like strings or None*

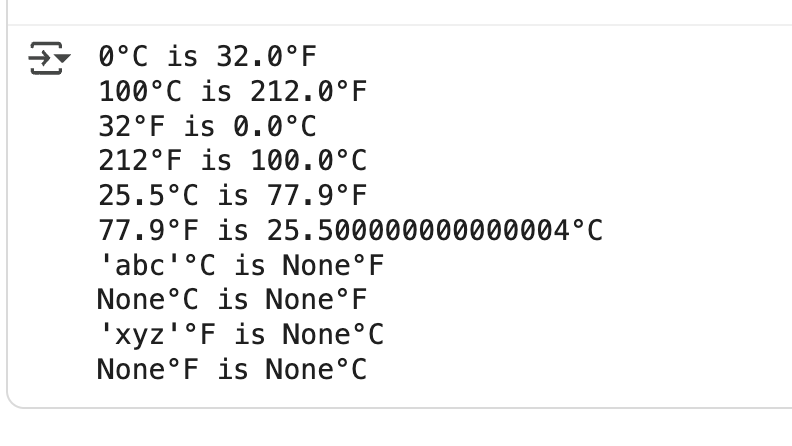
*PROMPT:*

*Generate test cases and implement functions for converting between Celsius and Fahrenheit. Include test cases for known pairs (0°C = 32°F, 100°C = 212°F), decimals, and invalid inputs (strings, None).*

*CODE:*

**

*OUTPUT:*

**

*EXPLANATION:*

*The first code cell defines two functions:*

*celsius\_to\_fahrenheit and fahrenheit\_to\_celsius for converting temperatures, including handling invalid inputs. The second code cell contains test cases for these functions, verifying known conversions, decimal values, and how invalid inputs are handled.*

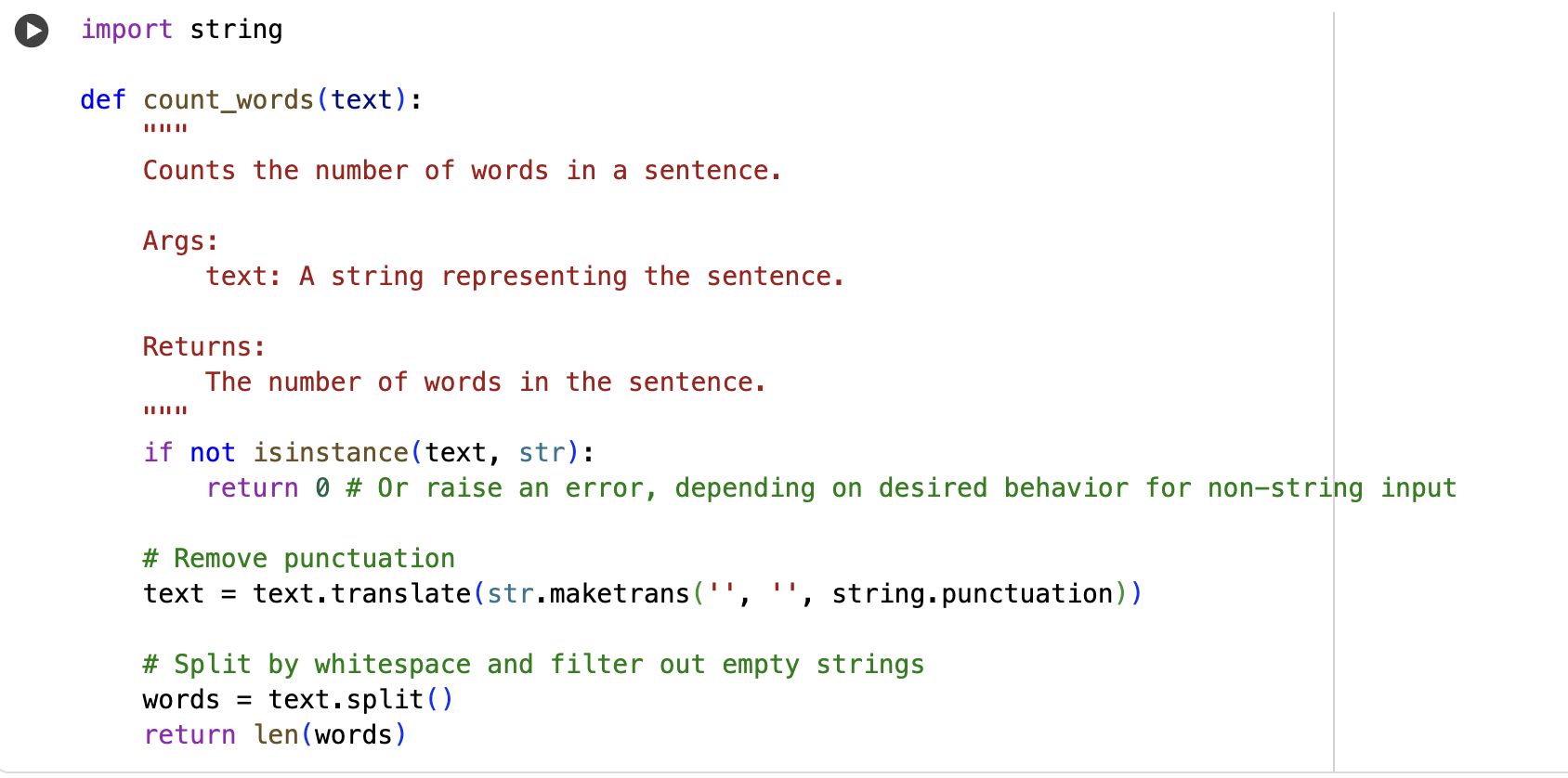
***TASK 3~***

*Use AI to write test cases for a function count\_words(text) that returns the number of words in a sentence.  
Requirement  
Handle normal text, multiple spaces, punctuation, and empty strings*

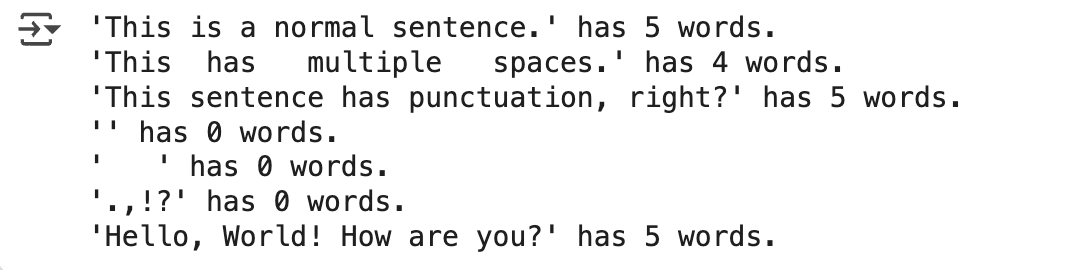
*PROMPT:*

*Generate test cases for a function count\_words(text) that returns the number of words in a sentence, handling normal text, multiple spaces, punctuation, and empty strings. Then, implement the function and test it with the generated test cases.*

*CODE:*

**

*OUTPUT:*

**

*EXPLANATION:*

*This code cell provides test cases for the count\_words function. It calls the function with different types of strings, including:*

* *A normal sentence.*
* *A sentence with multiple spaces between words.*
* *A sentence with punctuation.*
* *An empty string.*
* *A string containing only spaces.*
* *A string containing only punctuation.*
* *A string with mixed case and punctuation.*

*For each test case, it prints the input string and the number of words returned by the count\_words function, allowing you to verify that the function handles these different scenarios correctly.*

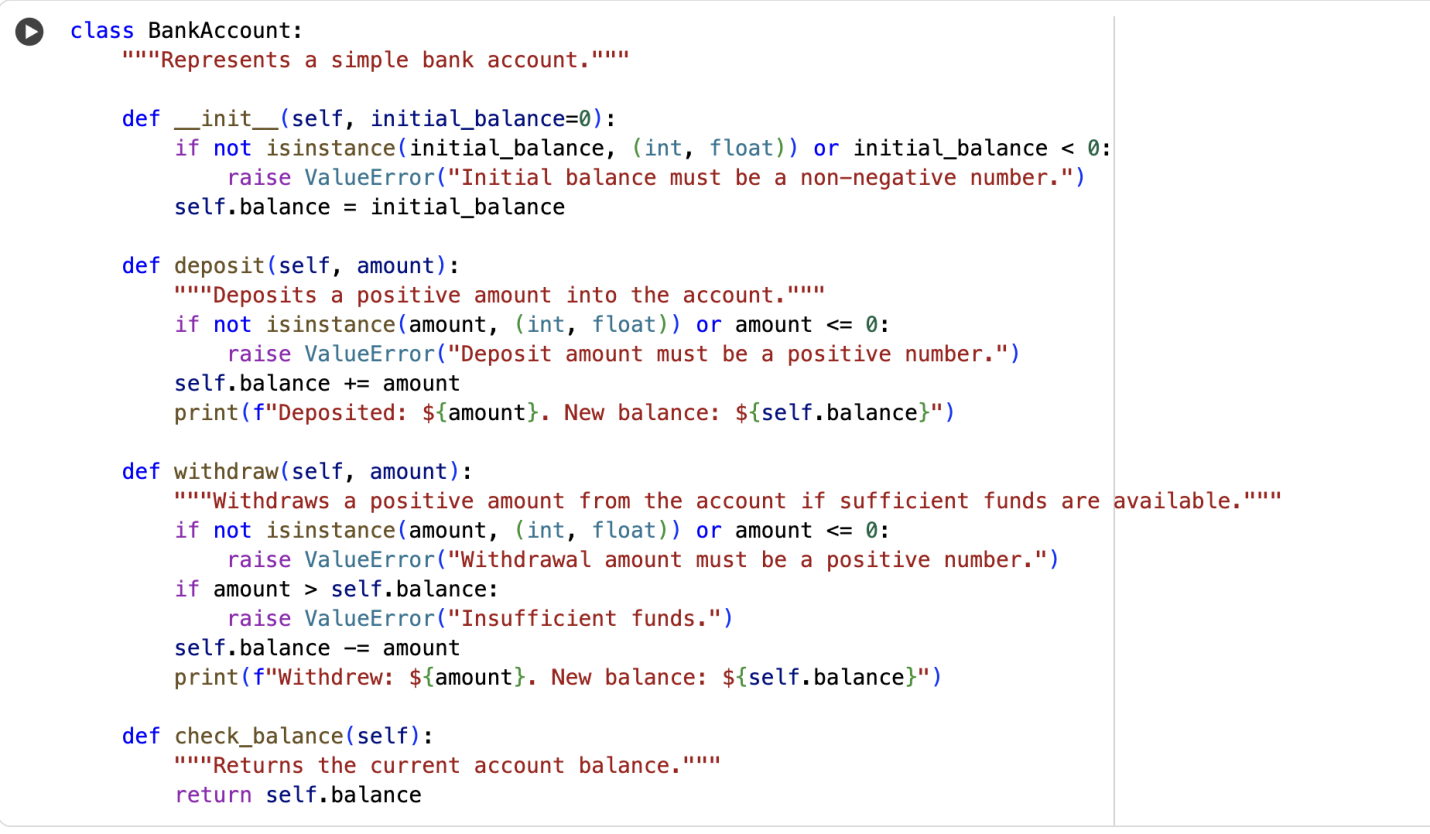
***TASK 4~***

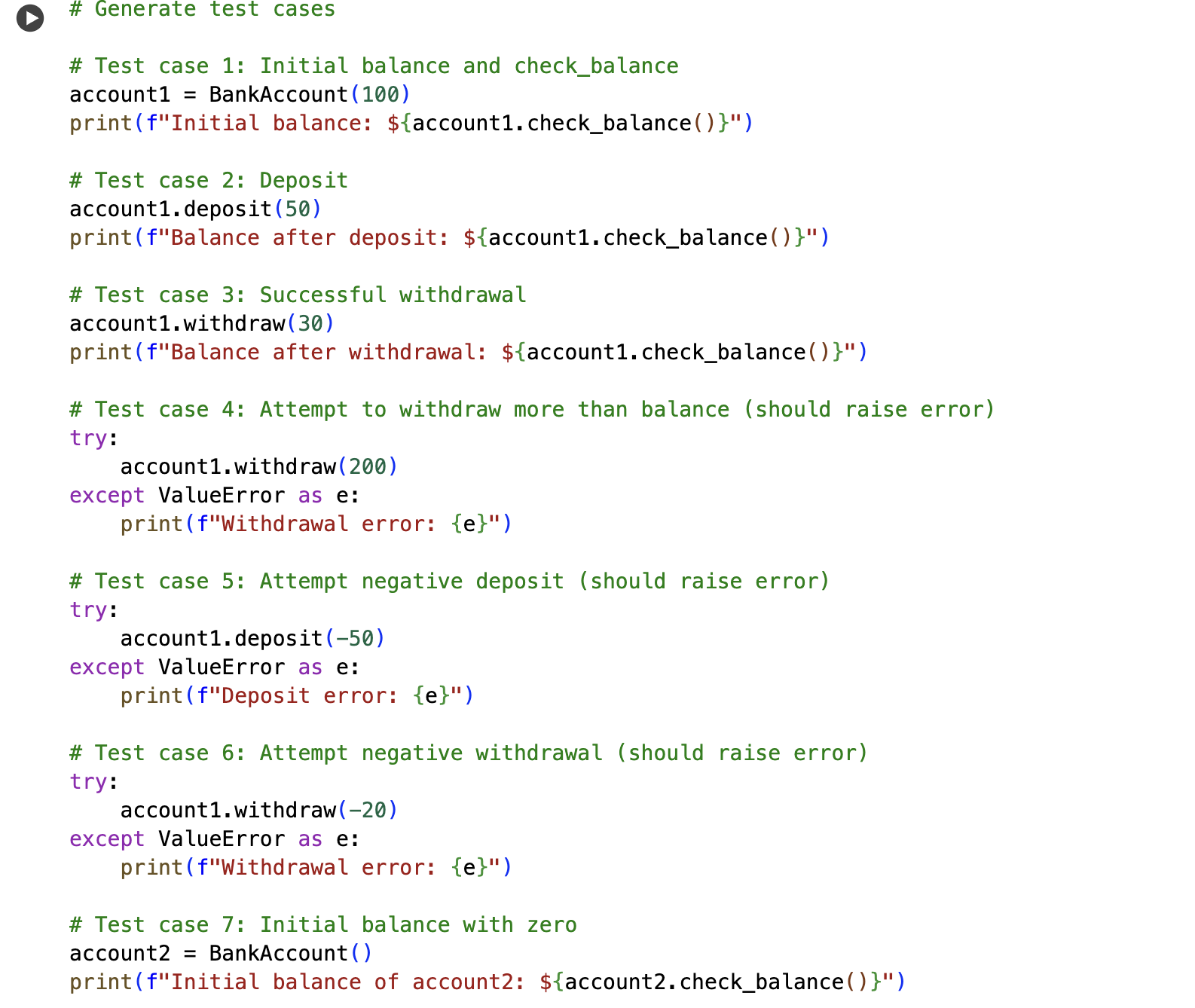
*Generate test cases for a BankAccount class with:  
Methods:  
deposit(amount)  
withdraw(amount)  
check\_balance()  
Requirements:  
• Negative deposits/withdrawals should raise an error.  
• Cannot withdraw more than balance*

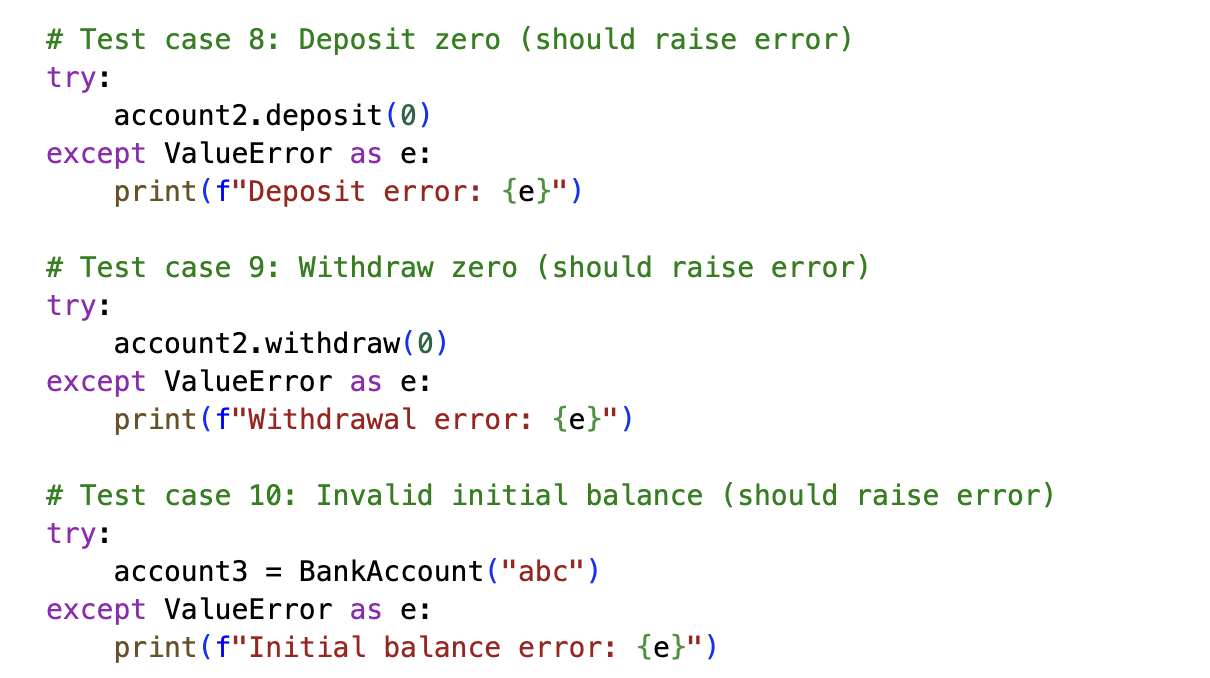
*PROMPT:*

*Generate test cases and implement a BankAccount class with deposit, withdraw, and check\_balance methods, ensuring negative deposits/withdrawals raise errors and withdrawals do not exceed the balance.*

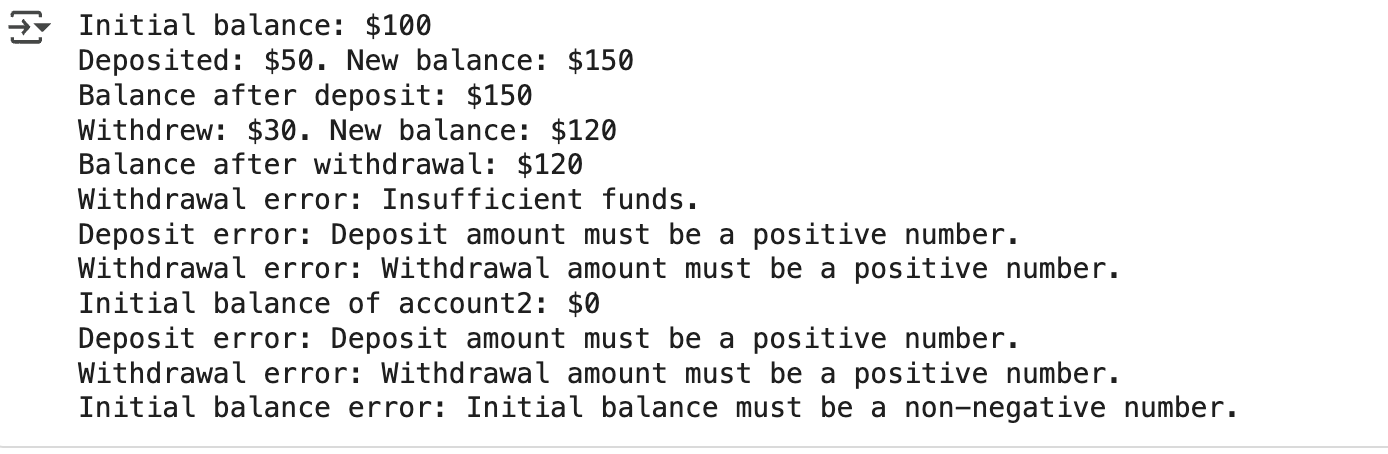
*CODE:*

**

**

**

*OUTPUT:*

**

*EXPLANATION:*

*This code cell contains various test cases for the BankAccount class. It demonstrates how to:*

* *Create a BankAccount instance with an initial balance and check the balance.*
* *Deposit a positive amount.*
* *Withdraw a positive amount successfully.*
* *Handle errors for attempting to withdraw more than the balance, depositing a negative amount, withdrawing a negative amount, and attempting to deposit or withdraw zero.*
* *Handle errors for providing an invalid initial balance.*

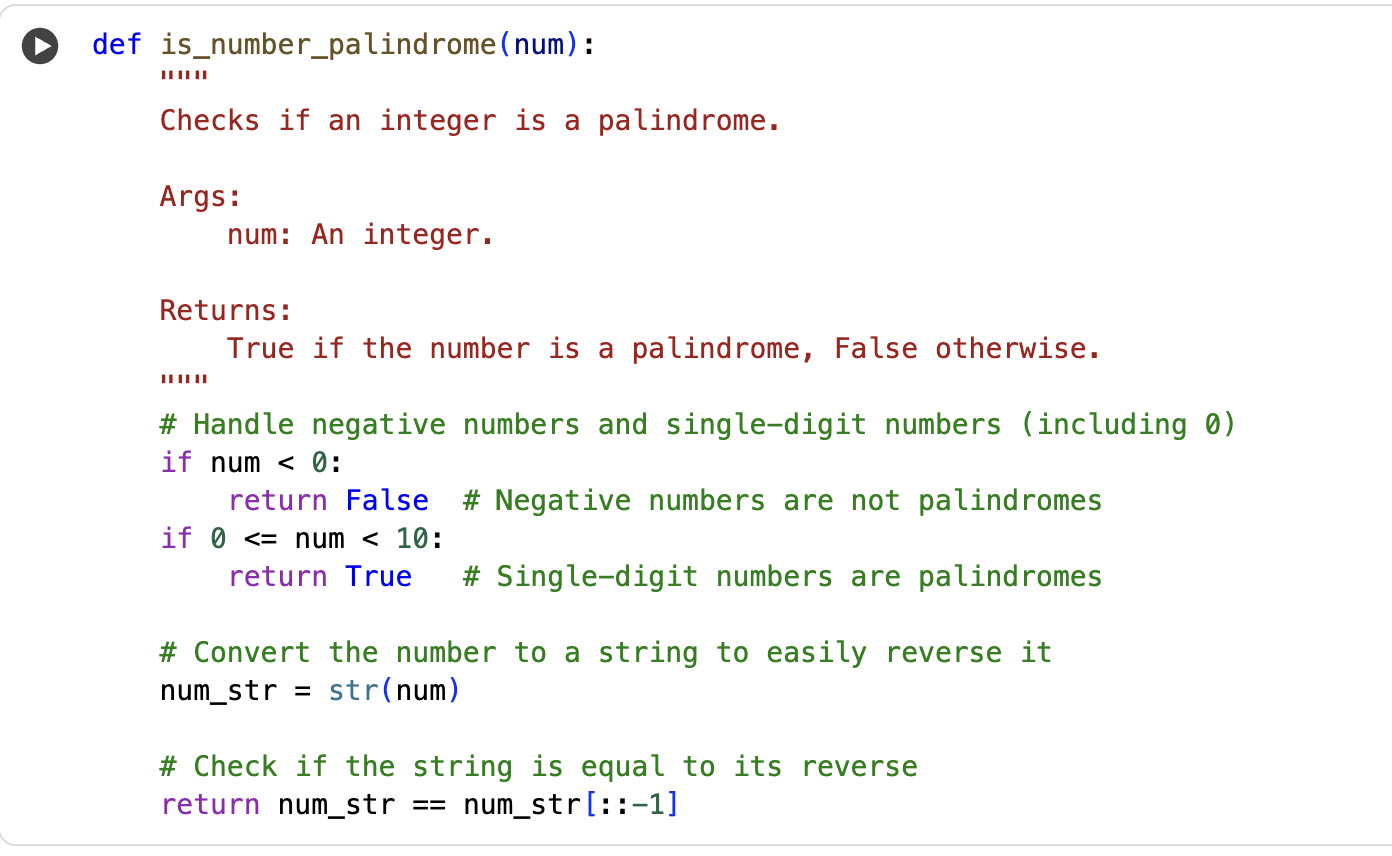
*These test cases help verify that the BankAccount class behaves as expected under different scenarios, including error conditions.*

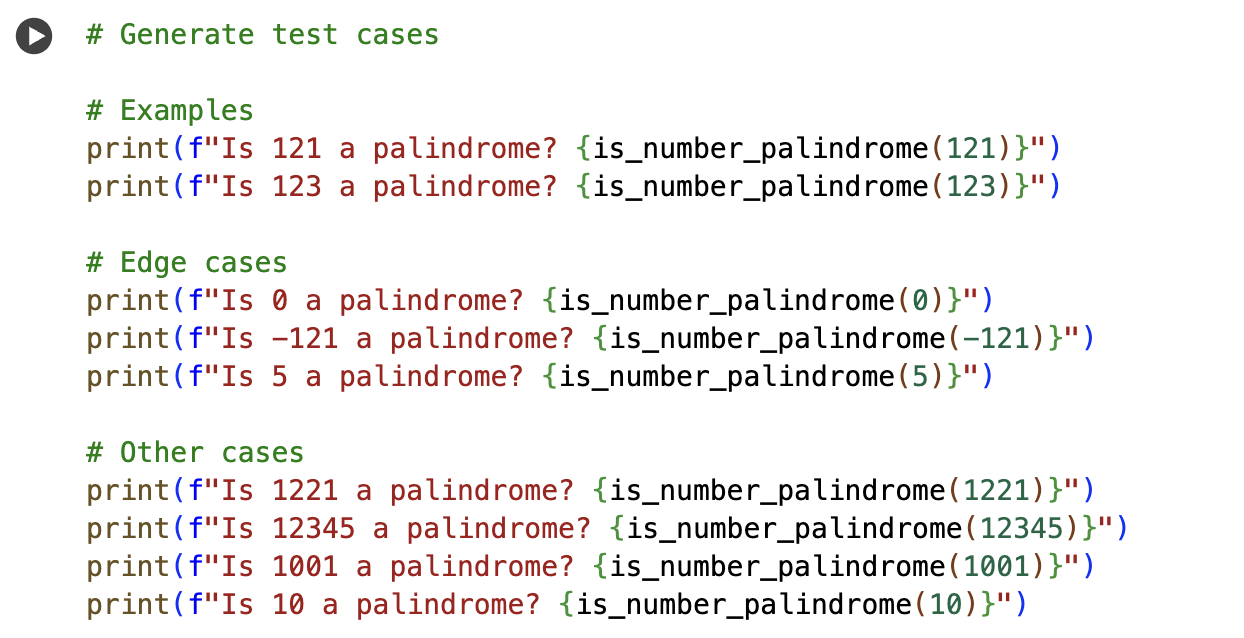
***TASK 5~***

*Generate test cases for is\_number\_palindrome(num), which checks if an integer reads  
the same backward.  
Examples:  
121 → True  
123 → False  
0, negative numbers → handled gracefully*

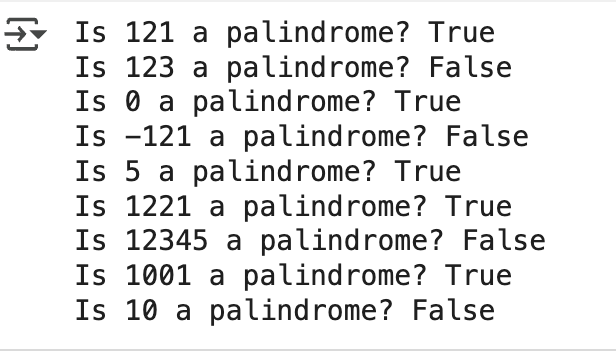
*PROMPT:*

*CODE:*

**

**

*OUTPUT:*

**

*EXPLANATION:*

*This code cell contains test cases for the is\_number\_palindrome function. It calls the function with various integer inputs, including:*

* *The examples provided (121 and 123).*
* *Edge cases like 0, a negative number (-121), and a single-digit positive number (5).*
* *Other numbers that are palindromes (1221, 1001) and not palindromes (12345, 10).*

*For each input, it prints whether the number is a palindrome based on the function's output, allowing you to verify the function's behavior for different scenarios.*