

Lab Assignment 08(Monday)

Name:M.Hasini

Hallticket:2303A51109

Batch-02

Task Description #1 (Username Validator – Apply AI in

Authentication Context)

- Task: Use AI to generate at least 3 assert test cases for a function is_valid_username(username) and then implement the function using Test-Driven Development principles.
- Requirements:
 - o Username length must be between 5 and 15 characters.
 - o Must contain only alphabets and digits.
 - o Must not start with a digit.
 - o No spaces allowed.

Example Assert Test Cases:

```
assert is_valid_username("User123") == True  
assert is_valid_username("12User") == False  
assert is_valid_username("Us er") == False
```

Expected Output #1:

- Username validation logic successfully passing all AI-generated test cases.

The screenshot shows a code editor window with a dark theme. The file is named 'lab_09.py'. The code defines a function 'is_valid_username' that checks if a given username is valid based on specific rules. The code is as follows:

```
1  def is_valid_username(username):  
2      if Len(username) < 3 or Len(username) > 16:  
3          return False  
4      if not username.isalnum():  
5          return False  
6      return True
```

The code editor includes standard navigation and search tools at the top, such as 'Accept', 'Tab', and 'Ctrl' keys.

```

lab_09.py > ...
1  def is_vaild_username(username):
2      if len(username) < 3 or len(username) > 16:
3          return False
4      if not username.isalnum():
5          return False
6      return True
7  assert is_vaild_username("user123") == True
8  assert is_vaild_username("us") == False
9  assert is_vaild_username("this_is_a_very_long_username") == False
10 assert is_vaild_username("user!@#") == False
11 print("All test cases passed!")

```

Code:

```

def is_vaild_username(username):
    if len(username) < 3 or len(username) > 16:
        return False
    if not username.isalnum():
        return False
    return True
assert is_vaild_username("user123") == True
assert is_vaild_username("us") == False
assert is_vaild_username("this_is_a_very_long_username") == False
assert is_vaild_username("user!@#") == False
print("All test cases passed!")

```

Output:

```

● PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasin/lab_09.py
    All test cases passed!
○ PS C:\Users\hasin> []

```

Task Description #2 (Even–Odd & Type Classification – Apply)

AI for Robust Input Handling)

- Task: Use AI to generate at least 3 assert test cases for a function classify_value(x) and implement it using conditional logic and loops.
- Requirements:
 - o If input is an integer, classify as "Even" or "Odd".
 - o If input is 0, return "Zero".
 - o If input is non-numeric, return "Invalid Input".

Example Assert Test Cases:

```

assert classify_value(8) == "Even"
assert classify_value(7) == "Odd"
assert classify_value("abc") == "Invalid Input"

```

Expected Output #2:

- Function correctly classifying values and passing all test cases.

```

12
13 def classify_value(x):
14     try:
15         if x == 0:
16             return "Zero"
17         elif x % 2 == 0:
18             return "Even"
19         else:
20             return "Odd"
21     except TypeError:
22         return "Invalid input: please provide an integer"
23
24 assert classify_value(4) == "Even"
25 assert classify_value(7) == "Odd"
26 assert classify_value(0) == "Zero"
27 assert classify_value("string") == "Invalid input: please provide an integer"
28
29 print("All test cases passed!")

```

Code:

```

def classify_value(x):
    try:
        if x == 0:
            return "Zero"
        elif x % 2 == 0:
            return "Even"
        else:
            return "Odd"
    except TypeError:
        return "Invalid input: please provide an integer"

assert classify_value(4) == "Even"
assert classify_value(7) == "Odd"
assert classify_value(0) == "Zero"
assert classify_value("string") == "Invalid input: please provide an integer"

print("All test cases passed!")

```

output:

- PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasin/PycharmProjects/untitled/main.py
All test cases passed!
- PS C:\Users\hasin> ◻

Task Description #3 (Palindrome Checker – Apply AI for

String Normalization)

- Task: Use AI to generate at least 3 assert test cases for a function `is_palindrome(text)` and implement the function.
- Requirements:
 - o Ignore case, spaces, and punctuation.
 - o Handle edge cases such as empty strings and single characters.

Example Assert Test Cases:

```
assert is_palindrome("Madam") == True  
assert is_palindrome("A man a plan a canal Panama") == True  
assert is_palindrome("Python") == False
```

Expected Output #3:

- Function correctly identifying palindromes and passing all AI-generated tests.

```
31  def is_palindrome(s):  
32      cleaned_str = ''.join(s.split()).lower()  
33      return cleaned_str == cleaned_str[::-1]  
34  assert is_palindrome("A man a plan a canal Panama") == True  
35  assert is_palindrome("Hello") == False  
36  assert is_palindrome("No 'x' in Nixon") == True  
37  assert is_palindrome("Was it a car or a cat I saw") == True  
38  assert is_palindrome("Not a palindrome") == False  
39  print("All test cases passed!")
```

Code:

```
def is_palindrome(s):  
    cleaned_str = ''.join(s.split()).lower()  
    return cleaned_str == cleaned_str[::-1]  
assert is_palindrome("A man a plan a canal Panama") == True  
assert is_palindrome("Hello") == False  
assert is_palindrome("No 'x' in Nixon") == True
```

```
assert is_palindrome("Was it a car or a cat I saw") == True
assert is_palindrome("Not a palindrome") == False
print("All test cases passed!")
```

output:

```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312
All test cases passed!
PS C:\Users\hasin> []
```

Task Description #4 (Email ID Validation – Apply AI for Data)

Validation)

- Task: Use AI to generate at least 3 assert test cases for a function validate_email(email) and implement the function.
- Requirements:
 - o Must contain @ and .
 - o Must not start or end with special characters.
 - o Should handle invalid formats gracefully.

Example Assert Test Cases:

```
assert validate_email("user@example.com") == True
assert validate_email("userexample.com") == False
assert validate_email("@gmail.com") == False
```

Expected Output #5:

- Email validation function passing all AI-generated test cases and handling edge cases correctly.

```
41 def validate_email(email):
42     import re
43     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$'
44     if re.match(pattern, email):
45         return "Valid Email"
46     else:
47         return "Invalid Email"
48 assert validate_email("test@example.com") == "Valid Email"
49 assert validate_email("invalid.email") == "Invalid Email"
50 assert validate_email("user@domain.co.uk") == "Valid Email"
51 assert validate_email("user@domain") == "Invalid Email"
52 assert validate_email("user@.com") == "Invalid Email"
53 print(["All test cases passed!"])"")
```

Code:

```
def validate_email(email):
    import re
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$'
    if re.match(pattern, email):
        return "Valid Email"
    else:
        return "Invalid Email"
assert validate_email("test@example.com") == "Valid Email"
assert validate_email("invalid.email") == "Invalid Email"
assert validate_email("user@domain.co.uk") == "Valid Email"
assert validate_email("user@domain") == "Invalid Email"
assert validate_email("user@.com") == "Invalid Email"
print("All test cases passed!")
```

output:

```
▶ PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312
  All test cases passed!
▶ PS C:\Users\hasin> □
```

Task 5 (Perfect Number Checker – Test Case Design)

- Function: Check if a number is a perfect number (sum of divisors = number).
- Test Cases to Design:
 - Normal case: 6 → True, 10 → False.
 - Edge case: 1.
 - Negative number case.
 - Larger case: 28.
- Requirement: Validate correctness with assertions.

```

54
55     def perfect_number(n):
56         if n < 1:
57             return False
58         divisors_sum = sum(i for i in range(1, n) if n % i == 0)
59         return divisors_sum == n
60     assert perfect_number(6) == True
61     assert perfect_number(28) == True
62     assert perfect_number(12) == False
63     assert perfect_number(496) == True
64     assert perfect_number(8128) == True
65     assert perfect_number(0) == False
66     assert perfect_number(-1) == False
67     print("All test cases passed!")"""

```

Code:

```

def perfect_number(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n
assert perfect_number(6) == True
assert perfect_number(28) == True
assert perfect_number(12) == False
assert perfect_number(496) == True
assert perfect_number(8128) == True
assert perfect_number(0) == False
assert perfect_number(-1) == False
print("All test cases passed!")

```

output:

SyntaxError: invalid syntax

- PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:\Users\hasin\PycharmProjects\Task 6\Task 6.py
 All test cases passed!
- PS C:\Users\hasin> []

Task 6 (Abundant Number Checker – Test Case Design)

- Function: Check if a number is abundant (sum of divisors > number).
- Test Cases to Design:
 - o Normal case: 12 → True, 15 → False.
 - o Edge case: 1.
 - o Negative number case.
 - o Large case: 945.

Requirement: Validate correctness with unittest

```
68
69  import unittest
70
71
72  def number_abundant(n):
73      if n < 1:
74          return False
75      divisors_sum = sum(i for i in range(1, n) if n % i == 0)
76      return divisors_sum > n
77  class TestNumberAbundant(unittest.TestCase):
78      def test_abundant_numbers(self):
79          self.assertTrue(number_abundant(12))
80          self.assertTrue(number_abundant(18))
81          self.assertTrue(number_abundant(20))
82          self.assertTrue(number_abundant(24))
83      def test_non_abundant_numbers(self):
84          self.assertFalse(number_abundant(6))
85          self.assertFalse(number_abundant(28))
86          self.assertFalse(number_abundant(496))
87          self.assertFalse(number_abundant(8128))
88      def test_edge_cases(self):
89          self.assertFalse(number_abundant(0))
90          self.assertFalse(number_abundant(-1))
91          self.assertFalse(number_abundant(-5))
92  if __name__ == "__main__":
93      unittest.main()
```

Code:

```
import unittest

def number_abundant(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum > n
class TestNumberAbundant(unittest.TestCase):
    def test_abundant_numbers(self):
        self.assertTrue(number_abundant(12))
        self.assertTrue(number_abundant(18))
        self.assertTrue(number_abundant(20))
        self.assertTrue(number_abundant(24))
    def test_non_abundant_numbers(self):
        self.assertFalse(number_abundant(6))
        self.assertFalse(number_abundant(28))
        self.assertFalse(number_abundant(496))
        self.assertFalse(number_abundant(8128))
    def test_edge_cases(self):
        self.assertFalse(number_abundant(0))
```

```

        self.assertFalse(number_abundant(-1))
        self.assertFalse(number_abundant(-5))
if __name__ == "__main__":
    unittest.main()

```

Output:

```

● PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe "c:/...
...
-----
Ran 3 tests in 0.002s

OK

```

Task 7 (Deficient Number Checker – Test Case Design)

- Function: Check if a number is deficient (sum of divisors < number).
- Test Cases to Design:
 - Normal case: 8 → True, 12 → False.
 - Edge case: 1.
 - Negative number case.
 - Large case: 546.

Requirement: Validate correctness with pytest.

```

94
95  def number_deficient(n):
96      if n < 1:
97          return False
98      divisors_sum = sum(i for i in range(1, n) if n % i == 0)
99      return divisors_sum < n
100 def test_number_deficient():
101     assert number_deficient(8) == True
102     assert number_deficient(15) == True
103     assert number_deficient(21) == True
104     assert number_deficient(27) == True
105     assert number_deficient(6) == False
106     assert number_deficient(28) == False
107     assert number_deficient(496) == False
108     assert number_deficient(8128) == False
109     assert number_deficient(0) == False
110     assert number_deficient(-1) == False
111     assert number_deficient(-5) == False
112     print("All test cases passed!")
113

```

Code:

```
def number_deficient(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum < n
def test_number_deficient():
    assert number_deficient(8) == True
    assert number_deficient(15) == True
    assert number_deficient(21) == True
    assert number_deficient(27) == True
    assert number_deficient(6) == False
    assert number_deficient(28) == False
    assert number_deficient(496) == False
    assert number_deficient(8128) == False
    assert number_deficient(0) == False
    assert number_deficient(-1) == False
    assert number_deficient(-5) == False
    print("All test cases passed!")
```

Output:

```
*PS C:\Users\hasin> py -m pytest
=====
platform win32 -- Python 3.12.1, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\hasin
collecting 0 items / 2 errors
```

Task 8:

Write a function LeapYearChecker and validate its implementation

using 10 pytest test cases

```
115 def leapyearchecker(year):
116     if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
117         return True
118     else:
119         return False
120 def test_leapyearchecker():
121     assert leapyearchecker(2020) == True
122     assert leapyearchecker(2021) == False
123     assert leapyearchecker(1900) == False
124     assert leapyearchecker(2000) == True
125     assert leapyearchecker(2100) == False
126     assert leapyearchecker(2400) == True
127     assert leapyearchecker(0) == True
128     assert leapyearchecker(-4) == True
129     assert leapyearchecker(-100) == False
130     assert leapyearchecker(-400) == True
131     assert leapyearchecker(-1900) == False
132     print("All test cases passed!")
```

Code:

```
def leapyearchecker(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
def test_leapyearchecker():
    assert leapyearchecker(2020) == True
    assert leapyearchecker(2021) == False
    assert leapyearchecker(1900) == False
    assert leapyearchecker(2000) == True
    assert leapyearchecker(2100) == False
    assert leapyearchecker(2400) == True
    assert leapyearchecker(0) == True
    assert leapyearchecker(-4) == True
    assert leapyearchecker(-100) == False
    assert leapyearchecker(-400) == True
    assert leapyearchecker(-1900) == False
    print("All test cases passed!")
```

output:

```
* PS C:\Users\hasin> py -m pytest
=====
platform win32 -- Python 3.12.1, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\hasin
collecting 0 items / 2 errors
```

Task 9 :

Write a function SumOfDigits and validate its implementation

using 7 pytest test cases.

```
121
122     def sumofdigits(n):
123         if n < 0:
124             return "Invalid input: please provide a non-negative integer"
125         return sum(int(digit) for digit in str(n))
126     def test_sumofdigits():
127         assert sumofdigits(123) == 6
128         assert sumofdigits(0) == 0
129         assert sumofdigits(999) == 27
130         assert sumofdigits(4567) == 22
131         assert sumofdigits(-123) == "Invalid input: please provide a non-negative integer"
132         assert sumofdigits(-1) == "Invalid input: please provide a non-negative integer"
133         assert sumofdigits(-100) == "Invalid input: please provide a non-negative integer"
134     print("All test cases passed!")
```

Code:

```
def sumofdigits(n):
    if n < 0:
        return "Invalid input: please provide a non-negative integer"
    return sum(int(digit) for digit in str(n))
def test_sumofdigits():
    assert sumofdigits(123) == 6
    assert sumofdigits(0) == 0
    assert sumofdigits(999) == 27
    assert sumofdigits(4567) == 22
    assert sumofdigits(-123) == "Invalid input: please provide a non-negative
integer"
    assert sumofdigits(-1) == "Invalid input: please provide a non-negative
integer"
    assert sumofdigits(-100) == "Invalid input: please provide a non-negative
integer"
    print("All test cases passed!")
```

Output:

```
*PS C:\Users\hasin> py -m pytest
=====
platform win32 -- Python 3.12.1, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\hasin
collecting 0 items / 2 errors
```

Task 10:

Write a function SortNumbers (implement bubble sort) and validate

its implementation using 25 pytest test cases.

```

136  def sortnumbers(nums):
137      n = len(nums)
138      for i in range(n):
139          for j in range(0, n-i-1):
140              if nums[j] > nums[j+1]:
141                  nums[j], nums[j+1] = nums[j+1], nums[j]
142  def test_sortnumbers():
143      nums1 = [5, 2, 9, 1, 5, 6]
144      sortnumbers(nums1)
145      assert nums1 == [1, 2, 5, 5, 6, 9]
146
147      nums2 = [3, 0, -1, 8, 7]
148      sortnumbers(nums2)
149      assert nums2 == [-1, 0, 3, 7, 8]
150
151      nums3 = [10]
152      sortnumbers(nums3)
153      assert nums3 == [10]
154
155      nums4 = []
156      sortnumbers(nums4)
157      assert nums4 == []
158      nums5 = [1, 2, 3, 4, 5]
159      sortnumbers(nums5)
160      assert nums5 == [1, 2, 3, 4, 5]
161      nums6 = [5, 4, 3, 2, 1]
162      sortnumbers(nums6)
163      assert nums6 == [1, 2, 3, 4, 5]
164      nums7 = [1, 1, 1, 1]
165      sortnumbers(nums7)
166      assert nums7 == [1, 1, 1, 1]
167      nums8 = [2, 3, 2, 1, 3]
168      sortnumbers(nums8)
169      assert nums8 == [1, 2, 2, 3, 3]
170      nums10 = [0, 0, 0, 0]
171      sortnumbers(nums10)
172      assert nums10 == [0, 0, 0, 0]
173      nums11 = [5, 3, 8, 6, 2]
174      sortnumbers(nums11)
175      assert nums11 == [2, 3, 5, 6, 8]

```

Code:

```

def sortnumbers(nums):
    n = len(nums)
    for i in range(n):
        for j in range(0, n-i-1):
            if nums[j] > nums[j+1]:
                nums[j], nums[j+1] = nums[j+1], nums[j]
def test_sortnumbers():
    nums1 = [5, 2, 9, 1, 5, 6]
    sortnumbers(nums1)
    assert nums1 == [1, 2, 5, 5, 6, 9]

    nums2 = [3, 0, -1, 8, 7]

```

```

sortnumbers(nums2)
assert nums2 == [-1, 0, 3, 7, 8]

nums3 = [10]
sortnumbers(nums3)
assert nums3 == [10]

nums4 = []
sortnumbers(nums4)
assert nums4 == []
nums5 = [1, 2, 3, 4, 5]
sortnumbers(nums5)
assert nums5 == [1, 2, 3, 4, 5]
nums6 = [5, 4, 3, 2, 1]
sortnumbers(nums6)
assert nums6 == [1, 2, 3, 4, 5]
nums7 = [1, 1, 1, 1]
sortnumbers(nums7)
assert nums7 == [1, 1, 1, 1]
nums8 = [2, 3, 2, 1, 3]
sortnumbers(nums8)
assert nums8 == [1, 2, 2, 3, 3]
nums10 = [0, 0, 0, 0]
sortnumbers(nums10)
assert nums10 == [0, 0, 0, 0]
nums11 = [5, 3, 8, 6, 2]
sortnumbers(nums11)
assert nums11 == [2, 3, 5, 6, 8]
nums12 = [9, 7, 5, 3, 1]
sortnumbers(nums12)
assert nums12 == [1, 3, 5, 7, 9]

print("All test cases passed!")

```

Output:

Task 11 :

Write a function ReverseString and validate its implementation
using 5 unittest test cases

```

182
183     import unittest
184
185     def reverse_string(s):
186         return s[::-1]
187
188     class TestReverseString(unittest.TestCase):
189         def test_reverse_string(self):
190             self.assertEqual(reverse_string("hello"), "olleh")
191             self.assertEqual(reverse_string("Python"), "nohtyP")
192             self.assertEqual(reverse_string(""), "")
193             self.assertEqual(reverse_string("a"), "a")
194             self.assertEqual(reverse_string("12345"), "54321")
195             self.assertEqual(reverse_string("racecar"), "racecar")
196             self.assertEqual(reverse_string("A man a plan a canal Panama"), "amanaP lanac a nlp a nam A")
197
198     if __name__ == "__main__":
199         unittest.main()

```

code:

```

import unittest

def reverse_string(s):
    return s[::-1]
class TestReverseString(unittest.TestCase):
    def test_reverse_string(self):
        self.assertEqual(reverse_string("hello"), "olleh")
        self.assertEqual(reverse_string("Python"), "nohtyP")
        self.assertEqual(reverse_string(""), "")
        self.assertEqual(reverse_string("a"), "a")
        self.assertEqual(reverse_string("12345"), "54321")
        self.assertEqual(reverse_string("racecar"), "racecar")
        self.assertEqual(reverse_string("A man a plan a canal Panama"),
"amanaP lanac a nlp a nam A")

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

```

output:

```

.
-----
Ran 1 test in 0.000s
OK

```

Task 12 :

Write a function AnagramChecker and validate its implementation using 10 unittest test cases.

```

198
199 import unittest
200
201
202 def anagram_checker(str1, str2):
203     return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
204 class TestAnagramChecker(unittest.TestCase):
205     def test_anagram_checker(self):
206         self.assertTrue(anagram_checker("listen", "silent"))
207         self.assertFalse(anagram_checker("hello", "world"))
208         self.assertTrue(anagram_checker("Dormitory", "Dirty Room"))
209         self.assertTrue(anagram_checker("The eyes", "They see"))
210         self.assertTrue(anagram_checker("Astronomer", "Moon starer"))
211         self.assertTrue(anagram_checker("Conversation", "Voices rant on"))
212 if __name__ == "__main__":
213     unittest.main()

```

Code:

```

import unittest

def anagram_checker(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
class TestAnagramChecker(unittest.TestCase):
    def test_anagram_checker(self):
        self.assertTrue(anagram_checker("listen", "silent"))
        self.assertFalse(anagram_checker("hello", "world"))
        self.assertTrue(anagram_checker("Dormitory", "Dirty Room"))
        self.assertTrue(anagram_checker("The eyes", "They see"))
        self.assertTrue(anagram_checker("Astronomer", "Moon starer"))
        self.assertTrue(anagram_checker("Conversation", "Voices rant on"))
if __name__ == "__main__":
    unittest.main()

```

output:

```

● PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasin/OneDrive/Desktop/twon
.
-----
Ran 1 test in 0.001s

OK
○ PS C:\Users\hasin>

```

Task 13 :

Write a function ArmstrongChecker and validate its implementation using 8 unittest test cases.

```
215 import unittest
216
217
218 def armstrong_number(n):
219     num_str = str(n)
220     num_digits = len(num_str)
221     armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
222     return armstrong_sum == n
223 class TestArmstrongNumber(unittest.TestCase):
224     def test_armstrong_number(self):
225         self.assertTrue(armstrong_number(153))
226         self.assertTrue(armstrong_number(370))
227         self.assertTrue(armstrong_number(371))
228         self.assertTrue(armstrong_number(407))
229         self.assertFalse(armstrong_number(123))
230         self.assertFalse(armstrong_number(0))
231         self.assertFalse(armstrong_number(-153))
232 if __name__ == "__main__":
233     unittest.main()
234
```

Code:

```
import unittest

def armstrong_number(n):
    num_str = str(n)
    num_digits = len(num_str)
    armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
    return armstrong_sum == n
class TestArmstrongNumber(unittest.TestCase):
    def test_armstrong_number(self):
        self.assertTrue(armstrong_number(153))
        self.assertTrue(armstrong_number(370))
        self.assertTrue(armstrong_number(371))
        self.assertTrue(armstrong_number(407))
        self.assertFalse(armstrong_number(123))
        self.assertFalse(armstrong_number(0))
        self.assertFalse(armstrong_number(-153))
if __name__ == "__main__":
    unittest.main()
```

Output:

```
FAIL: test_armstrong_number (__main__.TestArmstrongNumber.test_armstrong_number)
-----
Traceback (most recent call last):
  File "c:\Users\hasin\OneDrive\Desktop\import unittest.py", line 16, in test_armstrong_number
    self.assertFalse(armstrong_number(0))
AssertionError: True is not false

-----
Ran 1 test in 0.002s
```

