LAB_ASSIGNMENT-3.1

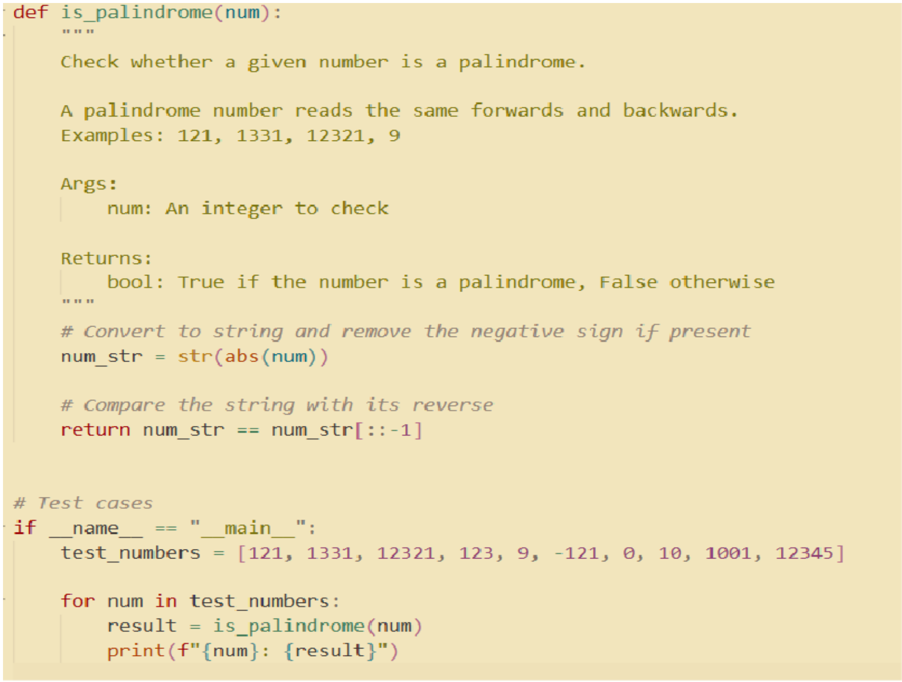
NAME: CHENCHU RAJU

ROLL NO: 2303A52206

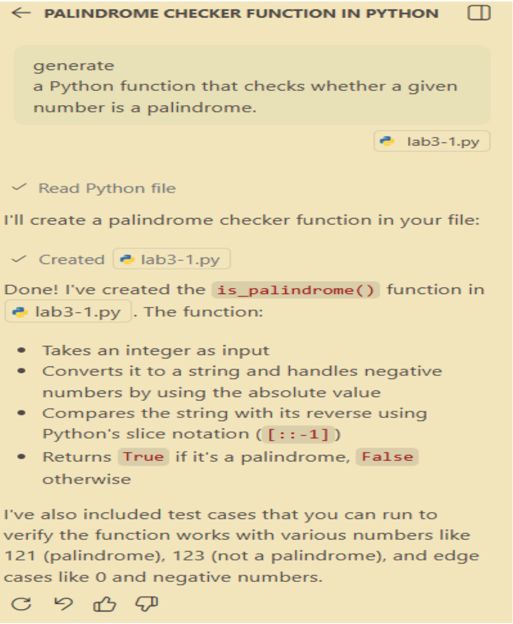
BATCH: 43

Question-1

CODE,CHAT AND OUTPUT:



```python
def is_palindrome(num):
    """
    Check whether a given number is a palindrome.

    A palindrome number reads the same forwards and backwards.
    Examples: 121, 1331, 12321, 9

    Args:
        num: An integer to check

    Returns:
        bool: True if the number is a palindrome, False otherwise
    """
    # Convert to string and remove the negative sign if present
    num_str = str(abs(num))

    # Compare the string with its reverse
    return num_str == num_str[::-1]


# Test cases
if __name__ == "__main__":
    test_numbers = [121, 1331, 12321, 123, 9, -121, 0, 10, 1001, 12345]

    for num in test_numbers:
        result = is_palindrome(num)
        print(f"{num}: {result}")
```



generate
a Python function that checks whether a given number is a palindrome.

✓ Read Python file

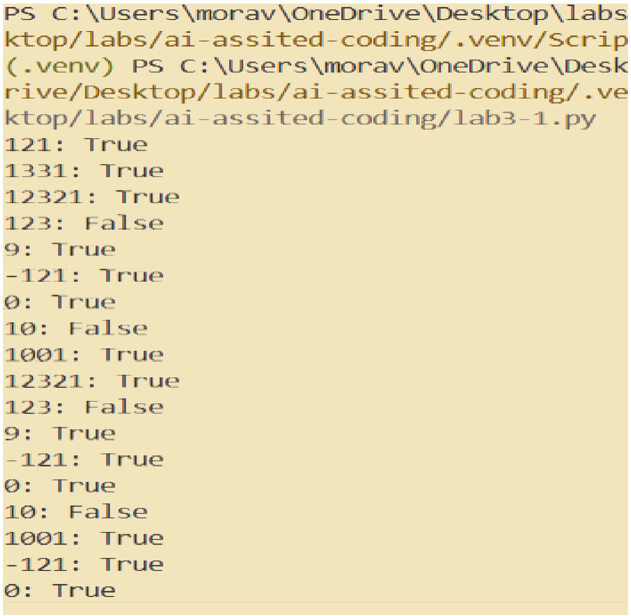
I'll create a palindrome checker function in your file:

✓ Created lab3-1.py

Done! I've created the `is_palindrome()` function in lab3-1.py. The function:

- Takes an integer as input
- Converts it to a string and handles negative numbers by using the absolute value
- Compares the string with its reverse using Python's slice notation (`[::-1]`)
- Returns `True` if it's a palindrome, `False` otherwise

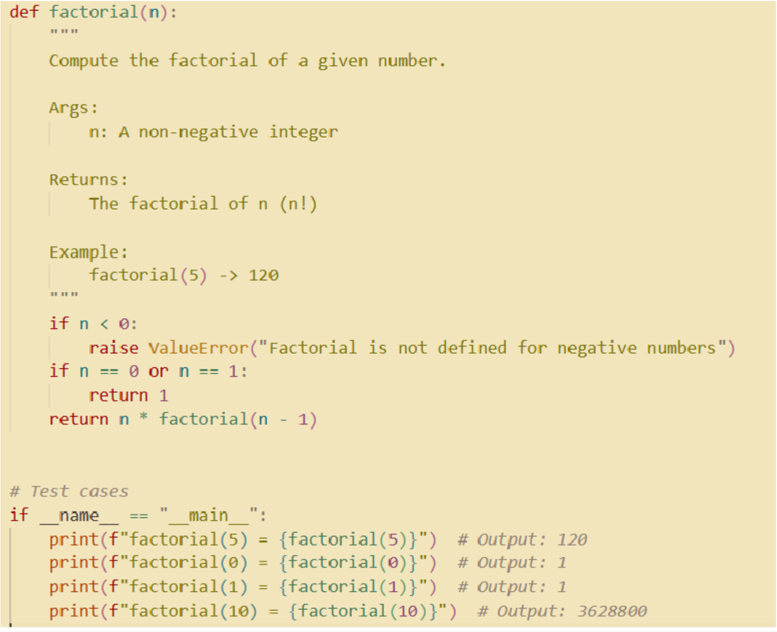
I've also included test cases that you can run to verify the function works with various numbers like 121 (palindrome), 123 (not a palindrome), and edge cases like 0 and negative numbers.



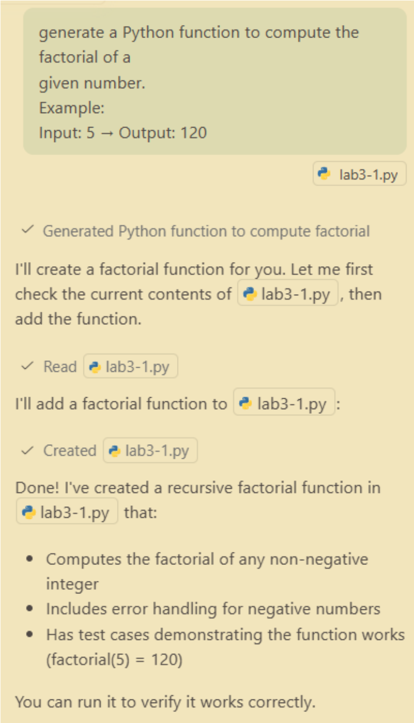
```
PS C:\Users\morav\OneDrive\Desktop\labs
ktop/labs/ai-assited-coding/.venv/Scrip
(.venv) PS C:\Users\morav\OneDrive\Desk
rive/Desktop/labs/ai-assited-coding/.ve
ktop/labs/ai-assited-coding/lab3-1.py
121: True
1331: True
12321: True
123: False
9: True
-121: True
0: True
10: False
1001: True
12321: True
123: False
9: True
-121: True
0: True
10: False
1001: True
-121: True
0: True
```

Question-2:

CHAT, OUTPUT AND CODE:



```python
def factorial(n):
    """
    Compute the factorial of a given number.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n (n!)

    Example:
        factorial(5) -> 120
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)


# Test cases
if __name__ == "__main__":
    print(f"factorial(5) = {factorial(5)}")  # Output: 120
    print(f"factorial(0) = {factorial(0)}")  # Output: 1
    print(f"factorial(1) = {factorial(1)}")  # Output: 1
    print(f"factorial(10) = {factorial(10)}")  # Output: 3628800
```



generate a Python function to compute the
factorial of a
given number.
Example:
Input: 5 → Output: 120

lab3-1.py

✓ Generated Python function to compute factorial

I'll create a factorial function for you. Let me first check the current contents of lab3-1.py, then add the function.

✓ Read lab3-1.py

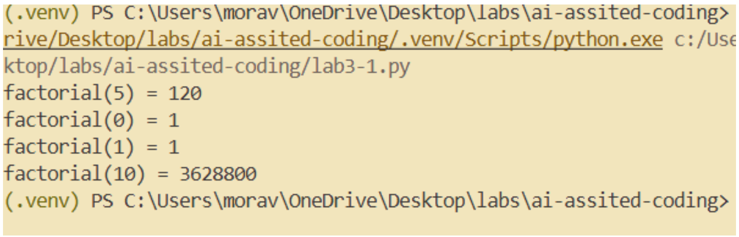
I'll add a factorial function to lab3-1.py:

✓ Created lab3-1.py

Done! I've created a recursive factorial function in lab3-1.py that:

- Computes the factorial of any non-negative integer
- Includes error handling for negative numbers
- Has test cases demonstrating the function works (factorial(5) = 120)

You can run it to verify it works correctly.

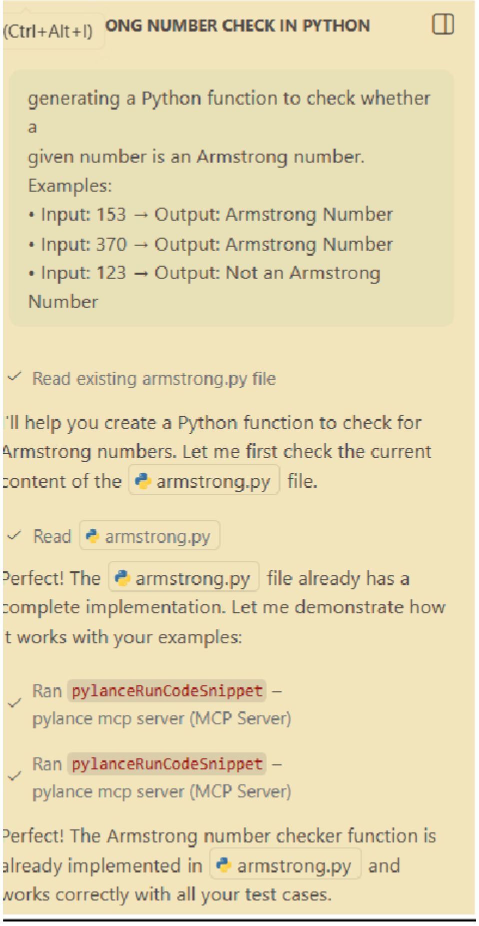


```
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding>
rive/Desktop/labs/ai-assited-coding/.venv/Scripts/python.exe c:/Use
ktop/labs/ai-assited-coding/lab3-1.py
factorial(5) = 120
factorial(0) = 1
factorial(1) = 1
factorial(10) = 3628800
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding>
```

Comparisons:

On comparing with the zero-shot solution the code is a little bit optimal and added more test cases and the printing of the output has also changes to the given test case

Question-3:



generating a Python function to check whether a
given number is an Armstrong number.
Examples:
- Input: 153 → Output: Armstrong Number
- Input: 370 → Output: Armstrong Number
- Input: 123 → Output: Not an Armstrong Number

✓ Read existing armstrong.py file

'll help you create a Python function to check for Armstrong numbers. Let me first check the current content of the armstrong.py file.

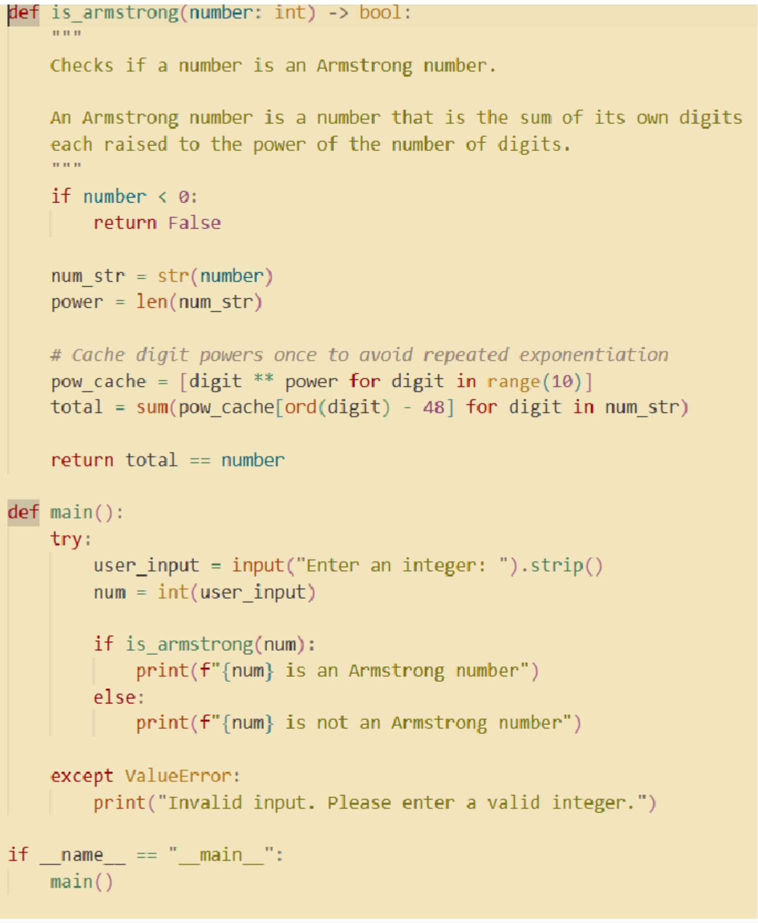
✓ Read armstrong.py

Perfect! The armstrong.py file already has a complete implementation. Let me demonstrate how t works with your examples:

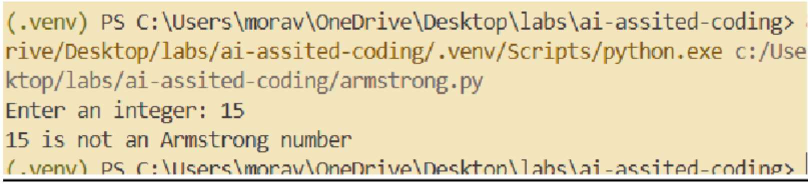
✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

✓ Ran `pylanceRunCodeSnippet` – pylance mcp server (MCP Server)

Perfect! The Armstrong number checker function is already implemented in armstrong.py and works correctly with all your test cases.

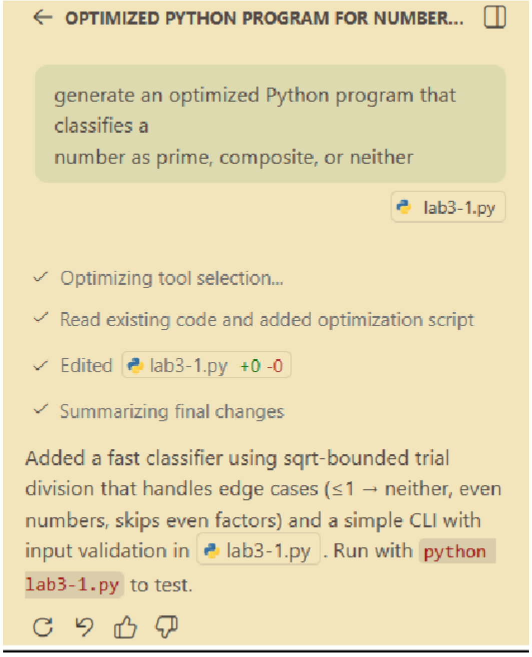


```python
def is_armstrong(number: int) -> bool:
    """
    Checks if a number is an Armstrong number.

    An Armstrong number is a number that is the sum of its own digits
    each raised to the power of the number of digits.
    """
    if number < 0:
        return False

    num_str = str(number)
    power = len(num_str)

    # Cache digit powers once to avoid repeated exponentiation
    pow_cache = [digit ** power for digit in range(10)]
    total = sum(pow_cache[ord(digit) - 48] for digit in num_str)

    return total == number

def main():
    try:
        user_input = input("Enter an integer: ").strip()
        num = int(user_input)

        if is_armstrong(num):
            print(f"{num} is an Armstrong number")
        else:
            print(f"{num} is not an Armstrong number")

    except ValueError:
        print("Invalid input. Please enter a valid integer.")

if __name__ == "__main__":
    main()
```



```
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding>
rive/Desktop/labs/ai-assited-coding/.venv/Scripts/python.exe c:/Use
ktop/labs/ai-assited-coding/armstrong.py
Enter an integer: 15
15 is not an Armstrong number
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding>
```

Question-4



generate an optimized Python program that classifies a
number as prime, composite, or neither

lab3-1.py

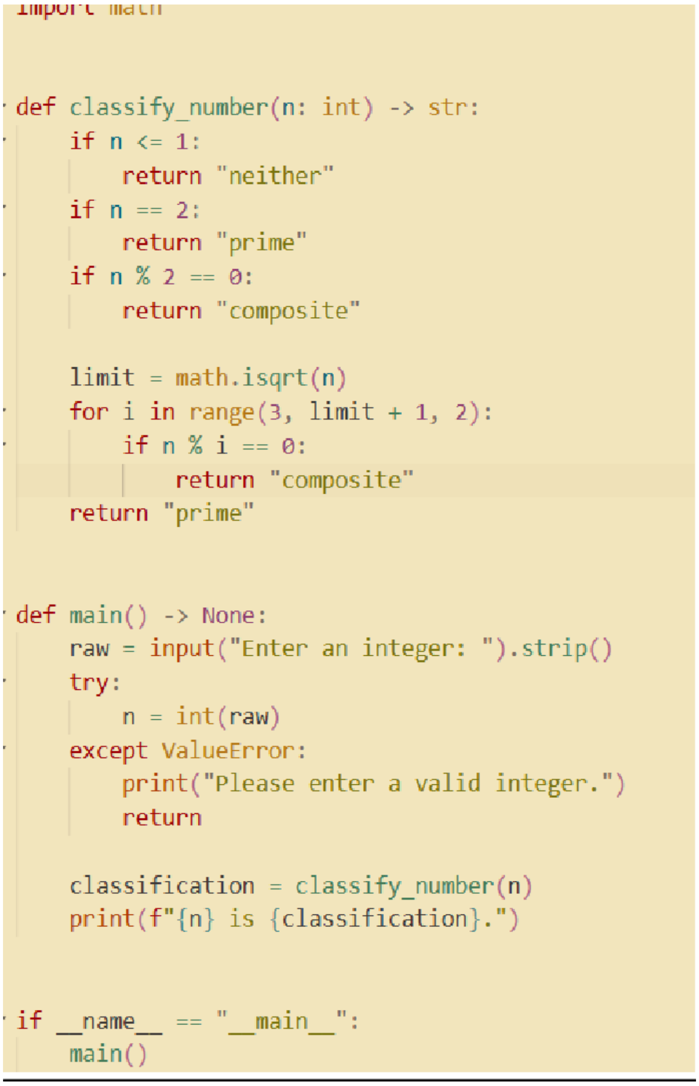
✓ Optimizing tool selection...

✓ Read existing code and added optimization script

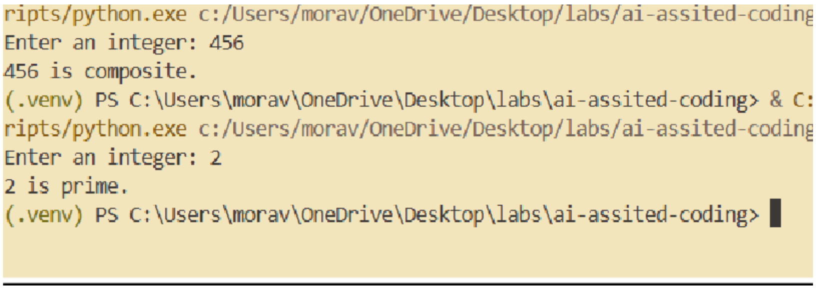
✓ Edited lab3-1.py +0 -0

✓ Summarizing final changes

Added a fast classifier using sqrt-bounded trial division that handles edge cases (≤1 → neither, even numbers, skips even factors) and a simple CLI with input validation in lab3-1.py. Run with `python lab3-1.py` to test.

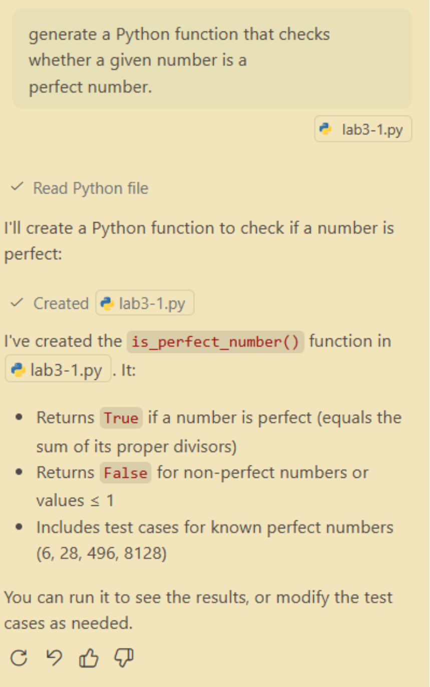


```
import math


def classify_number(n: int) -> str:
    if n <= 1:
        return "neither"
    if n == 2:
        return "prime"
    if n % 2 == 0:
        return "composite"

    limit = math.isqrt(n)
    for i in range(3, limit + 1, 2):
        if n % i == 0:
            return "composite"
    return "prime"


def main() -> None:
    raw = input("Enter an integer: ").strip()
    try:
        n = int(raw)
    except ValueError:
        print("Please enter a valid integer.")
        return

    classification = classify_number(n)
    print(f"{n} is {classification}.")


if __name__ == "__main__":
    main()
```



```
ripts/python.exe c:/Users/morav/OneDrive/Desktop/labs/ai-assited-coding
Enter an integer: 456
456 is composite.
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> & C:
ripts/python.exe c:/Users/morav/OneDrive/Desktop/labs/ai-assited-coding
Enter an integer: 2
2 is prime.
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> 
```

Question-5



generate a Python function that checks whether a given number is a perfect number.

✓ Read Python file

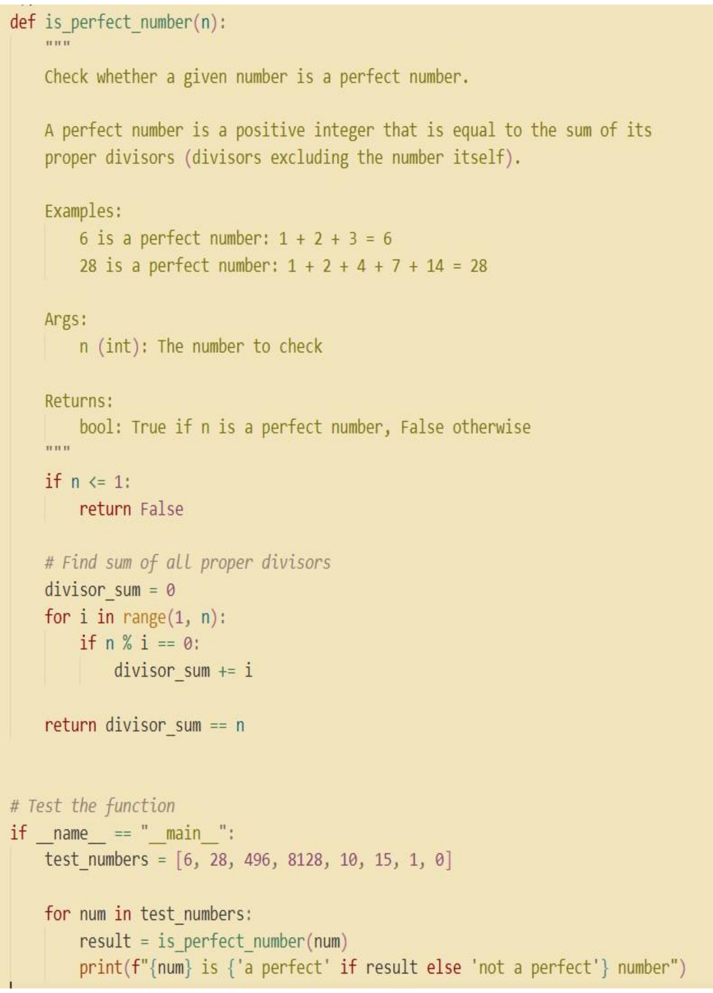
I'll create a Python function to check if a number is perfect:

✓ Created lab3-1.py

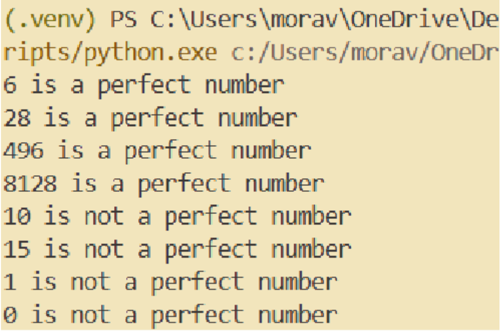
I've created the `is_perfect_number()` function in lab3-1.py. It:

- Returns `True` if a number is perfect (equals the sum of its proper divisors)
- Returns `False` for non-perfect numbers or values ≤ 1
- Includes test cases for known perfect numbers (6, 28, 496, 8128)

You can run it to see the results, or modify the test cases as needed.

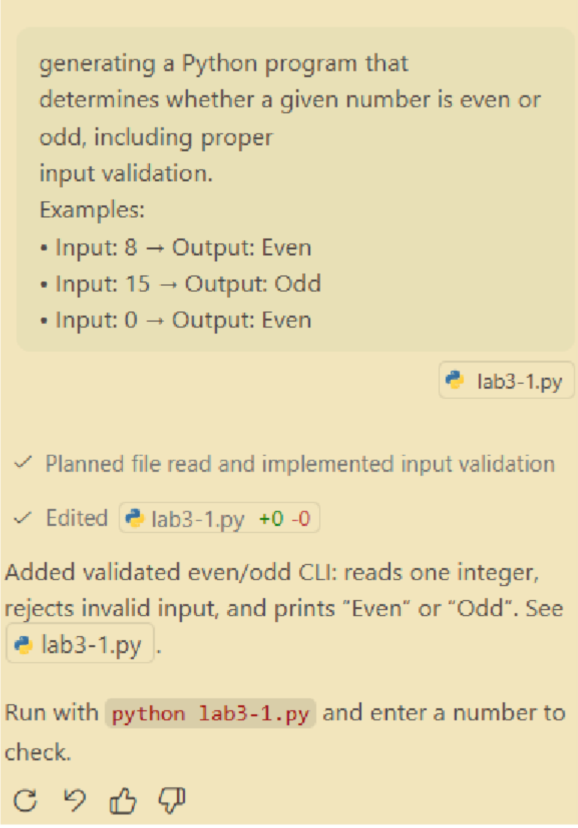


```python
def is_perfect_number(n):
    """
    Check whether a given number is a perfect number.

    A perfect number is a positive integer that is equal to the sum of its
    proper divisors (divisors excluding the number itself).

    Examples:
        6 is a perfect number: 1 + 2 + 3 = 6
        28 is a perfect number: 1 + 2 + 4 + 7 + 14 = 28

    Args:
        n (int): The number to check

    Returns:
        bool: True if n is a perfect number, False otherwise
    """
    if n <= 1:
        return False

    # Find sum of all proper divisors
    divisor_sum = 0
    for i in range(1, n):
        if n % i == 0:
            divisor_sum += i

    return divisor_sum == n


# Test the function
if __name__ == "__main__":
    test_numbers = [6, 28, 496, 8128, 10, 15, 1, 0]

    for num in test_numbers:
        result = is_perfect_number(num)
        print(f"{num} is {'a perfect' if result else 'not a perfect'} number")
```



```
(.venv) PS C:\Users\morav\OneDrive\De
ripts/python.exe c:/Users/morav/OneDr
6 is a perfect number
28 is a perfect number
496 is a perfect number
8128 is a perfect number
10 is not a perfect number
15 is not a perfect number
1 is not a perfect number
0 is not a perfect number
```

Question-6



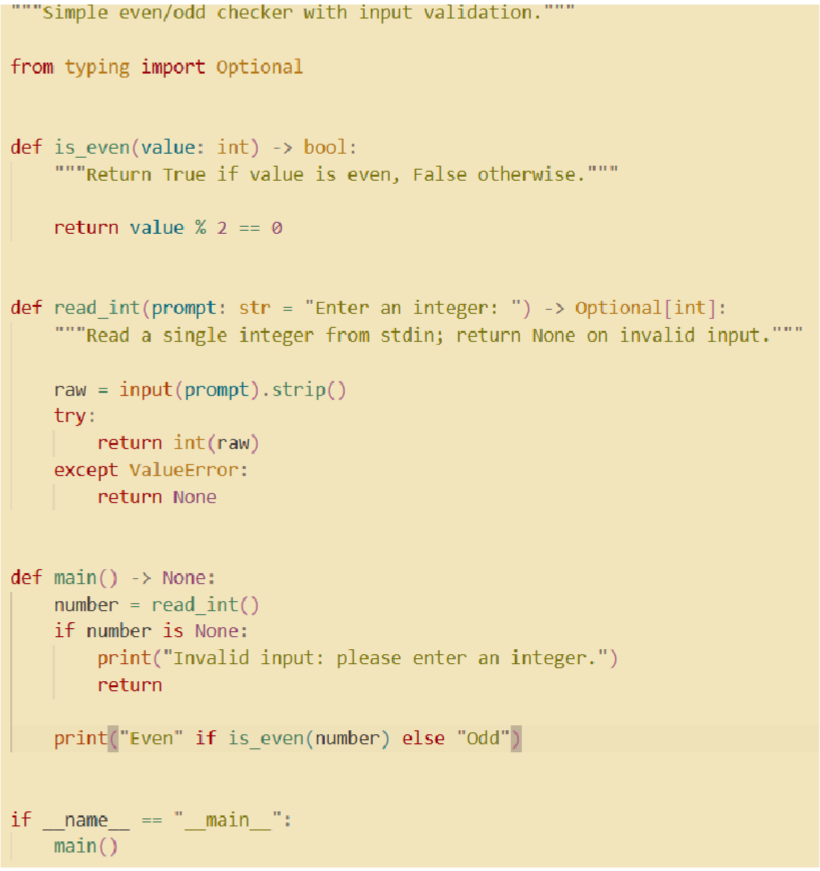
generating a Python program that determines whether a given number is even or odd, including proper input validation.
Examples:
- Input: 8 → Output: Even
- Input: 15 → Output: Odd
- Input: 0 → Output: Even

✓ Planned file read and implemented input validation

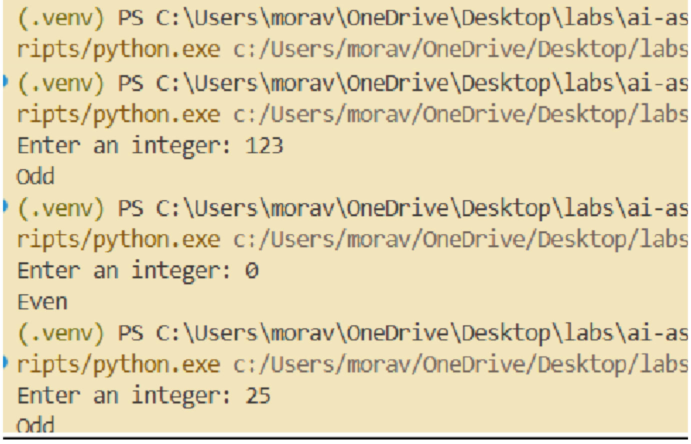
✓ Edited lab3-1.py +0 -0

Added validated even/odd CLI: reads one integer, rejects invalid input, and prints "Even" or "Odd". See lab3-1.py .

Run with `python lab3-1.py` and enter a number to check.



```python
"""Simple even/odd checker with input validation."""

from typing import Optional


def is_even(value: int) -> bool:
    """Return True if value is even, False otherwise."""

    return value % 2 == 0


def read_int(prompt: str = "Enter an integer: ") -> Optional[int]:
    """Read a single integer from stdin; return None on invalid input."""

    raw = input(prompt).strip()
    try:
        return int(raw)
    except ValueError:
        return None


def main() -> None:
    number = read_int()
    if number is None:
        print("Invalid input: please enter an integer.")
        return

    print("Even" if is_even(number) else "Odd")


if __name__ == "__main__":
    main()
```



```
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-as
ripts/python.exe c:/Users/morav/OneDrive/Desktop/labs
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-as
ripts/python.exe c:/Users/morav/OneDrive/Desktop/labs
Enter an integer: 123
Odd
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-as
ripts/python.exe c:/Users/morav/OneDrive/Desktop/labs
Enter an integer: 0
Even
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-as
ripts/python.exe c:/Users/morav/OneDrive/Desktop/labs
Enter an integer: 25
Odd
```