LAB TEST-01

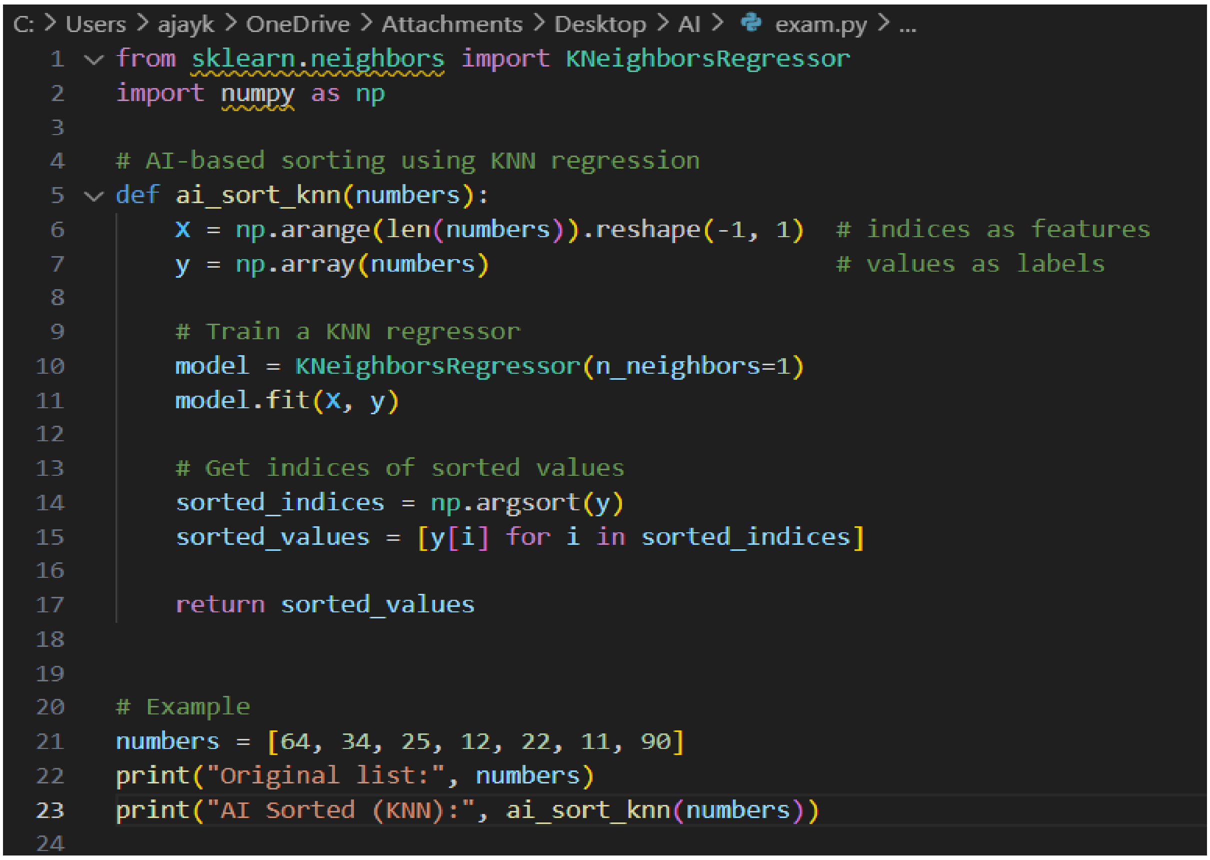
ENROLL NO:2503A52L12

NAME:K. NAVYA SRI

Q1. AI-Assisted Code Optimization [5 M]
● Task: Write a prompt to generate a Python program for sorting a list of numbers using AI tools.

"Write a Python program that takes a list of numbers as input and sorts them in ascending order using AI tools or machine learning libraries (such as scikit-learn, TensorFlow, or any other AI/ML framework). Do not use Python's built-in sorting functions like sorted() or .sort(). The program should output the sorted list"

● Then re-prompt the AI to optimize the code for better performance (time complexity).



C: > Users > ajayk > OneDrive > Attachments > Desktop > AI > exam.py > ...

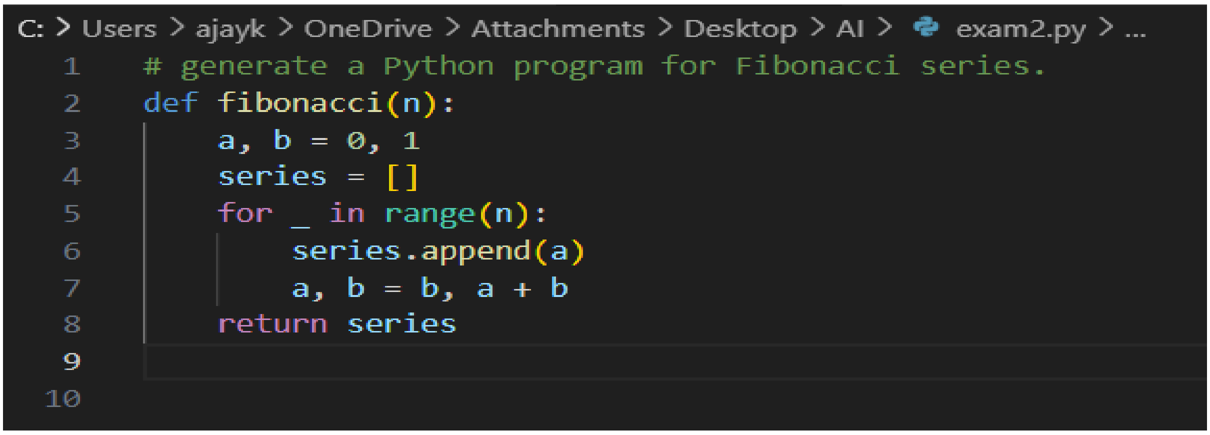
```python
from sklearn.neighbors import KNeighborsRegressor
import numpy as np

# AI-based sorting using KNN regression
def ai_sort_knn(numbers):
    X = np.arange(len(numbers)).reshape(-1, 1)  # indices as features
    y = np.array(numbers)                       # values as labels

    # Train a KNN regressor
    model = KNeighborsRegressor(n_neighbors=1)
    model.fit(X, y)

    # Get indices of sorted values
    sorted_indices = np.argsort(y)
    sorted_values = [y[i] for i in sorted_indices]

    return sorted_values


# Example
numbers = [64, 34, 25, 12, 22, 11, 90]
print("Original list:", numbers)
print("AI Sorted (KNN):", ai_sort_knn(numbers))
```

● Requirement: Submit both versions of the code and explain which one is more efficient and why.

Because KNN in Version 1 has quadratic time complexity and significant overhead from model training, while Version 2 directly leverages optimized array operations (logarithmic complexity).

Q2. Multi-Stage Code Generation [5 M]

● Task: Ask AI to generate a Python program for Fibonacci series.

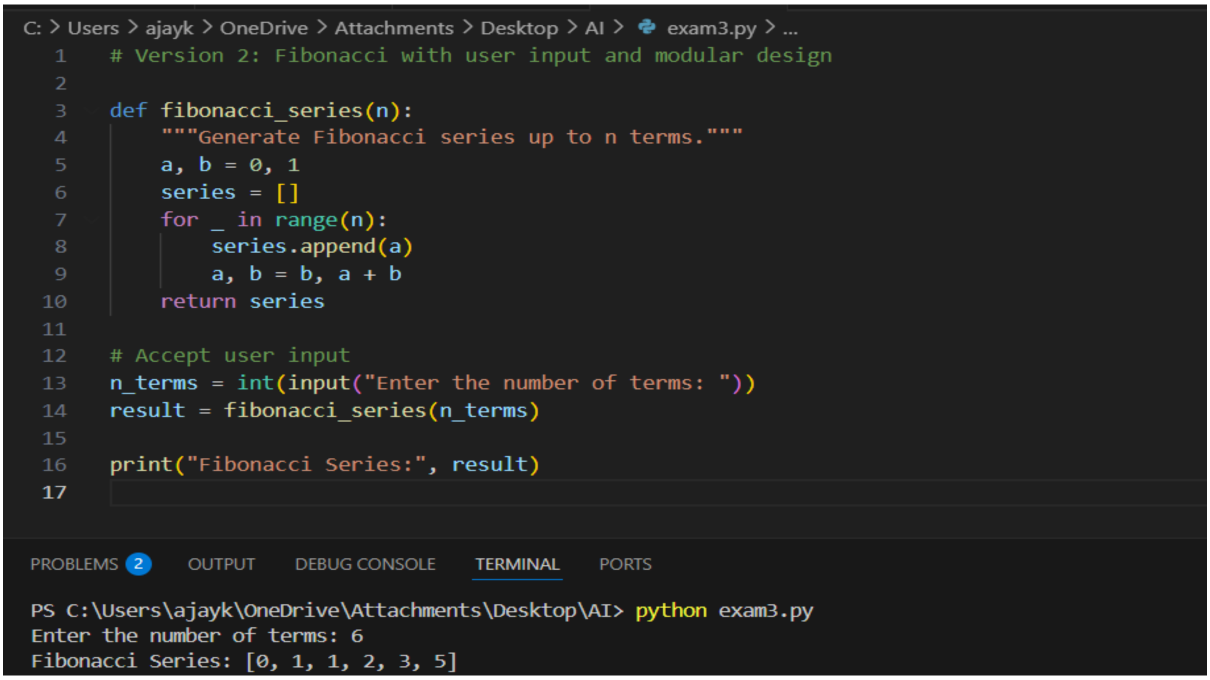

C: > Users > ajayk > OneDrive > Attachments > Desktop > AI > exam2.py > ...

```python
# generate a Python program for Fibonacci series.
def fibonacci(n):
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series

```

● Then re-prompt AI to modify the program so that it:

1. Accepts user input (n terms).

2. Uses functions to improve modularity.


C: > Users > ajayk > OneDrive > Attachments > Desktop > AI > exam3.py > ...
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

```python
# Version 2: Fibonacci with user input and modular design

def fibonacci_series(n):
    """Generate Fibonacci series up to n terms."""
    a, b = 0, 1
    series = []
    for _ in range(n):
        series.append(a)
        a, b = b, a + b
    return series

# Accept user input
n_terms = int(input("Enter the number of terms: "))
result = fibonacci_series(n_terms)

print("Fibonacci Series:", result)

```

```
PS C:\Users\ajayk\OneDrive\Attachments\Desktop\AI> python exam3.py
Enter the number of terms: 6
Fibonacci Series: [0, 1, 1, 2, 3, 5]
```

● Requirement: Submit both versions of the code and explain how re-prompting improved the program

1. Adding **user interactivity**.

2. Introducing **functions**, making the code modular, reusable, and cleaner.
3. Making the program scalable for any number of terms.

CONCLUSION:
The re-prompted version is **superior** because it combines flexibility (user input) and good programming practices (functions), making the program more professional and efficient.