ASSIGNMENT – 1.4

2303A51355

Batch-10

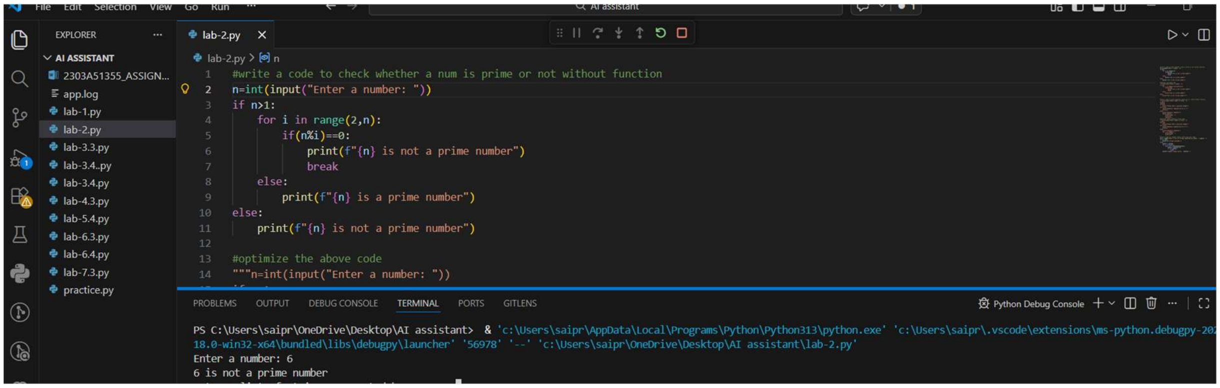
Task-1

Prompt: write a code to check whether a num is prime or not without function

code :

```
n=int(input("Enter a number: "))
if n>1:
    for i in range(2,n):
        if(n%i)==0:
            print(f"{n} is not a prime number")
            break
    else:
        print(f"{n} is a prime number")
else:
    print(f"{n} is not a prime number")
```

Output :



```
#write a code to check whether a num is prime or not without function
n=int(input("Enter a number: "))
if n>1:
    for i in range(2,n):
        if(n%i)==0:
            print(f"{n} is not a prime number")
            break
    else:
        print(f"{n} is a prime number")
else:
    print(f"{n} is not a prime number")

#optimize the above code
"""n=int(input("Enter a number: "))
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\saipr\.vscode\extensions\ms-python.debugpy-202
18.0-win32-x64\bundled\libs\debugpy\launcher' '56978' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assistant\lab-2.py'
Enter a number: 6
6 is not a prime number
```

Code Analysis:

□ The program takes an integer input and checks if it is greater than 1.

□ It uses a loop from 2 to n−1 to test divisibility.

□ If any divisor is found, it prints that the number is not prime and exits the loop.

□ If no divisor is found, it prints that the number is prime.

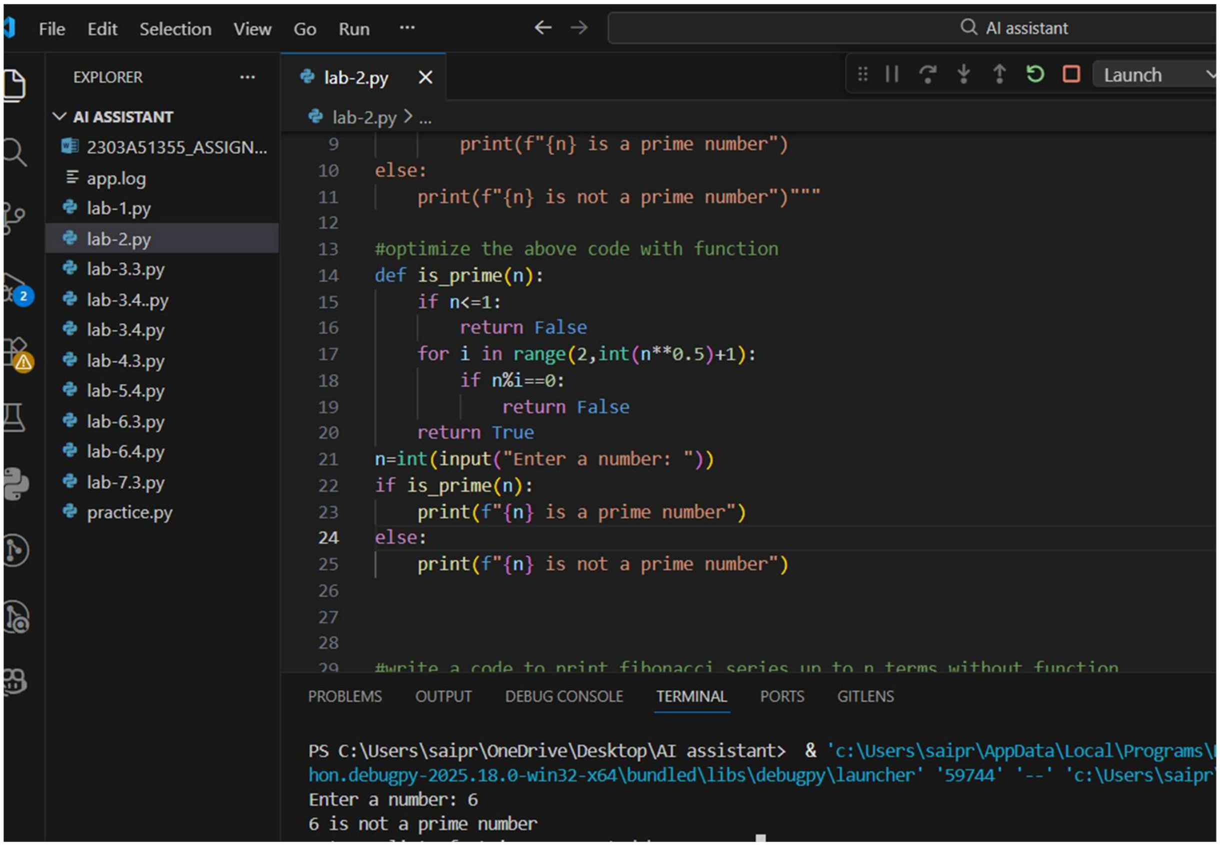
□ This approach is correct but inefficient for large numbers.

Task-2
Prompt: #optimize the above code using function

Code :

```
def is_prime(n):
    if n<=1:
        return False
    for i in range(2,int(n**0.5)+1):
        if n%i==0:
            return False
    return True
n=int(input("Enter a number: "))
if is_prime(n):
    print(f"{n} is a prime number")
else:
    print(f"{n} is not a prime number")
```

Output :



```python
        print(f"{n} is a prime number")
else:
    print(f"{n} is not a prime number")"""

#optimize the above code with function
def is_prime(n):
    if n<=1:
        return False
    for i in range(2,int(n**0.5)+1):
        if n%i==0:
            return False
    return True
n=int(input("Enter a number: "))
if is_prime(n):
    print(f"{n} is a prime number")
else:
    print(f"{n} is not a prime number")



#write a code to print fibonacci series up to n terms without function
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '59744' '--' 'c:\Users\saipr
Enter a number: 6
6 is not a prime number
```

Code Analysis:

□ The function checks divisibility only up to √n, reducing time complexity.

□ It returns False immediately when a divisor is found.

□ This improves performance significantly for large inputs.

□ The logic is modular and reusable using a function.

□ Time complexity is **O(√n)** instead of **O(n)**.

Task-3

Prompt: write a code to print fibonacci series up to n terms without function

Code :

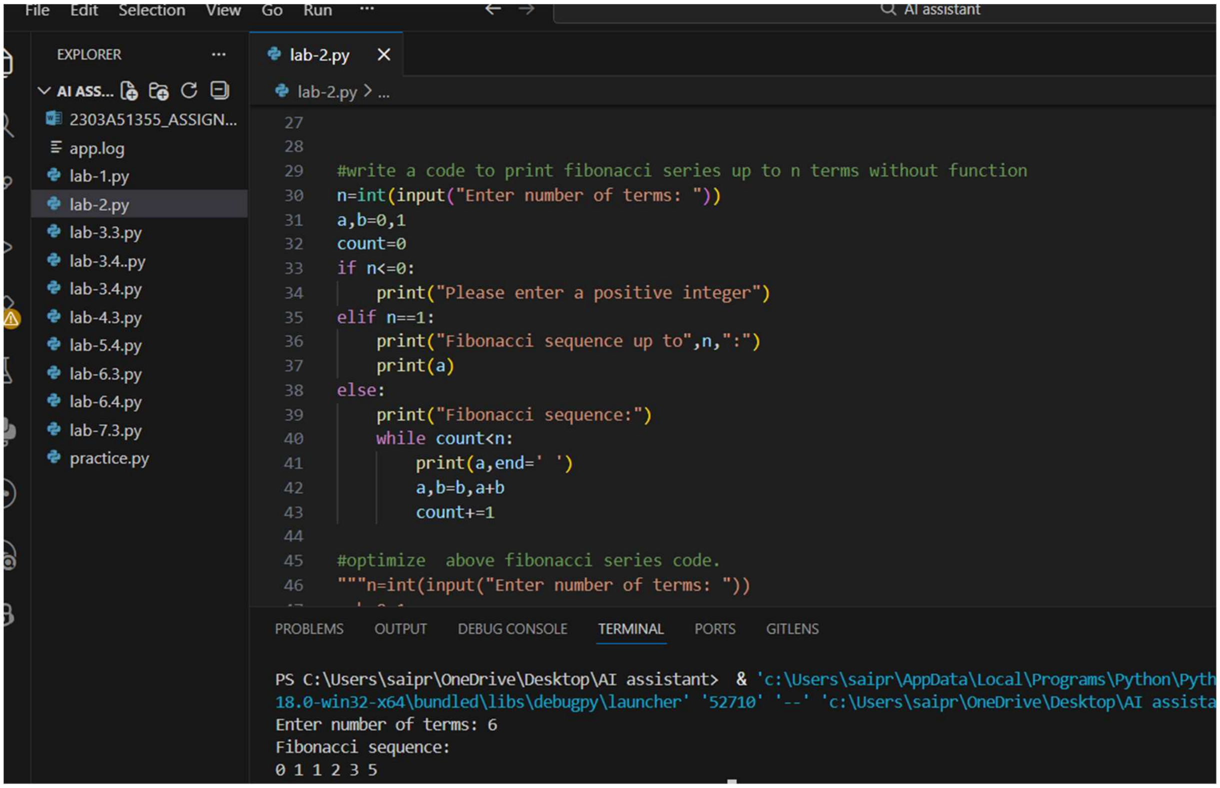
n=int(input("Enter number of

terms: "))

a,b=0,1

```
count=0
if n<=0:
    print("Please enter a positive
integer")
elif n==1:
    print("Fibonacci sequence up
to",n,":")
    print(a)
else:
    print("Fibonacci sequence:")
    while count<n:
        print(a,end=' ')
        a,b=b,a+b
        count+=1
```

Output:



```python
#write a code to print fibonacci series up to n terms without function
n=int(input("Enter number of terms: "))
a,b=0,1
count=0
if n<=0:
    print("Please enter a positive integer")
elif n==1:
    print("Fibonacci sequence up to",n,":")
    print(a)
else:
    print("Fibonacci sequence:")
    while count<n:
        print(a,end=' ')
        a,b=b,a+b
        count+=1

#optimize  above fibonacci series code.
"""n=int(input("Enter number of terms: "))
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\Python\Pyth
18.0-win32-x64\bundled\libs\debugpy\launcher' '52710' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assista
Enter number of terms: 6
Fibonacci sequence:
0 1 1 2 3 5
```

Code Analysis :

□ The program generates Fibonacci numbers using iteration.

□ It handles edge cases like zero or negative input.

□ The while loop prints terms until the count reaches n.

□ Variables a and b are updated in each iteration.

□ It is simple but slightly verbose in structure.

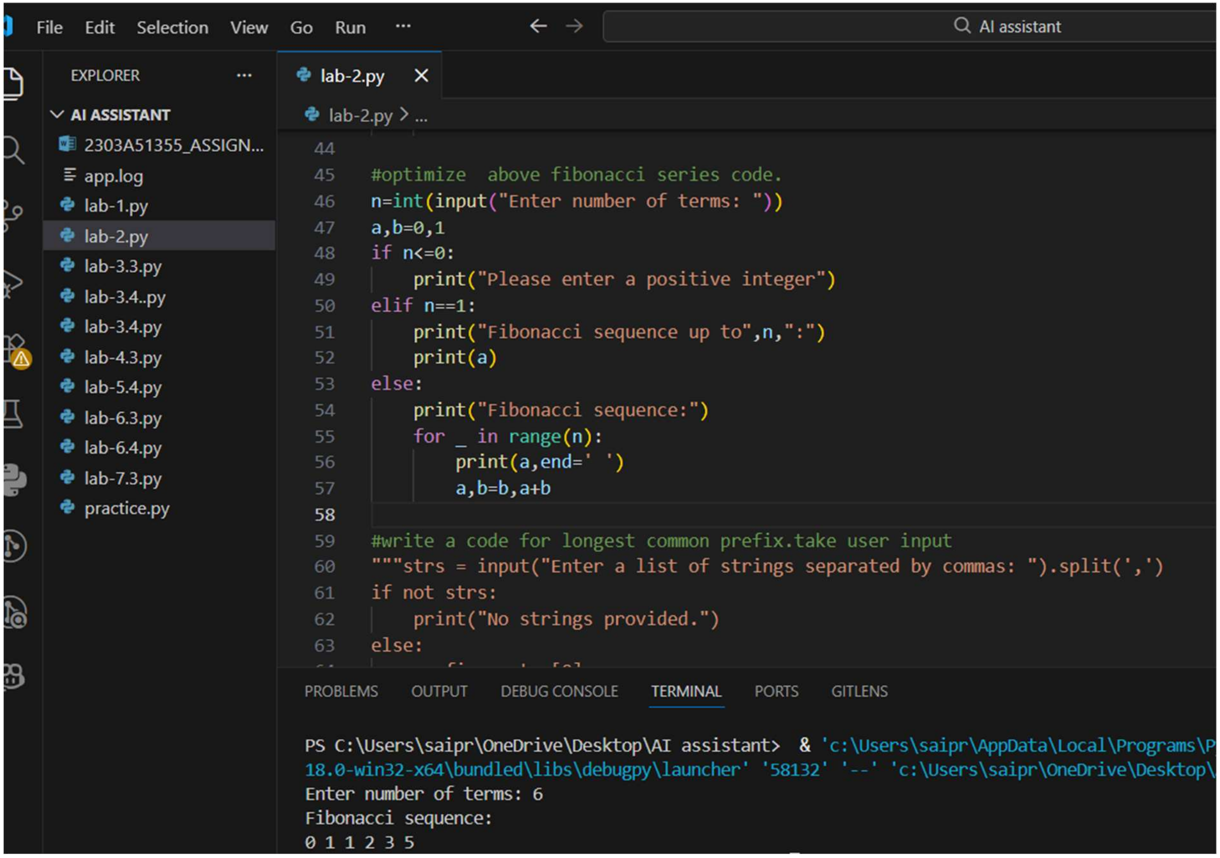
Task-4

Prompt: optimize above fibonacci series code.

Code :

```
n=int(input("Enter number of terms: "))
a,b=0,1
if n<=0:
   print("Please enter a positive integer")
elif n==1:
   print("Fibonacci sequence up to",n,":")
   print(a)
else:
   print("Fibonacci sequence:")
   for _ in range(n):
       print(a,end=' ')
       a,b=b,a+b
```

Output:



```python

#optimize  above fibonacci series code.
n=int(input("Enter number of terms: "))
a,b=0,1
if n<=0:
    print("Please enter a positive integer")
elif n==1:
    print("Fibonacci sequence up to",n,":")
    print(a)
else:
    print("Fibonacci sequence:")
    for _ in range(n):
        print(a,end=' ')
        a,b=b,a+b

#write a code for longest common prefix.take user input
"""strs = input("Enter a list of strings separated by commas: ").split(',')
if not strs:
    print("No strings provided.")
else:
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\P
18.0-win32-x64\bundled\libs\debugpy\launcher' '58132' '--' 'c:\Users\saipr\OneDrive\Desktop\
Enter number of terms: 6
Fibonacci sequence:
0 1 1 2 3 5
```

Code Analysis :

□ This version uses a for-loop, making the code more concise.

□ The logic is cleaner and easier to read.

□ It still maintains correct Fibonacci sequence generation.

□ Reduces unnecessary variables like count.

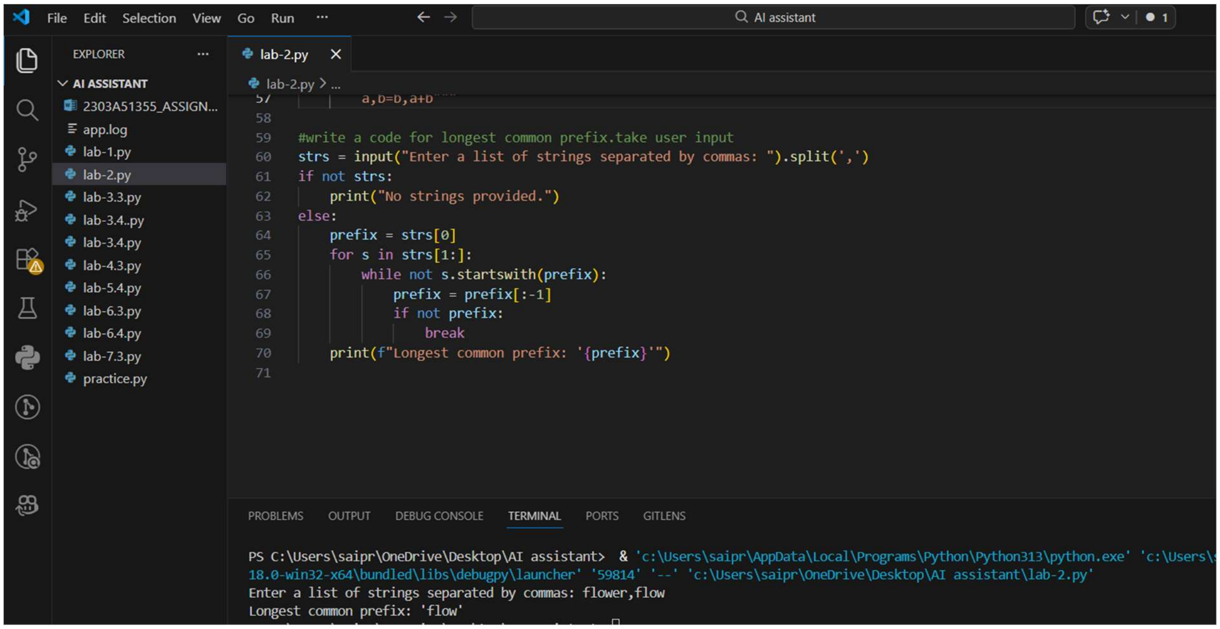
□ Improves readability and efficiency of execution.

Task-5

Prompt: write a code for longest common prefix.take user input.

```
strs = input("Enter a list of strings separated by commas: ").split(',')
if not strs:
    print("No strings provided.")
else:
    prefix = strs[0]
    for s in strs[1:]:
        while not s.startswith(prefix):
            prefix = prefix[:-1]
            if not prefix:
                break
    print(f"Longest common prefix: '{prefix}'")
```

Output :



```
            a,b=b,a+b

#write a code for longest common prefix.take user input
strs = input("Enter a list of strings separated by commas: ").split(',')
if not strs:
    print("No strings provided.")
else:
    prefix = strs[0]
    for s in strs[1:]:
        while not s.startswith(prefix):
            prefix = prefix[:-1]
            if not prefix:
                break
    print(f"Longest common prefix: '{prefix}'")
```

```
PS C:\Users\saipr\OneDrive\Desktop\AI assistant>  & 'c:\Users\saipr\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\s
18.0-win32-x64\bundled\libs\debugpy\launcher' '59814' '--' 'c:\Users\saipr\OneDrive\Desktop\AI assistant\lab-2.py'
Enter a list of strings separated by commas: flower,flow
Longest common prefix: 'flow'
```

Code Analysis :

- The program takes multiple strings separated by commas.
- It assumes the first string as the initial prefix.
- The prefix is reduced until all strings start with it.
- If no common prefix exists, it outputs an empty string.
- Time complexity depends on the number and length of strings.