

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 :

The screenshot shows the VS Code interface with the file 'lab-2.py' open in the editor. The code checks if a number is prime by iterating from 2 to n-1. If any divisor is found, it prints that the number is not prime and breaks the loop. Otherwise, it prints that the number is prime. The terminal below shows the execution of the program and its output for the input '6'.

```
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.python.debugpy-2023.8.0-win32-x64/bundled/lib/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 :

```
File Edit Selection View Go Run ... < > AI assistant
EXPLORER ... lab-2.py X
AI ASSISTANT 2303A51355_ASSIGN...
app.log
lab-1.py
lab-2.py
lab-3.py
lab-3.4.py
lab-3.4.py
lab-4.3.py
lab-5.4.py
lab-6.3.py
lab-6.4.py
lab-7.3.py
practice.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
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 \sqrt{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 $O(\sqrt{n})$ 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:

The screenshot shows a dark-themed interface of VS Code. In the Explorer sidebar, several Python files are listed under a folder named 'AI ASS...'. The file 'lab-2.py' is currently selected. The main editor area displays the following code:

```
27
28
29 #write a code to print fibonacci series up to n terms without function
30 n=int(input("Enter number of terms: "))
31 a,b=0,1
32 count=0
33 if n<=0:
34     print("Please enter a positive integer")
35 elif n==1:
36     print("Fibonacci sequence up to",n,:)
37     print(a)
38 else:
39     print("Fibonacci sequence:")
40     while count<n:
41         print(a,end=' ')
42         a,b=b,a+b
43         count+=1
44
45 #optimize above fibonacci series code.
46 """n=int(input("Enter number of terms: "))
47 """
48 PS C:\Users\saipr\OneDrive\Desktop\AI assistant> & 'c:\Users\saipr\AppData\Local\Programs\Python\Python
49 18.0-win32-x64\bundled\libs\debugpy\launcher' '52710' --- 'c:\Users\saipr\OneDrive\Desktop\AI assista
50 Enter number of terms: 6
51 Fibonacci sequence:
52 0 1 1 2 3 5
```

The terminal below the editor shows the command to run the script and the resulting output for 6 terms.

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:

```

File Edit Selection View Go Run ...
EXPLORER ... lab-2.py X
AI ASSISTANT lab-2.py > ...
44
45 #optimize above fibonacci series code.
46 n=int(input("Enter number of terms: "))
47 a,b=0,1
48 if n<0:
49     print("Please enter a positive integer")
50 elif n==1:
51     print("Fibonacci sequence up to",n,:)
52     print(a)
53 else:
54     print("Fibonacci sequence:")
55     for _ in range(n):
56         print(a,end=' ')
57         a,b=b,a+b
58
59 #write a code for longest common prefix.take user input
60 """strs = input("Enter a list of strings separated by commas: ").split(',')
61 if not strs:
62     print("No strings provided.")
63 else:
64     """
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
PS C:\Users\saipr\OneDrive\Desktop\AI assistant> & 'c:\Users\saipr\AppData\Local\Programs\Python\3.8.0-win32-x64\bundled\libs\debugpy\launcher' '58132' '--' 'c:\Users\saipr\OneDrive\Desktop\lab-2.py'
Enter number of terms: 6
Fibonacci sequence:
0 1 1 2 3 5

```

Code Analysis :

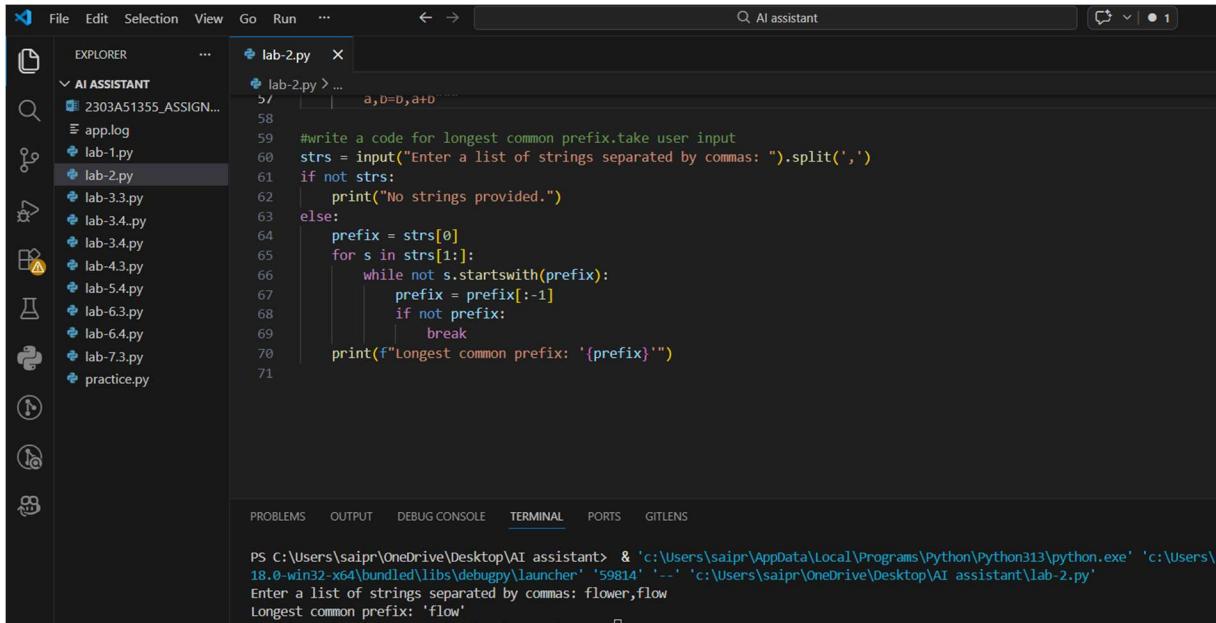
- This version uses a for-loop, making the code more concise.
- The logic is cleaner and easier to read.
- It still maintains correct Fibonacci sequence generation.
- Reduces unnecessary variables like count.
- Improves readability and efficiency of execution.

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 :



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows files in the workspace, including `AI ASSISTANT`, `app.log`, `lab-1.py`, `lab-2.py` (selected), `lab-3.py`, `lab-3.4.py`, `lab-4.3.py`, `lab-5.4.py`, `lab-6.3.py`, `lab-6.4.py`, `lab-7.3.py`, and `practice.py`.
- Code Editor:** Displays the Python script `lab-2.py` with the provided code.
- Terminal:** Shows the command-line output of running the script. It includes the path to the AI assistant executable, the command to run the script, and the user input followed by the output: "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.