

LAB ASSIGNMENT:01

NAME: SANIYA

ROLL NO: 2403A510E7

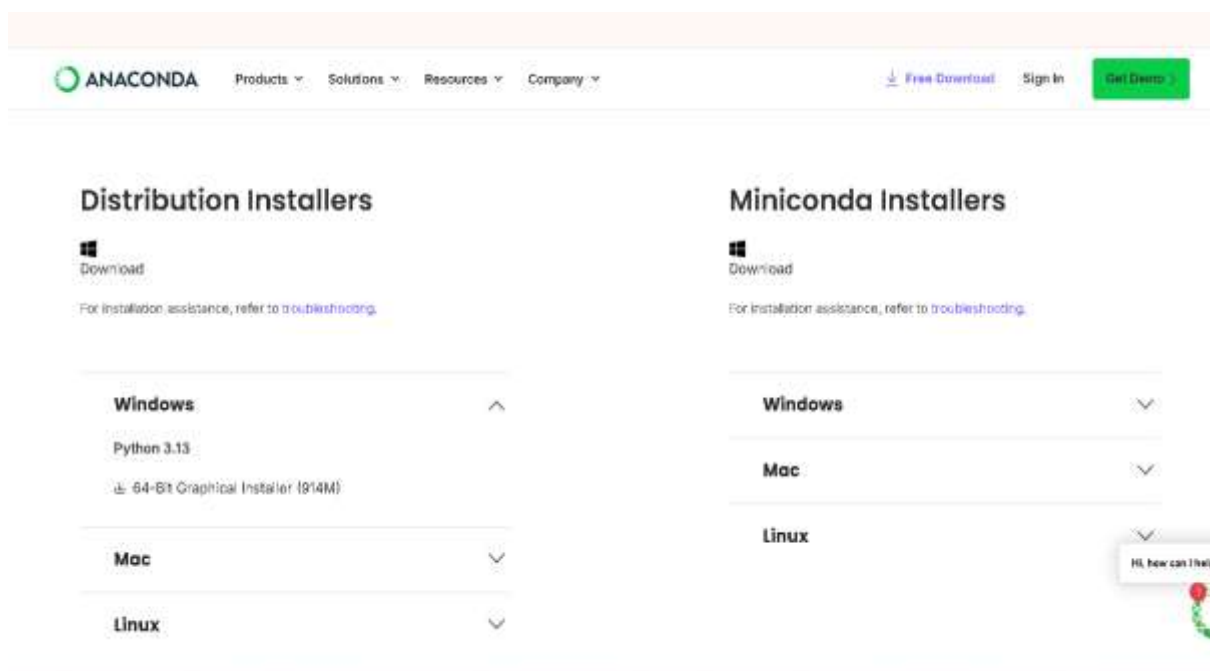
BATCH: 05

BRANCH: CSE

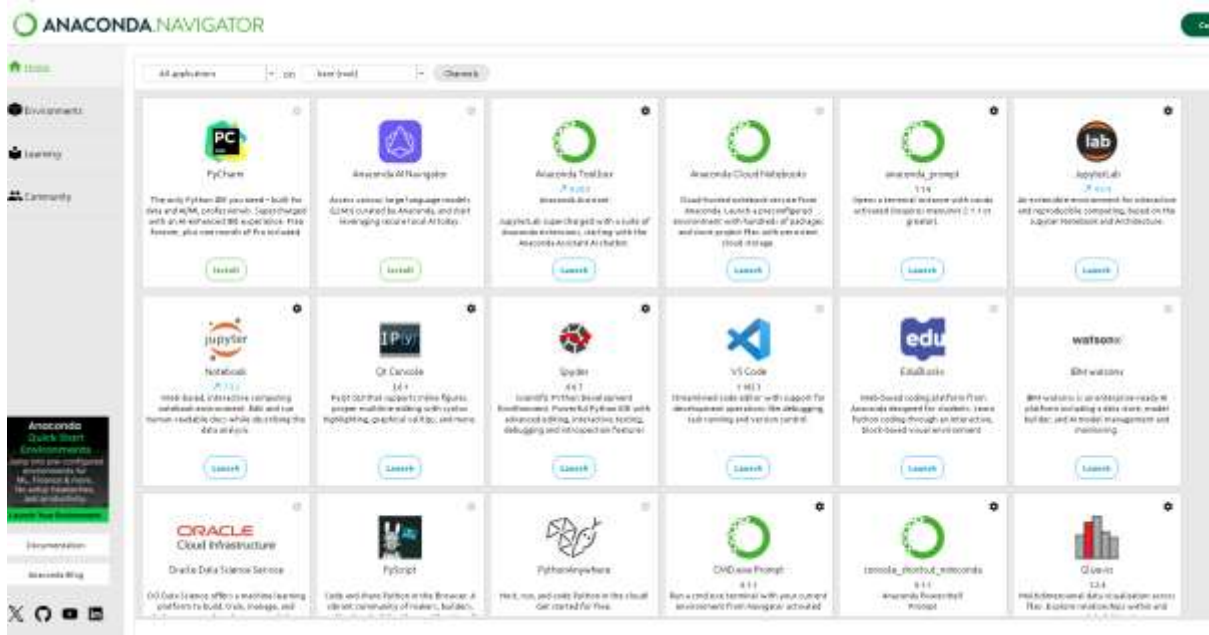
SUB: AI ASSISTED CODING

TASK - 01

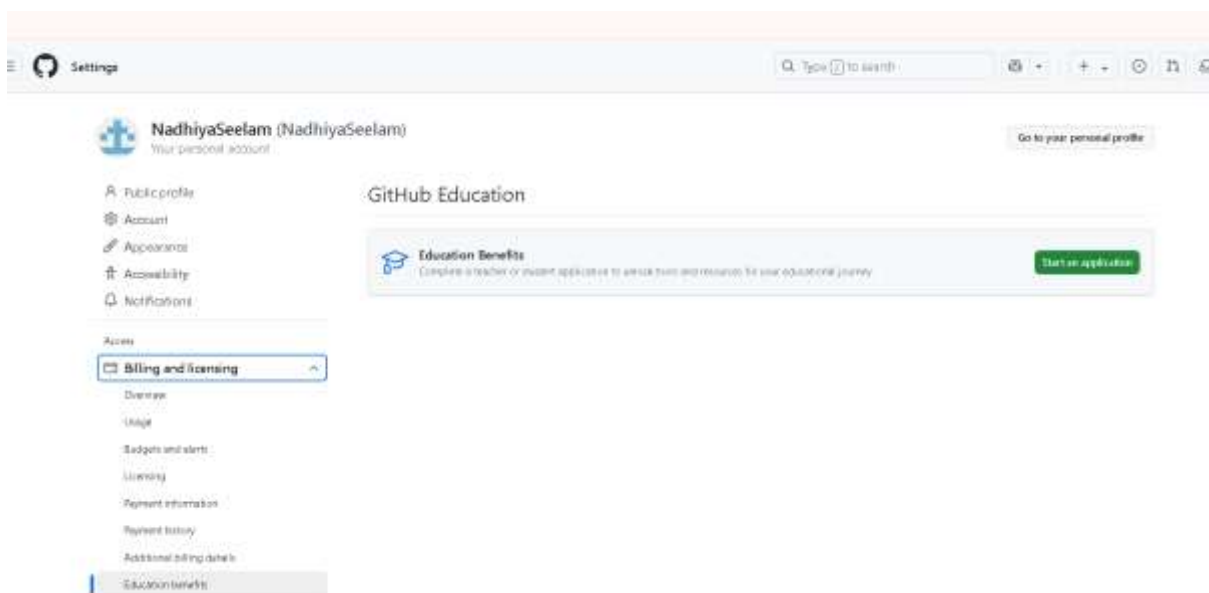
Screen shot-1



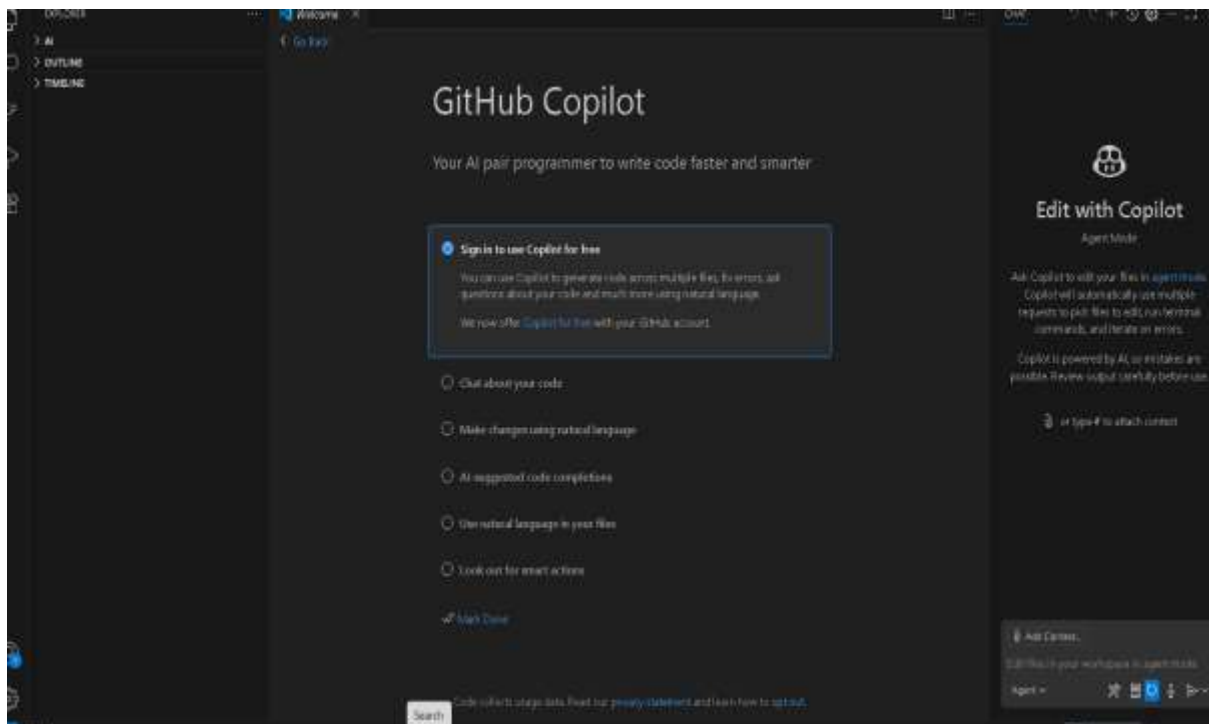
Screen shot-2



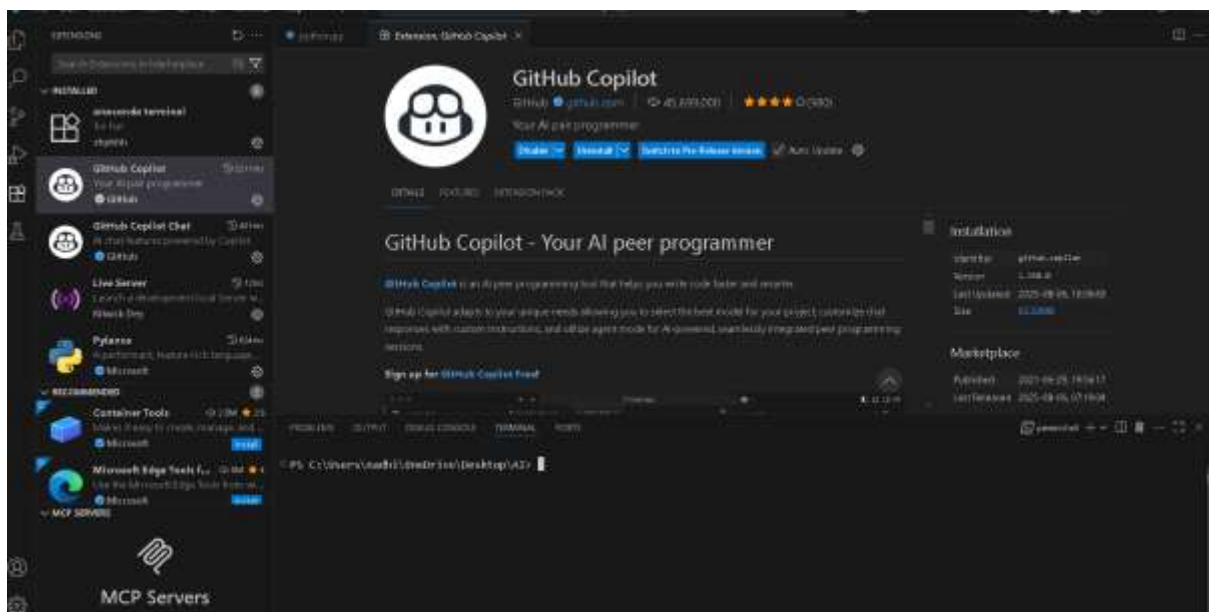
Screen shot-3



Screen shot-4



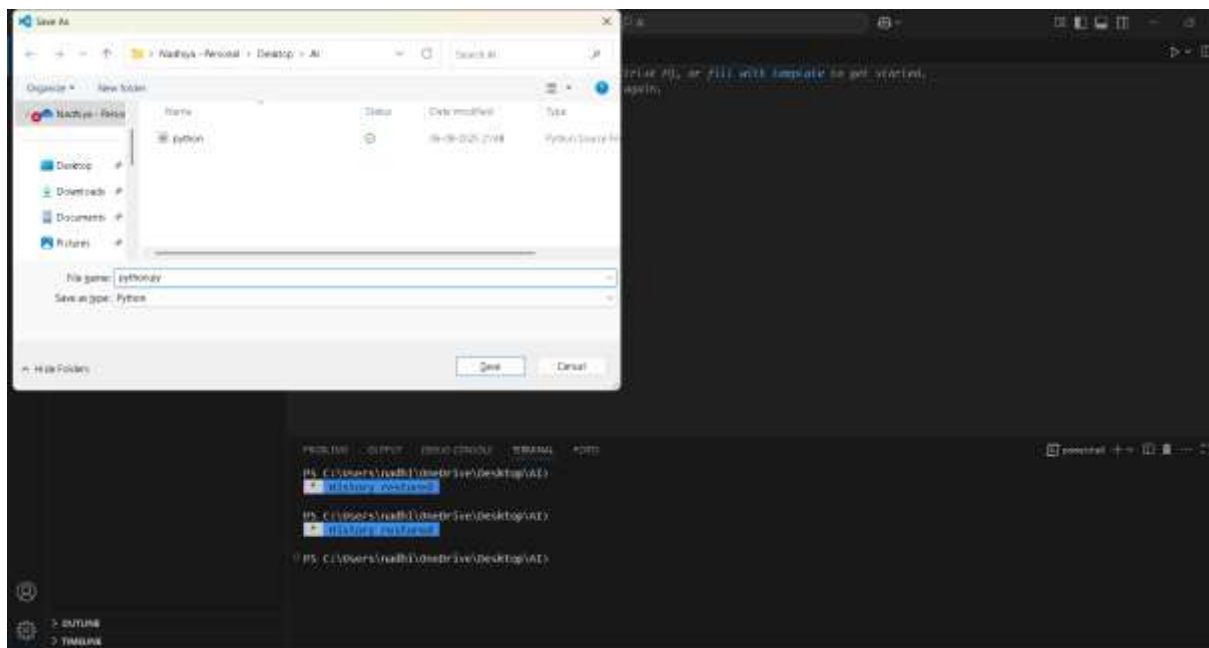
Screen shot-5



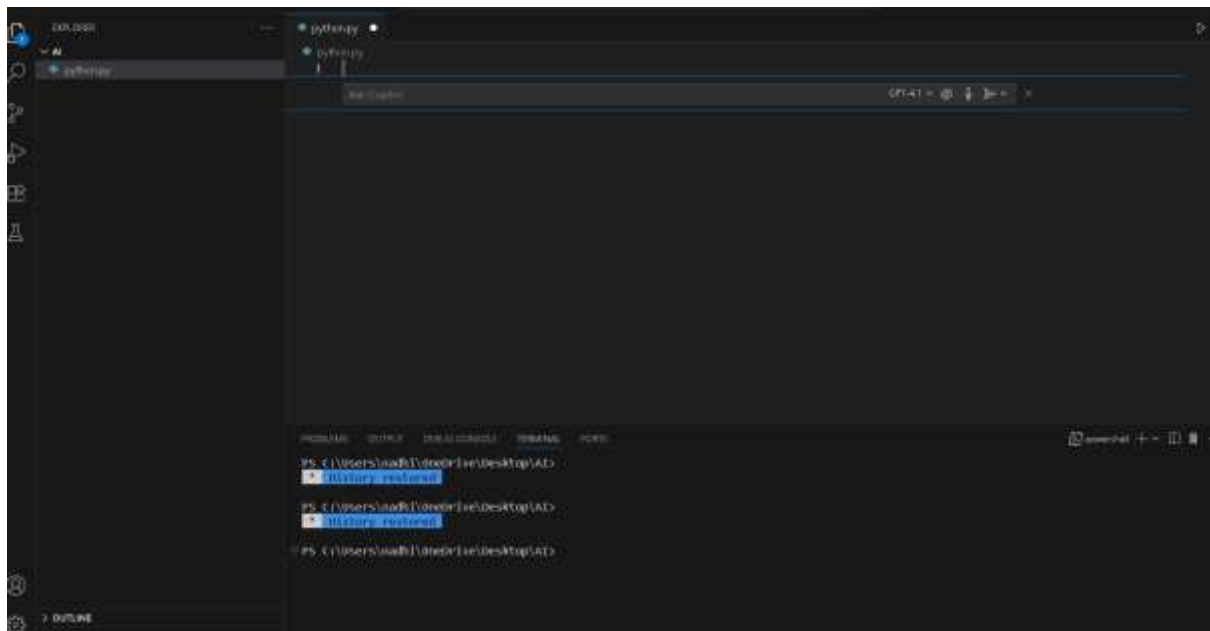
Screen shot-6



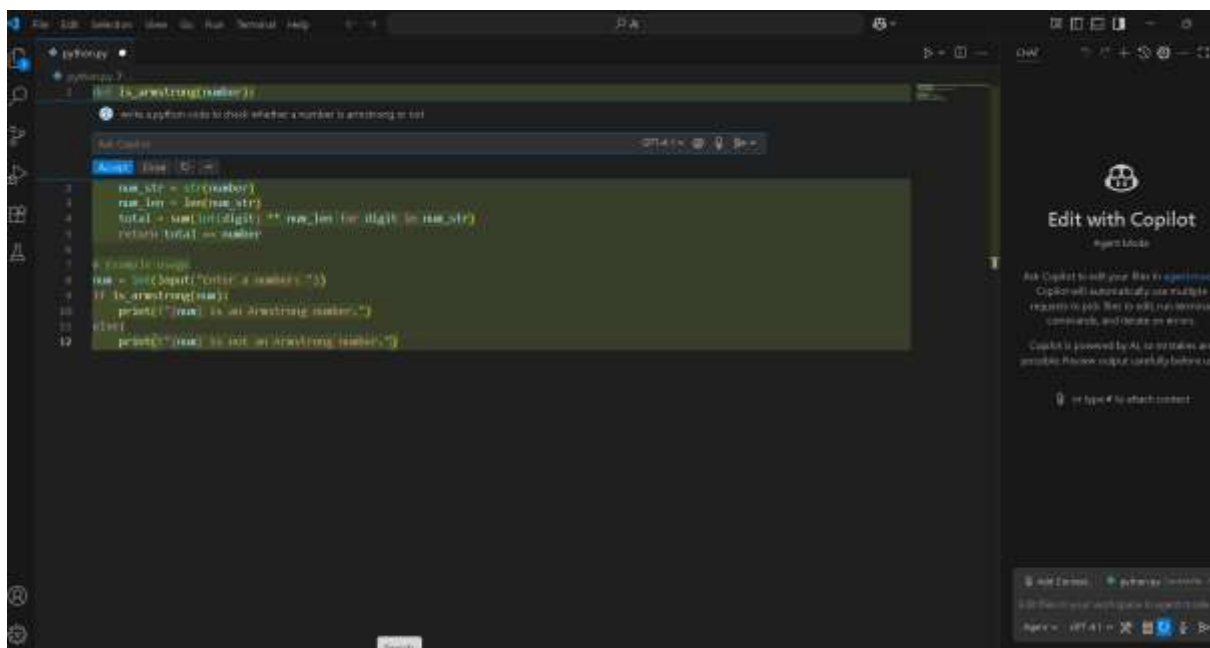
Screen shot-7



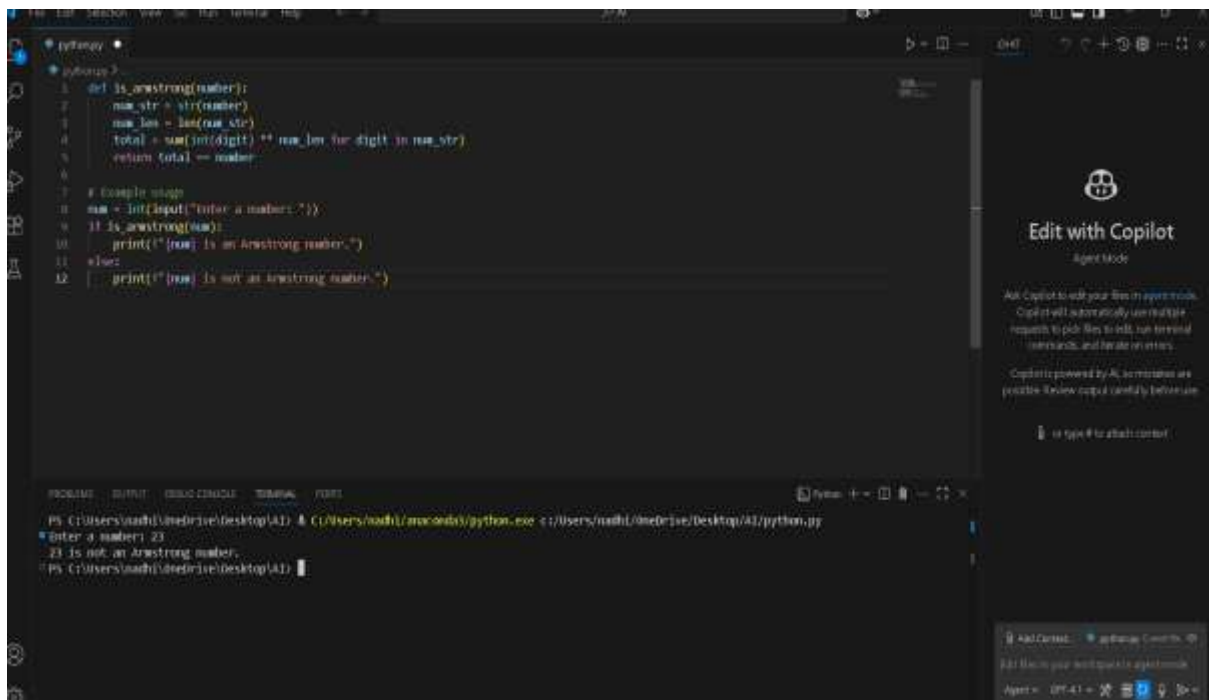
Screen shot-8



Screen shot-9



Screen shot-9

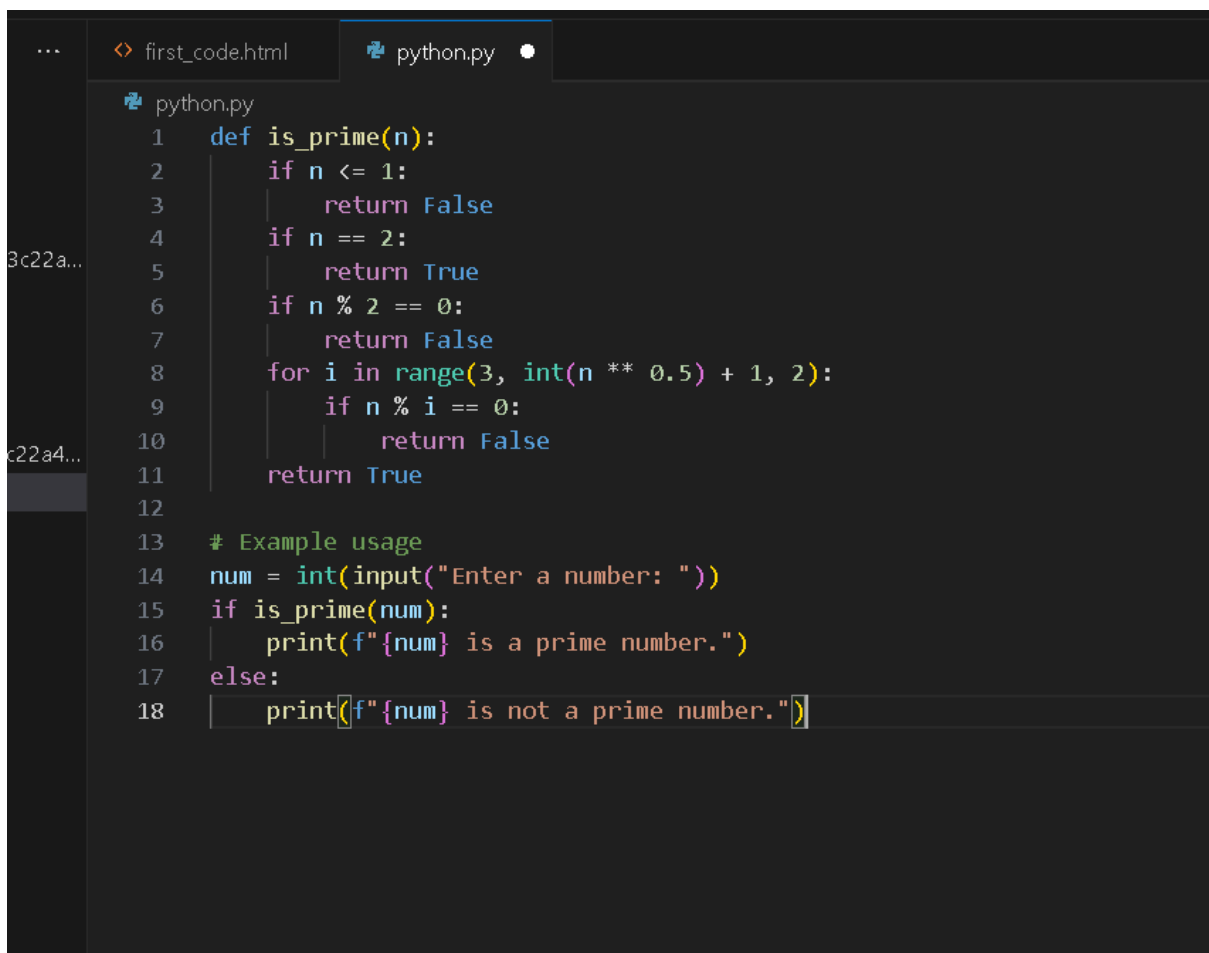


```
1 def is_armstrong(number):
2     num_str = str(number)
3     num_len = len(num_str)
4     total = sum(int(digit) ** num_len for digit in num_str)
5     return total == number
6
7 # Example usage
8 num = int(input("Enter a number: "))
9 if is_armstrong(num):
10     print(f"{num} is an Armstrong number.")
11 else:
12     print(f"{num} is not an Armstrong number.")
```

PS C:\Users\nadh\OneDrive\Desktop\AI> C:\Users\nadh\anaconda\python.exe c:/Users/nadh/OneDrive/Desktop/AI/python.py
Enter a number: 23
23 is not an Armstrong number.
PS C:\Users\nadh\OneDrive\Desktop\AI>

TASK – 02

Prompt : Write a python code to check whether a number is prime or not.



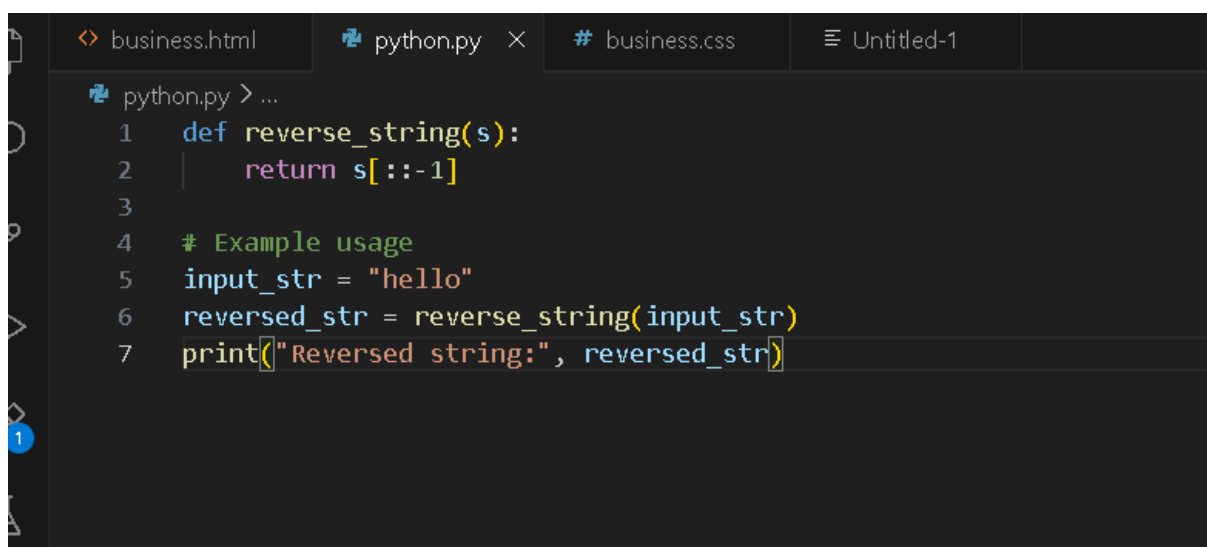
```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     if n == 2:
5         return True
6     if n % 2 == 0:
7         return False
8     for i in range(3, int(n ** 0.5) + 1, 2):
9         if n % i == 0:
10            return False
11    return True
12
13 # Example usage
14 num = int(input("Enter a number: "))
15 if is_prime(num):
16     print(f"{num} is a prime number.")
17 else:
18     print(f"{num} is not a prime number.")
```

Output: 2 is prime number

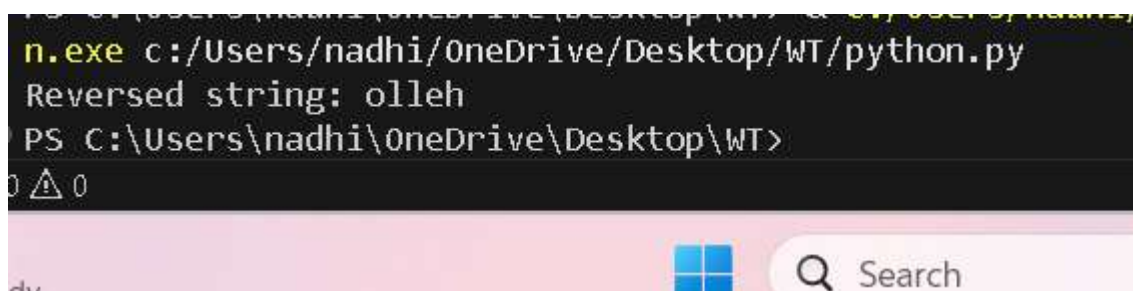
Explanation:

- The code checks if a given number is prime.
- A prime number is a number greater than 1 that has no divisors other than 1 and itself.
- The code usually:
 - Returns False if the number is less than or equal to 1.
 - Loops from 2 up to the square root of the number.
 - If the number is divisible by any of these, it returns False.
 - If no divisors are found, it returns True.

Task-3:



```
python.py > ...
1  def reverse_string(s):
2      return s[::-1]
3
4  # Example usage
5  input_str = "hello"
6  reversed_str = reverse_string(input_str)
7  print("Reversed string:", reversed_str)
```



```
python.exe c:/Users/nadhi/OneDrive/Desktop/WT/python.py
Reversed string: olleh
PS C:\Users\nadhi\OneDrive\Desktop\WT>
```

Explanation:

- The function `reverse_string(s)` takes a string `s` as input and returns its reverse using slicing (`s[::-1]`).
- The example usage sets `input_str` to "hello".
- It calls `reverse_string(input_str)`, which returns "olleh", and stores it in `reversed_str`.
- Finally, it prints Reversed string: olleh to the console

Task-04:

```
python.py > ...
1  # Recursive version of factorial
2  def factorial_recursive(n):
3      """
4      Calculate factorial of n recursively.
5      """
6      if n == 0 or n == 1:
7          return 1
8      else:
9          return n * factorial_recursive(n - 1)
10
11 # Iterative version of factorial
12 def factorial_iterative(n):
13     """
14     Calculate factorial of n iteratively.
15     """
16     result = 1
17     for i in range(2, n + 1):
18         result *= i
19     return result
20
21 # Example usage
22 if __name__ == "__main__":
23     num = 5
24     print("Recursive:", factorial_recursive(num)) # Output: 120
25     print("Iterative:", factorial_iterative(num)) # Output: 120
```

```
PS C:\Users\nadhi\OneDrive\Desktop\WT> & C:/Users/nadhi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nadhi/OneDrive/Desktop/WT/python.py
Recursive: 120
Iterative: 120
```

Explanation:

- This function calculates the factorial of `n` using recursion.
 - If `n` is 0 or 1, it returns 1 (base case).
 - Otherwise, it returns `n * factorial_recursive(n - 1)`.

- **factorial_iterative(n):**

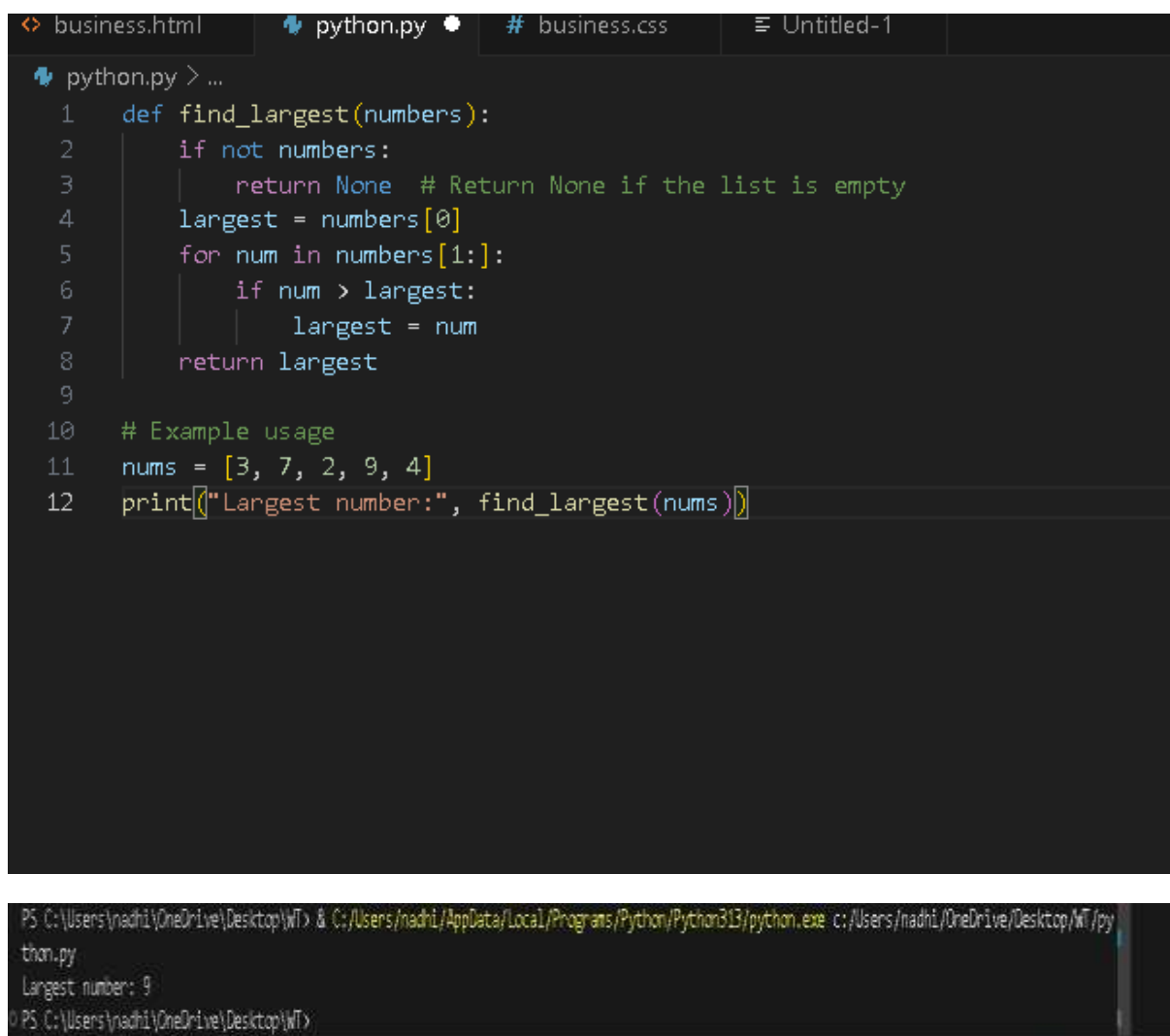
This function calculates the factorial of n using a loop.

- It initializes result to 1.
- Then multiplies result by each number from 2 up to n.

- **Example usage:**

- If the script is run directly, it sets num = 5.
- It prints the factorial of 5 using both the recursive and iterative functions.
- Both methods output 120.

Task-05:



The image shows a code editor with a dark theme. The top bar has tabs for 'business.html', 'python.py', 'business.css', and 'Untitled-1'. The 'python.py' tab is active, showing a Python script. The script defines a function 'find_largest' that takes a list of numbers and returns the largest one. It then uses this function on a list 'nums' and prints the result. Below the code editor is a terminal window showing the command to run the script and the output 'Largest number: 9'.

```
<> business.html python.py • # business.css ≡ Untitled-1
python.py > ...
1 def find_largest(numbers):
2     if not numbers:
3         return None # Return None if the list is empty
4     largest = numbers[0]
5     for num in numbers[1:]:
6         if num > largest:
7             largest = num
8     return largest
9
10 # Example usage
11 nums = [3, 7, 2, 9, 4]
12 print(["Largest number:", find_largest(nums)])

PS C:\Users\nadhi\OneDrive\Desktop\WT> & C:\Users\nadhi\AppData\Local\Programs\Python\Python313\python.exe c:\Users\nadhi\OneDrive\Desktop\WT\py
thon.py
Largest number: 9
PS C:\Users\nadhi\OneDrive\Desktop\WT>
```

Explanation:

- **find_largest(numbers):**

This function takes a list of numbers and returns the largest value.

- If the list is empty, it returns None.
- It starts by assuming the first number is the largest.
- It then loops through the rest of the list, updating [largest](#) if it finds a bigger number.
- Finally, it returns the largest number found.

- **Example usage:**

- A list [nums = \[3, 7, 2, 9, 4\]](#) is defined.
- The function is called with this list, and the result is printed.
- Output: Largest number: 9