

# Lab Program - 1

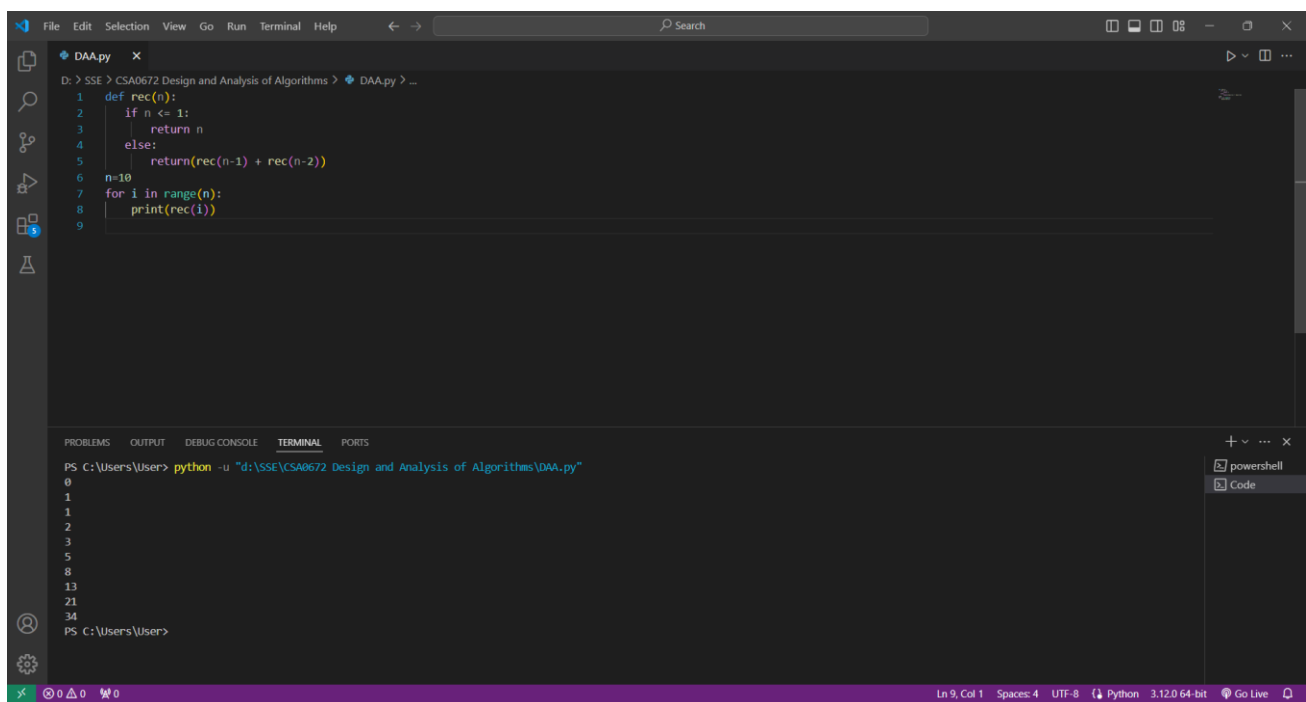
SOORYA – 192311060

1. Write a program to Print Fibonacci Series using recursion.

Code:

```
def rec(n):
    if n <= 1:
        return n
    else:
        return(rec(n-1) + rec(n-2))
n=10
for i in range(n):
    print(rec(i))
```

Screenshot for I/O:

A screenshot of a code editor window. The top pane shows a Python file named 'DAA.py' with the following code:

```
1 def rec(n):
2     if n <= 1:
3         return n
4     else:
5         return(rec(n-1) + rec(n-2))
6 n=10
7 for i in range(n):
8     print(rec(i))
9
```

The bottom pane shows the 'TERMINAL' output, which displays the Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. The terminal prompt is 'PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"'. The status bar at the bottom indicates 'Ln 9, Col 1', 'Spaces: 4', 'UTF-8', 'Python 3.12.0 64-bit', and 'Go Live'.

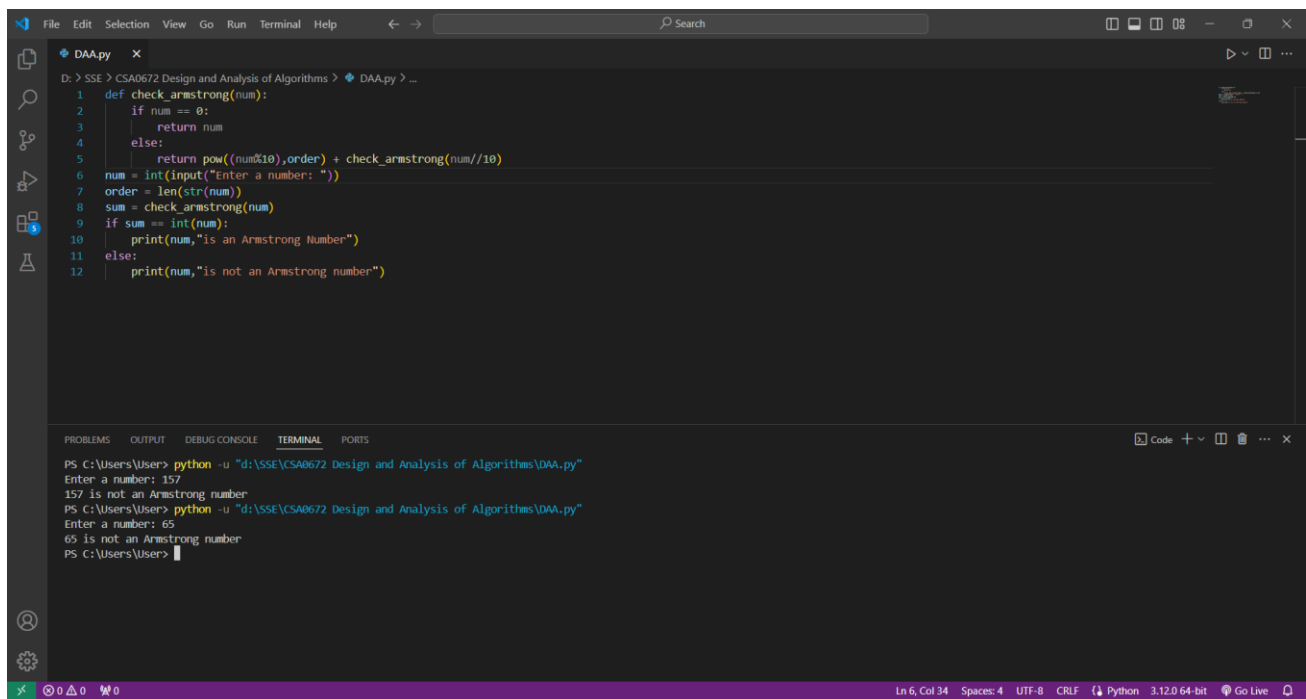
Time Complexity:  $O(n)$

2. Write a program to check the given no is Armstrong or not using recursive function.

### Code:

```
def check_armstrong(num):
    if num == 0:
        return num
    else:
        return pow((num%10),order) + check_armstrong(num//10)
num = int(input("Enter a number: "))
order = len(str(num))
sum = check_armstrong(num)
if sum == int(num):
    print(num,"is an Armstrong Number")
else:
    print(num,"is not an Armstrong number")
```

### Screenshot for I/O:



The screenshot displays a code editor with a dark theme. The top section shows the Python code for checking Armstrong numbers. The bottom section, labeled 'TERMINAL', shows the program's execution. It prompts the user to enter a number. In the first run, '157' is entered, and the output is '157 is not an Armstrong number'. In the second run, '65' is entered, and the output is '65 is not an Armstrong number'. The status bar at the bottom indicates the file is 'DAA.py', line 6, column 34, with 4 spaces, using UTF-8 encoding and CRLF line endings, and is running Python 3.12.0 64-bit.

```
D:\SSE > CSA0672 Design and Analysis of Algorithms > DAA.py > ...
1 def check_armstrong(num):
2     if num == 0:
3         return num
4     else:
5         return pow((num%10),order) + check_armstrong(num//10)
6 num = int(input("Enter a number: "))
7 order = len(str(num))
8 sum = check_armstrong(num)
9 if sum == int(num):
10     print(num,"is an Armstrong Number")
11 else:
12     print(num,"is not an Armstrong number")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"
Enter a number: 157
157 is not an Armstrong number
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"
Enter a number: 65
65 is not an Armstrong number
PS C:\Users\User>
```

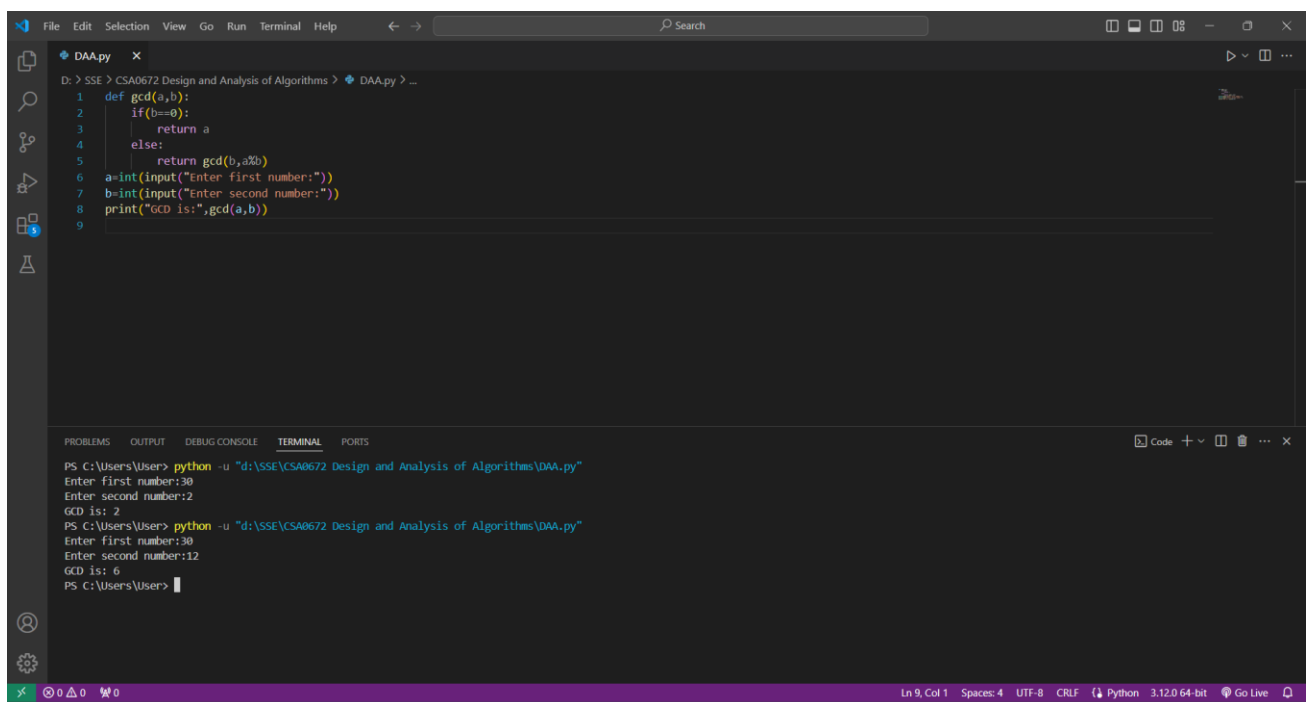
**Time Complexity:** $O(n)$

3. Write a program to find the GCD of two numbers using recursive factorization

**Code:**

```
def gcd(a,b):  
    if(b==0):  
        return a  
    else:  
        return gcd(b,a%b)  
a=int(input("Enter first number:"))  
b=int(input("Enter second number:"))  
print("GCD is:",gcd(a,b))
```

**Screenshot for I/O:**



The screenshot shows a code editor with a file named 'DAA.py'. The code defines a recursive function 'gcd(a,b)' that returns 'a' if 'b' is 0, otherwise it returns 'gcd(b, a%b)'. Below the function, it prompts the user for two numbers and prints the GCD. The terminal output shows two test cases: first with inputs 30 and 12 resulting in GCD 6, and second with inputs 30 and 12 resulting in GCD 6.

```
D:\SSE\CSA0672 Design and Analysis of Algorithms > DAA.py > ...  
1 def gcd(a,b):  
2     if(b==0):  
3         return a  
4     else:  
5         return gcd(b,a%b)  
6 a=int(input("Enter first number:"))  
7 b=int(input("Enter second number:"))  
8 print("GCD is:",gcd(a,b))  
9  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"  
Enter first number:30  
Enter second number:12  
GCD is: 6  
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"  
Enter first number:30  
Enter second number:12  
GCD is: 6  
PS C:\Users\User>
```

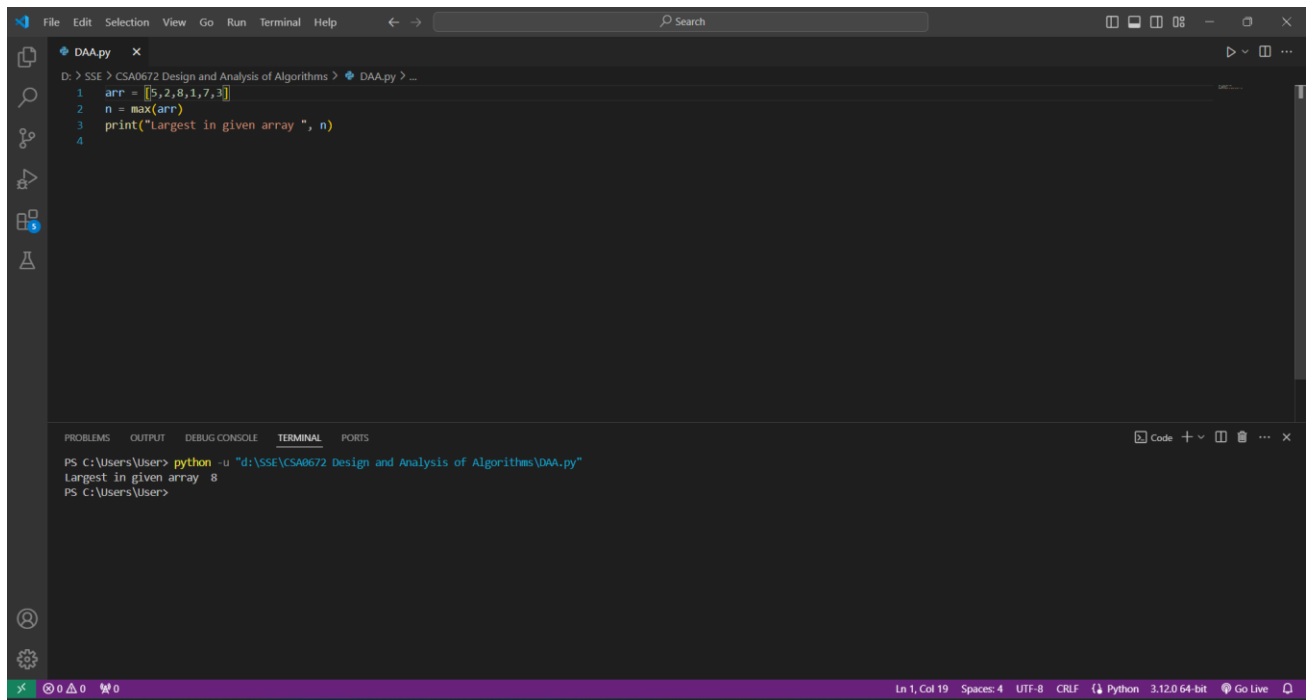
**Time Complexity:  $O(n)$**

4. Write a program to get the largest element of an array.

**Code:**

```
arr = [5,2,8,1,7,3]
n = max(arr)
print("Largest in given array ", n)
```

**Screenshot for I/O:**



The screenshot displays a code editor with a file named 'DAA.py'. The code in the editor is as follows:

```
1 arr = [5,2,8,1,7,3]
2 n = max(arr)
3 print("Largest in given array ", n)
4
```

Below the code editor, the terminal window shows the execution of the program. The command entered is `python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"`, and the output is `Largest in given array 8`.

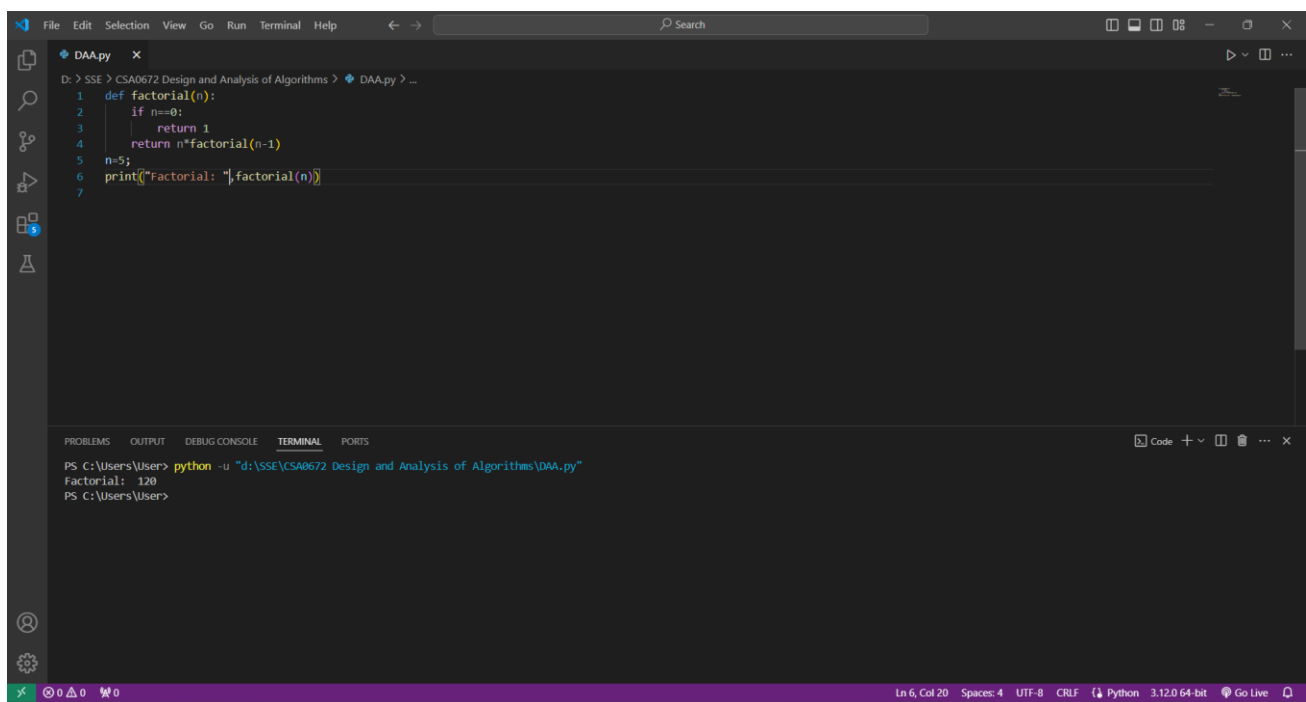
**Time Complexity:  $O(n)$**

5. Write a program to find the Factorial of a number using recursion.

### Code:

```
def factorial(n):  
    if n==0:  
        return 1  
    return n*factorial(n-1)  
n=5;  
print("Factorial: ",factorial(n))
```

### Screenshot for I/O:

A screenshot of a code editor window. The editor has a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The search bar is empty. The file explorer on the left shows a folder named 'DAA.py'. The main editor area displays the following Python code:

```
1 def factorial(n):  
2     if n==0:  
3         return 1  
4     return n*factorial(n-1)  
5 n=5;  
6 print("Factorial: ",factorial(n))  
7
```

The bottom panel shows the 'TERMINAL' tab with the following output:

```
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"  
Factorial: 120  
PS C:\Users\User>
```

The status bar at the bottom indicates 'Ln 6, Col 20', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', '3.12.0 64-bit', and 'Go Live'.

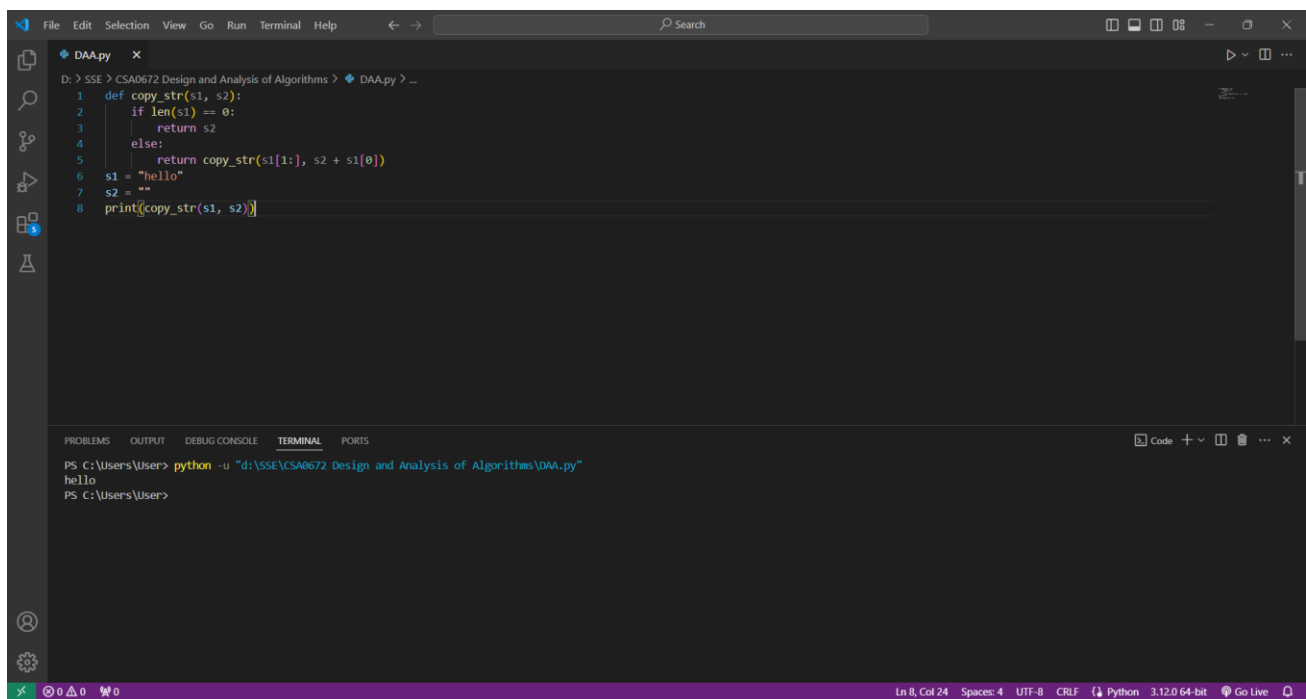
**Time Complexity:  $O(n)$**

6. Write a program for to copy one string to another using recursion

**Code:**

```
def copy_str(s1, s2):  
    if len(s1) == 0:  
        return s2  
    else:  
        return copy_str(s1[1:], s2 + s1[0])  
s1 = "hello"  
s2 = ""  
print(copy_str(s1, s2))
```

**Screenshot for I/O:**

A screenshot of a code editor window titled 'DAA.py'. The editor shows the following Python code:

```
1 def copy_str(s1, s2):  
2     if len(s1) == 0:  
3         return s2  
4     else:  
5         return copy_str(s1[1:], s2 + s1[0])  
6 s1 = "hello"  
7 s2 = ""  
8 print(copy_str(s1, s2))
```

The bottom panel of the editor shows the 'TERMINAL' output:

```
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"  
hello  
PS C:\Users\User>
```

The status bar at the bottom indicates 'Ln 8, Col 24', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', '3.12.0 64-bit', and 'Go Live'.

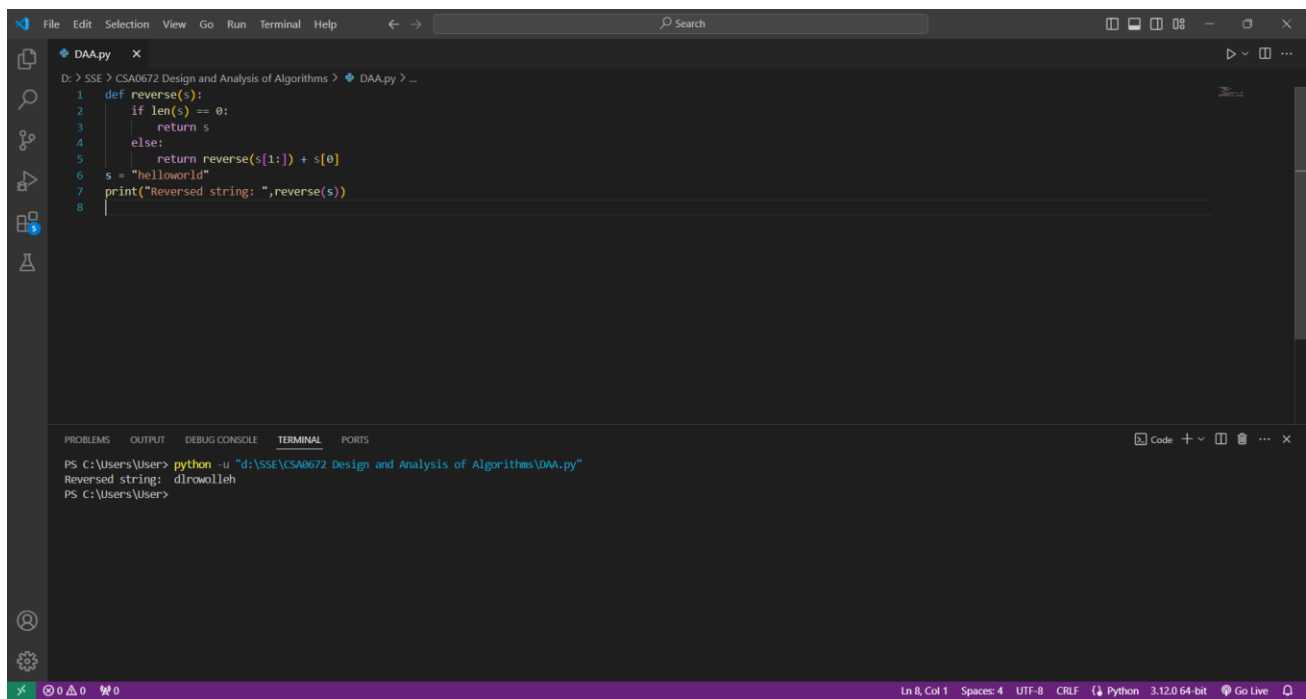
**Time Complexity:  $O(n)$**

7. Write a program to print the reverse of a string using recursion

**Code:**

```
def reverse(s):  
    if len(s) == 0:  
        return s  
    else:  
        return reverse(s[1:]) + s[0]  
  
s = "helloworld"  
print("Reversed string: ",reverse(s))
```

**Screenshot for I/O:**

A screenshot of a code editor window titled 'DAA.py'. The editor shows the following Python code:

```
1 def reverse(s):  
2     if len(s) == 0:  
3         return s  
4     else:  
5         return reverse(s[1:]) + s[0]  
6 s = "helloworld"  
7 print("Reversed string: ",reverse(s))  
8
```

The editor has a sidebar on the left with icons for Explorer, Search, Source Control, Run and Debug, and Test Explorer. At the bottom, there is a terminal window with the following output:

```
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"  
Reversed string: dlrowolleh  
PS C:\Users\User>
```

The status bar at the bottom indicates 'Ln 8, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', '3.12.0 64-bit', and 'Go Live'.

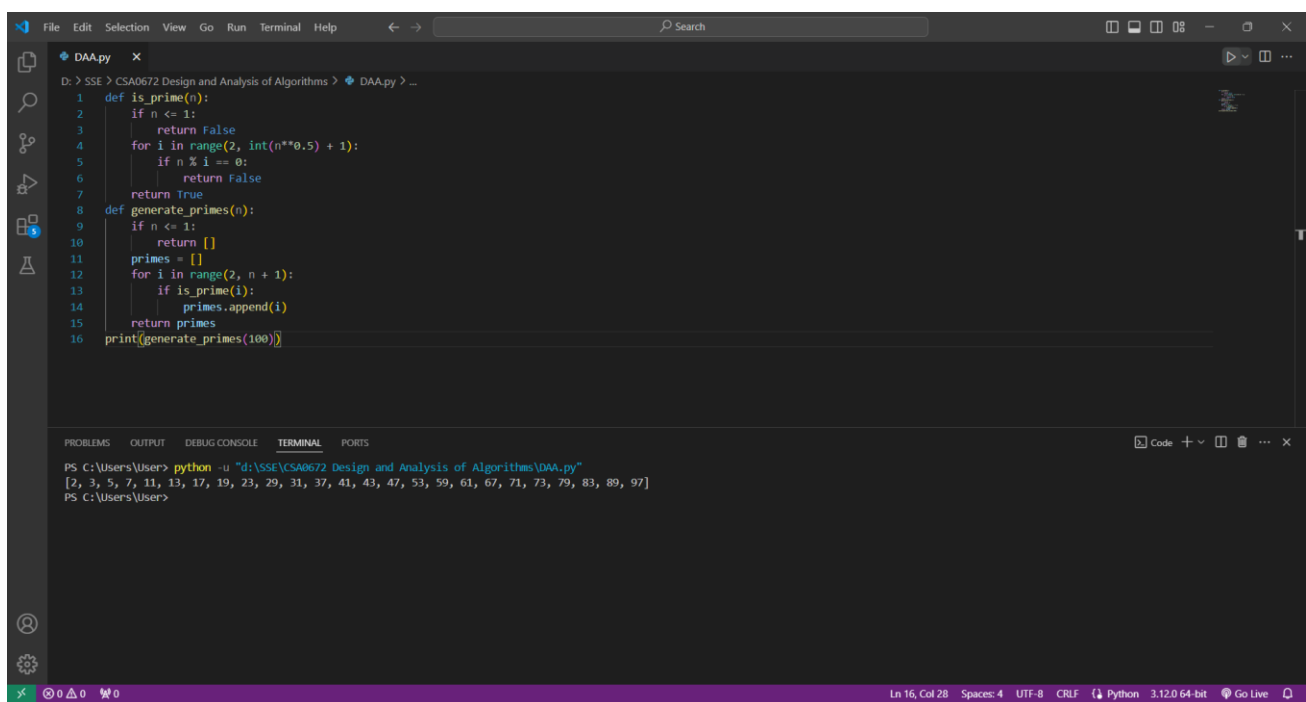
**Time Complexity:  $O(n)$**

8. Write a program to generate all the prime numbers using recursion

**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
def generate_primes(n):
    if n <= 1:
        return []
    primes = []
    for i in range(2, n + 1):
        if is_prime(i):
            primes.append(i)
    return primes
print(generate_primes(100))
```

**Screenshot for I/O:**

A screenshot of a code editor window with a dark theme. The editor shows a Python file named 'DAA.py' with the following code:

```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     for i in range(2, int(n**0.5) + 1):
5         if n % i == 0:
6             return False
7     return True
8 def generate_primes(n):
9     if n <= 1:
10        return []
11    primes = []
12    for i in range(2, n + 1):
13        if is_prime(i):
14            primes.append(i)
15    return primes
16 print(generate_primes(100))
```

The bottom panel of the editor shows the 'TERMINAL' output:

```
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
PS C:\Users\User>
```

The status bar at the bottom indicates 'Ln 16, Col 28', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', '3.12.0 64-bit', and 'Go Live'.



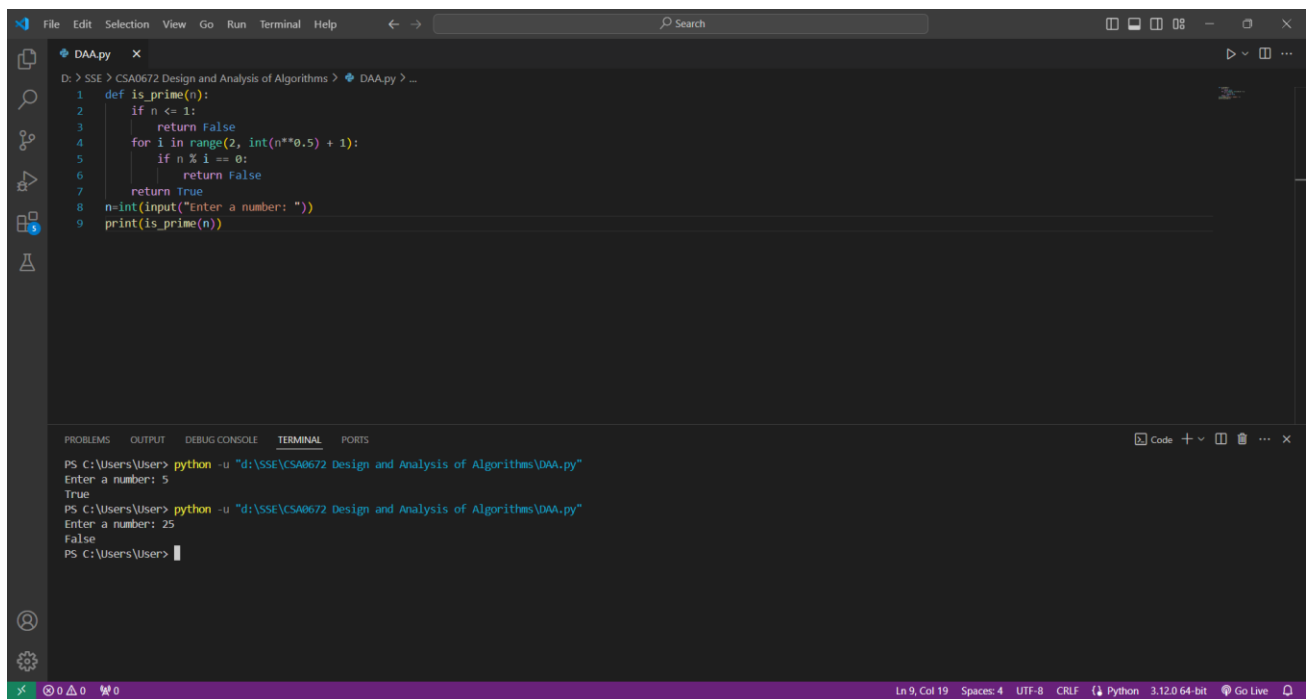
## Time Complexity: $O(n*m)$

9. Write a program to check a number is a prime number or not using recursion.

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: "))
print(is_prime(n))
```

Screenshot for I/O:



The screenshot shows a code editor with a file named `DAA.py` containing the following Python code:

```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     for i in range(2, int(n**0.5) + 1):
5         if n % i == 0:
6             return False
7     return True
8 n=int(input("Enter a number: "))
9 print(is_prime(n))
```

The terminal output at the bottom shows the program being executed twice:

```
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"
Enter a number: 5
True
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"
Enter a number: 25
False
PS C:\Users\User>
```

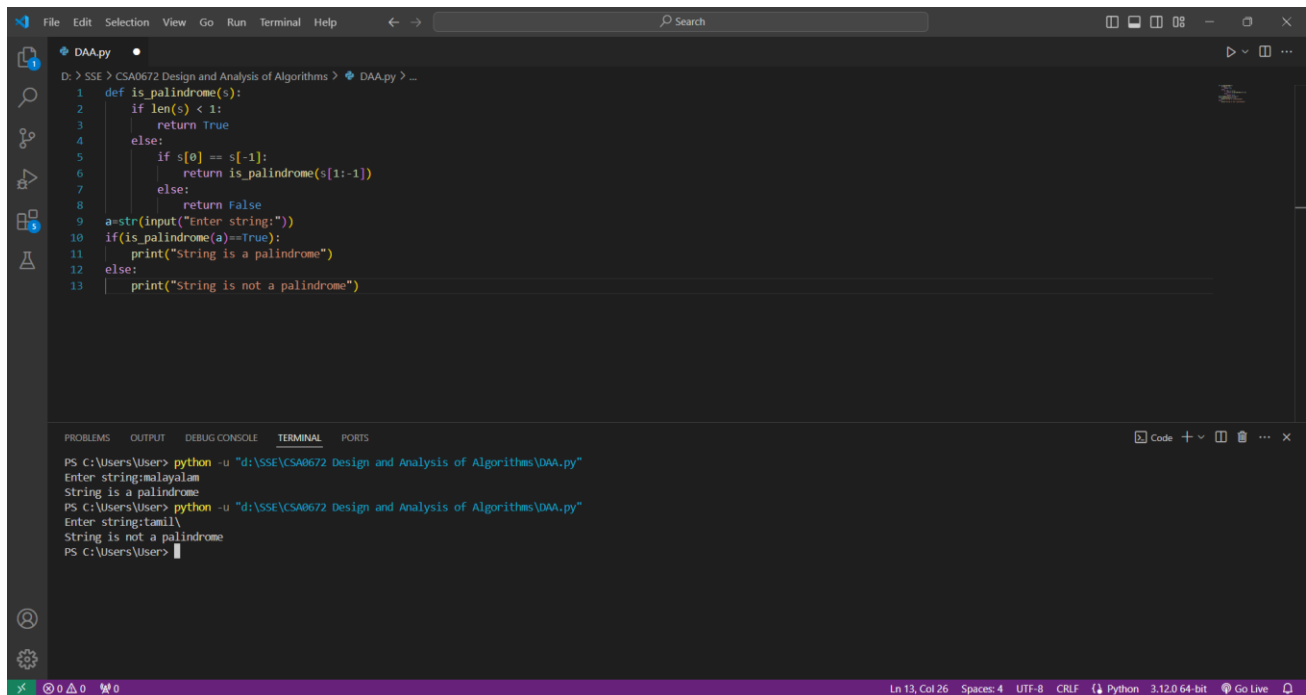
Time Complexity:  $O(n)$

10. Write a program for to check whether a given String is Palindrome or not using recursion.

### Code:

```
def is_palindrome(s):
    if len(s) < 1:
        return True
    else:
        if s[0] == s[-1]:
            return is_palindrome(s[1:-1])
        else:
            return False
a=str(input("Enter string:"))
if(is_palindrome(a)==True):
    print("String is a palindrome")
else:
    print("String is not a palindrome")
```

### Screenshot for I/O:

A screenshot of a Python IDE window titled 'DAA.py'. The editor shows the following code:

```
1 def is_palindrome(s):
2     if len(s) < 1:
3         return True
4     else:
5         if s[0] == s[-1]:
6             return is_palindrome(s[1:-1])
7         else:
8             return False
9 a=str(input("Enter string:"))
10 if(is_palindrome(a)==True):
11     print("String is a palindrome")
12 else:
13     print("String is not a palindrome")
```

The bottom panel shows the 'TERMINAL' output:

```
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"
Enter string:malayalam
String is a palindrome
PS C:\Users\User> python -u "d:\SSE\CSA0672 Design and Analysis of Algorithms\DAA.py"
Enter string:tamil
String is not a palindrome
PS C:\Users\User>
```

**Time Complexity:  $O(n)$**

