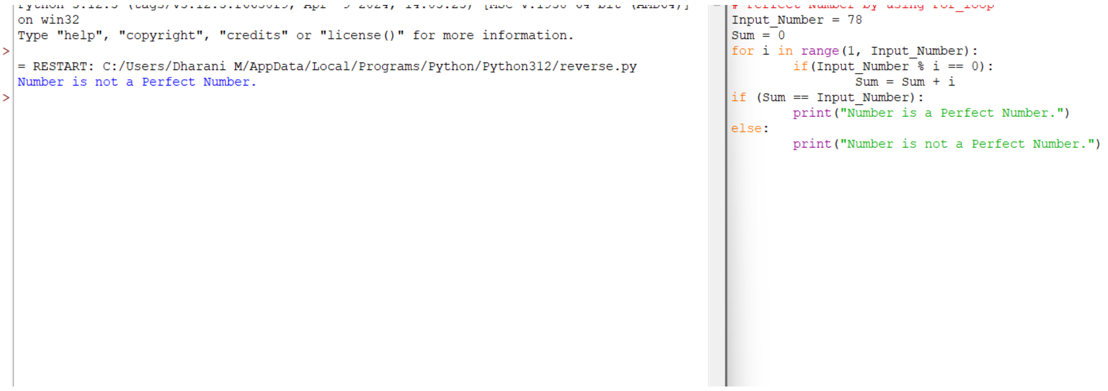
1. Write a program to find the reverse of a given number using recursive.

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/reverse.py
4321
>>>
```

```
def recursum(number,reverse):
  if number==0:
    return reverse
  remainder = int(number%10)
  reverse = (reverse*10)+remainder
  return recursum(int(number/10),reverse)

num = 1234
reverse = 0
print(recursum(num,reverse))
```

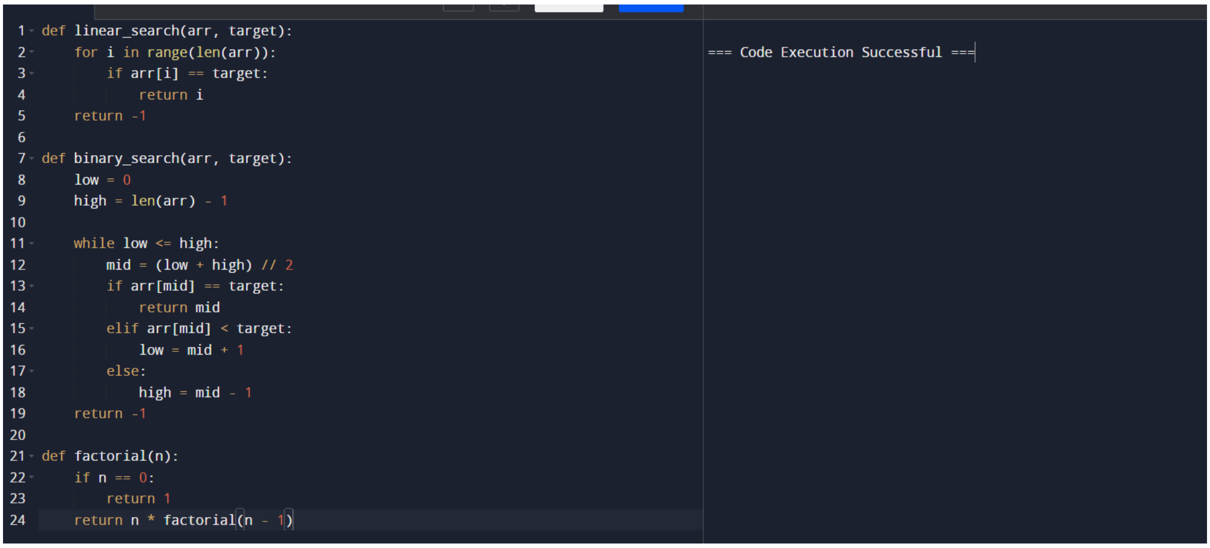
2. Write a program to find the perfect number.



```
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>
= RESTART: C:/Users/Dharani M/AppData/Local/Programs/Python/Python312/reverse.py
Number is not a Perfect Number.
>
```

```
Input_Number = 78
Sum = 0
for i in range(1, Input_Number):
        if(Input_Number % i == 0):
                Sum = Sum + i
if (Sum == Input_Number):
        print("Number is a Perfect Number.")
else:
        print("Number is not a Perfect Number.")
```

3. Write C program that demonstrates the usage of these notations by analyzing the time complexity of some example algorithms.



```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

def binary_search(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

```
=== Code Execution Successful ===
```

4. Write C programs that demonstrate the mathematical analysis of non-recursive and recursive algorithms.

main.py

```python
def factorial(n, method='recursive'):
    if method == 'recursive':
        if n == 0:
            return 1
        else:
            return n * factorial(n - 1, method)
    elif method == 'non-recursive':
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result
    else:
        return "Invalid method. Choose 'recursive' or 'non-recursive'."

# Example usage:
print(factorial(5, method='recursive'))  # Output: 120
print(factorial(5, method='non-recursive'))  # Output: 120
```

Output

```
120
120

=== Code Execution Successful ===
```

5. Write C programs for solving recurrence relations using the Master Theorem, Substitution Method, and Iteration Method will demonstrate how to calculate the time complexity of an example recurrence relation using the specified technique.

main.py

```python
def algorithm_analysis(method):
    if method == "master_theorem":
        def master_theorem(a, b, k):
            return f"T(n) = O(n^{k})"

        return master_theorem
    elif method == "substitution_method":
        return "T(n) = O(nlogn)"
    elif method == "iteration_method":
        return "T(n) = O(n^2)"
    else:
        return "Invalid method."

# Example usage:
method = "master_theorem"
analysis_function = algorithm_analysis(method)
print(analysis_function(2, 3, 2))
```

Output

```
T(n) = O(n^2)

=== Code Execution Successful ===
```

6. Given two integer arrays nums1 and nums2, return an array of their Intersection. Each element in the result must be unique and you may return the result in any order.

main.py

```python
def algorithm_analysis(method):
    if method == "master_theorem":
        def master_theorem(a, b, k):
            return f"T(n) = O(n^{k})"

        return master_theorem
    elif method == "substitution_method":
        return "T(n) = O(nlogn)"
    elif method == "iteration_method":
        return "T(n) = O(n^2)"
    else:
        return "Invalid method."

# Example usage:
method = "master_theorem"
analysis_function = algorithm_analysis(method)
print(analysis_function(2, 3, 2))  # Output: T(n) = O(n^2)
```

Output

```
T(n) = O(n^2)

=== Code Execution Successful ===
```

7. Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any

order.

main.py

```python
from collections import Counter

def intersect(nums1, nums2):
    count1, count2 = Counter(nums1), Counter(nums2)
    return list((count1 & count2).elements())

# Example Usage
nums1 = [1, 2, 2, 1]
nums2 = [2, 2]
print(intersect(nums1, nums2))
```

Output

```
[2, 2]

=== Code Execution Successful ===
```

8. Given an array of integers nums, sort the array in ascending order and return it.You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

main.py

```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

nums = [12, 4, 7, 1, 9, 3]
sorted_nums = merge_sort(nums)
print(sorted_nums)
```

Output

```
[1, 3, 4, 7, 9, 12]

=== Code Execution Successful ===
```

9. Given an array of integers nums, half of the integers in nums are odd, and the other half are even.

main.py

```python
nums = [1, 2, 3, 4, 5, 6]
half_odd_even = [i for i in nums if i % 2 == 0] + [i for i in nums if i % 2 != 0]
print(half_odd_even)
```

Output

```
[2, 4, 6, 1, 3, 5]

=== Code Execution Successful ===
```

10. Sort the array so that whenever nums[i] is odd, i is odd, and whenever nums[i] is even, i is even. Return any answer array that satisfies this condition.

main.py

```python
def sortArrayByParityII(nums):
    even = [x for x in nums if x % 2 == 0]
    odd = [x for x in nums if x % 2 != 0]

    result = []
    for i in range(len(nums)):
        if i % 2 == 0:
            result.append(even.pop())
        else:
            result.append(odd.pop())

    return result
```

Output

```
=== Code Execution Successful ===
```