03_September_Python Assignment_For Loops

September 8, 2023

[1]:
```
#1.Write a Python program to print the numbers from 1 to 10 using a `for` loop.

for i in range(1, 11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

[2]:
```
#2. Create a program that calculates the sum of all numbers in a list using a␣
 ↪`for` loop.

# Define a list of numbers
numbers = [5, 10, 15, 20, 25]

# Initialize a variable to store the sum
total_sum = 0

# Use a for loop to iterate through the list and calculate the sum
for num in numbers:
    total_sum += num

# Print the result
print("The sum of all numbers in the list is:", total_sum)
```

```
The sum of all numbers in the list is: 75
```

[3]:
```
#3. Write a program to print the characters of a string in reverse order using␣
 ↪a `for` loop.

# Input string
```

```
input_string = "Hello, World!"

# Initialize an empty string to store the reversed characters
reversed_string = ""

# Use a for loop to iterate through the string in reverse
for char in reversed(input_string):
    reversed_string += char

# Print the reversed string
print("Original string:", input_string)
print("Reversed string:", reversed_string)
```

```
Original string: Hello, World!
Reversed string: !dlroW ,olleH
```

[4]:
```
#4. Develop a program that finds the factorial of a given number using a `for`␣
 ↪loop.

# Input: Get the number from the user
num = int(input("Enter a number: "))

# Initialize a variable to store the factorial (start with 1 for multiplication)
factorial = 1

# Use a for loop to calculate the factorial
for i in range(1, num + 1):
    factorial *= i

# Print the result
print(f"The factorial of {num} is {factorial}")
```

```
Enter a number:  10
```

```
The factorial of 10 is 3628800
```

[5]:
```
#5. Create a program to print the multiplication table of a given number using␣
 ↪a `for` loop.

# Input: Get the number for which you want to print the multiplication table
num = int(input("Enter a number: "))

# Use a for loop to print the multiplication table
print(f"Multiplication table for {num}:")
for i in range(1, 11):
    product = num * i
    print(f"{num} x {i} = {product}")
```

```
Enter a number:  15
```

```
Multiplication table for 15:
15 x 1 = 15
15 x 2 = 30
15 x 3 = 45
15 x 4 = 60
15 x 5 = 75
15 x 6 = 90
15 x 7 = 105
15 x 8 = 120
15 x 9 = 135
15 x 10 = 150
```

[6]:
```
#6. Write a program that counts the number of even and odd numbers in a list␣
 ↪using a `for` loop.

# Define a list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Initialize counters for even and odd numbers
even_count = 0
odd_count = 0

# Use a for loop to iterate through the list and count even and odd numbers
for num in numbers:
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1

# Print the results
print("List of numbers:", numbers)
print("Number of even numbers:", even_count)
print("Number of odd numbers:", odd_count)
```

```
List of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Number of even numbers: 4
Number of odd numbers: 5
```

[8]:
```
#7. Develop a program that prints the squares of numbers from 1 to 5 using a␣
 ↪`for` loop.


 # Define a list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Initialize counters for even and odd numbers
```

```python
even_count = 0
odd_count = 0

# Use a for loop to iterate through the list and count even and odd numbers
for num in numbers:
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1

# Print the results
print("List of numbers:", numbers)
print("Number of even numbers:", even_count)
print("Number of odd numbers:", odd_count)
```

```
List of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Number of even numbers: 5
Number of odd numbers: 5
```

[9]:
```python
#8. Create a program to find the length of a string without using the `len()`␣
↪function.

# Input: Get the string from the user
input_string = input("Enter a string: ")

# Initialize a counter to keep track of the length
length = 0

# Use a for loop to iterate through the characters of the string
for char in input_string:
    length += 1

# Print the length of the string
print(f"The length of the string is: {length}")
```

```
Enter a string:  My name is Ganesh

The length of the string is: 17
```

[10]:
```python
#9. Write a program that calculates the average of a list of numbers using a␣
↪`for` loop.

# Define a list of numbers
numbers = [5, 10, 15, 20, 25]

# Initialize variables for sum and count
total_sum = 0
count = 0
```

```
# Use a for loop to calculate the sum and count
for num in numbers:
    total_sum += num
    count += 1

# Calculate the average
average = total_sum / count

# Print the result
print("List of numbers:", numbers)
print("Average:", average)
```

```
List of numbers: [5, 10, 15, 20, 25]
Average: 15.0
```

[11]:
```
# Input: Get the value of 'n' from the user
n = int(input("Enter the number of Fibonacci numbers to generate: "))

# Initialize variables for the first two Fibonacci numbers
fibonacci_sequence = [0, 1]

# Use a for loop to generate the Fibonacci sequence
for i in range(2, n):
    next_fib = fibonacci_sequence[i - 1] + fibonacci_sequence[i - 2]
    fibonacci_sequence.append(next_fib)

# Print the first 'n' Fibonacci numbers
print("The first", n, "Fibonacci numbers are:")
for num in fibonacci_sequence:
    print(num, end=" ")
```

```
Enter the number of Fibonacci numbers to generate:  18

The first 18 Fibonacci numbers are:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

[13]:
```
#Intermediate Level:
#11. Write a program to check if a given list contains any duplicates using a␣
 ↪`for` loop.

# Define a list of elements (you can replace this with your own list)
my_list = [1, 2, 3, 4, 5, 2, 7, 8]

# Initialize an empty set to store unique elements
unique_elements = set()

# Initialize a flag to indicate if duplicates are found
```

```
has_duplicates = False

# Use a for loop to iterate through the list
for element in my_list:
    if element in unique_elements:
        has_duplicates = True
        break
    else:
        unique_elements.add(element)

# Check and print the result
if has_duplicates:
    print("The list contains duplicates.")
else:
    print("The list does not contain duplicates.")
```

```
The list contains duplicates.
```

[14]:

```
#12. Create a program that prints the prime numbers in a given range using a␣
 ↪`for` loop.

# Input: Get the range from the user
start = int(input("Enter the start of the range: "))
end = int(input("Enter the end of the range: "))

print(f"Prime numbers between {start} and {end} are:")

# Use a for loop to iterate through the range
for num in range(start, end + 1):
    if num > 1:
        is_prime = True

        # Check for factors
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                is_prime = False
                break

        # If it's prime, print it
        if is_prime:
            print(num, end=" ")
```

```
Enter the start of the range:  13
Enter the end of the range:  76
```

```
Prime numbers between 13 and 76 are:
13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73
```

[15]:
```
#13. Develop a program that counts the number of vowels in a string using a␣
 ↪`for` loop

# Input: Get the string from the user
input_string = input("Enter a string: ")

# Initialize a counter to keep track of the number of vowels
vowel_count = 0

# Define a set of vowels for checking
vowels = set("aeiouAEIOU")

# Use a for loop to iterate through the characters in the string
for char in input_string:
    if char in vowels:
        vowel_count += 1

# Print the result
print(f"The number of vowels in the string is: {vowel_count}")
```

```
Enter a string:  My name is Ganesh
```

```
The number of vowels in the string is: 5
```

[16]:
```
#14. Write a program to find the maximum element in a 2D list using a nested␣
 ↪`for` loop.

# Define a 2D list (matrix)
matrix = [
    [3, 5, 1],
    [8, 2, 7],
    [4, 9, 6]
]

# Initialize a variable to store the maximum element (start with the first␣
 ↪element)
max_element = matrix[0][0]

# Use nested for loops to iterate through the matrix
for row in matrix:
    for element in row:
        if element > max_element:
            max_element = element

# Print the maximum element
print("The maximum element in the 2D list is:", max_element)
```

```
The maximum element in the 2D list is: 9
```

[17]:
```python
#15. Create a program that removes all occurrences of a specific element from a␣
↪list using a `for` loop.

# Define a list (you can replace this with your own list)
my_list = [1, 2, 3, 2, 4, 5, 2]

# Input: Get the element to remove from the user
element_to_remove = int(input("Enter the element to remove: "))

# Use a for loop to remove all occurrences of the element
for num in my_list[:]:  # Use slicing to create a copy of the list
    if num == element_to_remove:
        my_list.remove(num)

# Print the modified list
print("List after removing all occurrences of", element_to_remove, ":", my_list)
```

```
Enter the element to remove:  4
```

```
List after removing all occurrences of 4 : [1, 2, 3, 2, 5, 2]
```

[18]:
```python
#16. Develop a program that generates a multiplication table for numbers from 1␣
↪to 5 using a nested `for` loop.

# Define the range of numbers for the multiplication table
start = 1
end = 5

# Use nested for loops to generate the multiplication table
for i in range(1, 11):  # Numbers 1 to 10
    for j in range(start, end + 1):  # Numbers from 'start' to 'end'
        result = i * j
        print(f"{j} x {i} = {result}\t", end=" ")
    print()  # Move to the next line for the next row
```

```
1 x 1 = 1        2 x 1 = 2        3 x 1 = 3        4 x 1 = 4        5 x 1 = 5
1 x 2 = 2        2 x 2 = 4        3 x 2 = 6        4 x 2 = 8        5 x 2 = 10
1 x 3 = 3        2 x 3 = 6        3 x 3 = 9        4 x 3 = 12       5 x 3 = 15
1 x 4 = 4        2 x 4 = 8        3 x 4 = 12       4 x 4 = 16       5 x 4 = 20
1 x 5 = 5        2 x 5 = 10       3 x 5 = 15       4 x 5 = 20       5 x 5 = 25
1 x 6 = 6        2 x 6 = 12       3 x 6 = 18       4 x 6 = 24       5 x 6 = 30
1 x 7 = 7        2 x 7 = 14       3 x 7 = 21       4 x 7 = 28       5 x 7 = 35
1 x 8 = 8        2 x 8 = 16       3 x 8 = 24       4 x 8 = 32       5 x 8 = 40
1 x 9 = 9        2 x 9 = 18       3 x 9 = 27       4 x 9 = 36       5 x 9 = 45
1 x 10 = 10      2 x 10 = 20      3 x 10 = 30      4 x 10 = 40      5 x 10 = 50
```

[19]:
```
#17. Write a program that converts a list of Fahrenheit temperatures to Celsius␣
 ↪using a `for` loop.

# Define a list of Fahrenheit temperatures
fahrenheit_temperatures = [32, 68, 95, 104, 212]

# Initialize an empty list to store Celsius temperatures
celsius_temperatures = []

# Use a for loop to convert Fahrenheit to Celsius
for fahrenheit in fahrenheit_temperatures:
    celsius = (fahrenheit - 32) * 5/9
    celsius_temperatures.append(celsius)

# Print the results
print("Fahrenheit Temperatures:", fahrenheit_temperatures)
print("Celsius Temperatures:", celsius_temperatures)
```

```
Fahrenheit Temperatures: [32, 68, 95, 104, 212]
Celsius Temperatures: [0.0, 20.0, 35.0, 40.0, 100.0]
```

[20]:
```
#18. Create a program to print the common elements from two lists using a `for`␣
 ↪loop.

# Define two lists (you can replace these with your own lists)
list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]

# Initialize an empty list to store common elements
common_elements = []

# Use a for loop to find and store common elements
for item1 in list1:
    for item2 in list2:
        if item1 == item2:
            common_elements.append(item1)

# Print the common elements
print("Common elements between list1 and list2:", common_elements)
```

```
Common elements between list1 and list2: [3, 4, 5]
```

[21]:
```
#19. Develop a program that prints the pattern of right-angled triangles using␣
 ↪a `for` loop. Use '*' to draw the pattern

# Input: Get the number of rows for the right-angled triangle from the user
```

```
num_rows = int(input("Enter the number of rows for the right-angled triangle:␣
↪"))

# Use a for loop to print the right-angled triangle pattern
for i in range(1, num_rows + 1):
    for j in range(1, i + 1):
        print("*", end=" ")
    print()
```

```
Enter the number of rows for the right-angled triangle:  11

*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * * *
```

[22]:
```
#20. Write a program to find the greatest common divisor (GCD) of two numbers␣
↪using a `for` loop.

# Input: Get the two numbers from the user
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

# Find the smaller of the two numbers
if num1 < num2:
    smaller = num1
else:
    smaller = num2

# Initialize a variable to store the GCD
gcd = 1

# Use a for loop to find the GCD
for i in range(1, smaller + 1):
    if num1 % i == 0 and num2 % i == 0:
        gcd = i

# Print the GCD
print(f"The GCD of {num1} and {num2} is {gcd}")
```

```
Enter the first number:  12
Enter the second number:  11
```

```
The GCD of 12 and 11 is 1
```

[23]:
```python
#Advanced Level:
#21. Create a program that calculates the sum of the digits of numbers in a␣
 ↪list using a list comprehension.

# Define a list of numbers
numbers = [123, 45, 6789, 321]

# Define a list comprehension to calculate the sum of digits for each number
sums = [sum(int(digit) for digit in str(num)) for num in numbers]

# Print the result
print("Original list of numbers:", numbers)
print("Sum of digits for each number:", sums)
```

```
Original list of numbers: [123, 45, 6789, 321]
Sum of digits for each number: [6, 9, 30, 6]
```

[24]:
```python
#22. Write a program to find the prime factors of a given number using a `for`␣
 ↪loop and list comprehension.

# Input: Get the number from the user
number = int(input("Enter a number: "))

# Define a function to check if a number is prime
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

# Use a list comprehension to find prime factors
prime_factors = [x for x in range(2, number + 1) if number % x == 0 and␣
 ↪is_prime(x)]

# Print the prime factors
print(f"The prime factors of {number} are:", prime_factors)
```

```
Enter a number:  123
```

```
The prime factors of 123 are: [3, 41]
```

[25]:
```
#23. Develop a program that extracts unique elements from a list and stores␣
↪them in a new list using a list comprehension.

# Define a list with duplicate elements (you can replace this with your own␣
↪list)
original_list = [1, 2, 2, 3, 4, 4, 5, 6, 6]

# Use a list comprehension to extract unique elements
unique_elements = [x for i, x in enumerate(original_list) if x not in␣
↪original_list[:i]]

# Print the unique elements
print("Original list:", original_list)
print("Unique elements:", unique_elements)
```

```
Original list: [1, 2, 2, 3, 4, 4, 5, 6, 6]
Unique elements: [1, 2, 3, 4, 5, 6]
```

[26]:
```
#24. Create a program that generates a list of all palindromic numbers up to a␣
↪specified limit using a list comprehension.

# Input: Get the upper limit from the user
limit = int(input("Enter the upper limit for palindromic numbers: "))

# Define a list comprehension to generate palindromic numbers
palindromic_numbers = [num for num in range(1, limit + 1) if str(num) ==␣
↪str(num)[::-1]]

# Print the list of palindromic numbers
print("Palindromic numbers up to", limit, "are:", palindromic_numbers)
```

```
Enter the upper limit for palindromic numbers:  15
```

```
Palindromic numbers up to 15 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 11]
```

[27]:
```
#25. Write a program to flatten a nested list using list comprehension.

# Define a nested list (you can replace this with your own nested list)
nested_list = [[1, 2, 3], [4, 5], [6, 7, 8]]

# Use list comprehension to flatten the nested list
flattened_list = [item for sublist in nested_list for item in sublist]

# Print the flattened list
print("Original nested list:", nested_list)
print("Flattened list:", flattened_list)
```

```
Original nested list: [[1, 2, 3], [4, 5], [6, 7, 8]]
```

```
Flattened list: [1, 2, 3, 4, 5, 6, 7, 8]
```

[28]:
```
#26. Develop a program that computes the sum of even and odd numbers in a list␣
  ↪separately using list comprehension.

# Define a list of numbers (you can replace this with your own list)
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Use list comprehensions to calculate the sum of even and odd numbers␣
  ↪separately
even_sum = sum([num for num in numbers if num % 2 == 0])
odd_sum = sum([num for num in numbers if num % 2 != 0])

# Print the sums
print("Original list of numbers:", numbers)
print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
```

```
Original list of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Sum of even numbers: 20
Sum of odd numbers: 25
```

[31]:
```
#27. Create a program that generates a list of squares of odd numbers between 1␣
  ↪and 10 using list comprehension.

# Use list comprehension to generate squares of odd numbers between 1 and 10
squares_of_odd_numbers = [num ** 2 for num in range(1, 10) if num % 2 != 0]

# Print the list of squares
print("Squares of odd numbers between 1 and 10:", squares_of_odd_numbers)
```

```
Squares of odd numbers between 1 and 10: [1, 9, 25, 49, 81]
```

[32]:
```
#28. Write a program that combines two lists into a dictionary using list␣
  ↪comprehension.

# Define two lists (you can replace these with your own lists)
keys = ["name", "age", "city"]
values = ["Alice", 30, "New York"]

# Use list comprehension to combine the lists into a dictionary
combined_dict = {keys[i]: values[i] for i in range(len(keys))}

# Print the resulting dictionary
print("Combined Dictionary:", combined_dict)
```

```
Combined Dictionary: {'name': 'Alice', 'age': 30, 'city': 'New York'}
```

[33]:
```python
#29. Develop a program that extracts the vowels from a string and stores them␣
↪in a list using list comprehension.

# Input: Get the string from the user
input_string = input("Enter a string: ")

# Use list comprehension to extract vowels from the string
vowels = [char for char in input_string if char.lower() in "aeiou"]

# Print the list of vowels
print("Vowels in the string:", vowels)
```

```
Enter a string:  My name is Ganesh

Vowels in the string: ['a', 'e', 'i', 'a', 'e']
```

[34]:
```python
#30. Create a program that removes all non-numeric characters from a list of␣
↪strings using list comprehension.

# Define a list of strings (you can replace this with your own list)
string_list = ["abc123", "def456", "ghi789", "jkl"]

# Use list comprehension to remove non-numeric characters from each string
numeric_strings = [''.join(char for char in string if char.isnumeric()) for␣
↪string in string_list]

# Print the list of numeric strings
print("Original list of strings:", string_list)
print("Numeric strings:", numeric_strings)
```

```
Original list of strings: ['abc123', 'def456', 'ghi789', 'jkl']
Numeric strings: ['123', '456', '789', '']
```

[35]:
```python
#Challenge level

#31. Write a program to generate a list of prime numbers using the Sieve of␣
↪Eratosthenes algorithm and list comprehension.

# Input: Get the upper limit for prime numbers from the user
limit = int(input("Enter the upper limit for prime numbers: "))

# Create a list of boolean values to represent prime status (True for prime,␣
↪False for composite)
is_prime = [True] * (limit + 1)
is_prime[0] = is_prime[1] = False

# Apply the Sieve of Eratosthenes algorithm to mark composites
```

```
for num in range(2, int(limit**0.5) + 1):
    if is_prime[num]:
        for multiple in range(num * num, limit + 1, num):
            is_prime[multiple] = False

# Use list comprehension to generate a list of prime numbers
prime_numbers = [num for num, status in enumerate(is_prime) if status]

# Print the list of prime numbers
print("Prime numbers up to", limit, "are:", prime_numbers)
```

```
Enter the upper limit for prime numbers:  100

Prime numbers up to 100 are: [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]
```

[36]:
```
#32. Create a program that generates a list of all Pythagorean triplets up to a␣
 ↪specified limit using list comprehension.

# Input: Get the upper limit for Pythagorean triplets from the user
limit = int(input("Enter the upper limit for Pythagorean triplets: "))

# Use list comprehension to generate Pythagorean triplets
pythagorean_triplets = [(a, b, c) for a in range(1, limit + 1)
                                    for b in range(a, limit + 1)
                                    for c in range(b, limit + 1)
                                    if a**2 + b**2 == c**2]

# Print the list of Pythagorean triplets
print("Pythagorean triplets up to", limit, "are:", pythagorean_triplets)
```

```
Enter the upper limit for Pythagorean triplets:  26

Pythagorean triplets up to 26 are: [(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24,
25), (8, 15, 17), (9, 12, 15), (10, 24, 26), (12, 16, 20), (15, 20, 25)]
```

[38]:
```
#33. Develop a program that generates a list of all possible combinations of␣
 ↪two lists using list comprehension.

# Define two lists (you can replace these with your own lists)
list1 = [1, 2, 3, 4, 5]
list2 = ['a', 'b', 'c']

# Use list comprehension to generate all possible combinations of elements from␣
 ↪both lists
combinations = [(x, y) for x in list1 for y in list2]

# Print the list of combinations
```

```
print("List 1:", list1)
print("List 2:", list2)
print("All possible combinations:", combinations)
```

```
List 1: [1, 2, 3, 4, 5]
List 2: ['a', 'b', 'c']
All possible combinations: [(1, 'a'), (1, 'b'), (1, 'c'), (2, 'a'), (2, 'b'),
(2, 'c'), (3, 'a'), (3, 'b'), (3, 'c'), (4, 'a'), (4, 'b'), (4, 'c'), (5, 'a'),
(5, 'b'), (5, 'c')]
```

[41]:
```
#34. Write a program that calculates the mean, median, and mode of a list of␣
 ↪numbers using list comprehension.

from collections import Counter

# Define a list of numbers (you can replace this with your own list)
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 9]

# Calculate the mean using list comprehension
mean = sum(numbers) / len(numbers)

# Calculate the median using list comprehension
sorted_numbers = sorted(numbers)
median = (sorted_numbers[len(sorted_numbers) // 2] +␣
 ↪sorted_numbers[(len(sorted_numbers) - 1) // 2]) / 2

# Calculate the mode using list comprehension and Counter
counter = Counter(numbers)
mode = [num for num, count in counter.items() if count == max(counter.values())]

# Print the mean, median, and mode
print("List of numbers:", numbers)
print("Mean:", mean)
print("Median:", median)
print("Mode:", mode)
```

```
List of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 9]
Mean: 5.4
Median: 5.5
Mode: [9]
```

[42]:
```
#35. Create a program that generates Pascal's triangle up to a specified number␣
 ↪of rows using list
#comprehension.
```

```
# Input: Get the number of rows for Pascal's triangle from the user
num_rows = int(input("Enter the number of rows for Pascal's triangle: "))

# Function to calculate the next row of Pascal's triangle
def generate_next_row(prev_row):
    next_row = [1]  # The first element of each row is always 1
    next_row.extend([prev_row[i] + prev_row[i + 1] for i in range(len(prev_row)␣
 ↪- 1)])
    next_row.append(1)  # The last element of each row is always 1
    return next_row

# Generate Pascal's triangle using list comprehension
pascals_triangle = [[1]]
[pascals_triangle.append(generate_next_row(pascals_triangle[-1])) for _ in␣
 ↪range(num_rows - 1)]

# Print Pascal's triangle
print("Pascal's triangle with", num_rows, "rows:")
for row in pascals_triangle:
    print(row)
```

```
Enter the number of rows for Pascal's triangle:  28

Pascal's triangle with 28 rows:
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
[1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]
[1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
[1, 12, 66, 220, 495, 792, 924, 792, 495, 220, 66, 12, 1]
[1, 13, 78, 286, 715, 1287, 1716, 1716, 1287, 715, 286, 78, 13, 1]
[1, 14, 91, 364, 1001, 2002, 3003, 3432, 3003, 2002, 1001, 364, 91, 14, 1]
[1, 15, 105, 455, 1365, 3003, 5005, 6435, 6435, 5005, 3003, 1365, 455, 105, 15,
1]
[1, 16, 120, 560, 1820, 4368, 8008, 11440, 12870, 11440, 8008, 4368, 1820, 560,
120, 16, 1]
[1, 17, 136, 680, 2380, 6188, 12376, 19448, 24310, 24310, 19448, 12376, 6188,
2380, 680, 136, 17, 1]
[1, 18, 153, 816, 3060, 8568, 18564, 31824, 43758, 48620, 43758, 31824, 18564,
8568, 3060, 816, 153, 18, 1]
[1, 19, 171, 969, 3876, 11628, 27132, 50388, 75582, 92378, 92378, 75582, 50388,
27132, 11628, 3876, 969, 171, 19, 1]
[1, 20, 190, 1140, 4845, 15504, 38760, 77520, 125970, 167960, 184756, 167960,
125970, 77520, 38760, 15504, 4845, 1140, 190, 20, 1]
[1, 21, 210, 1330, 5985, 20349, 54264, 116280, 203490, 293930, 352716, 352716,
293930, 203490, 116280, 54264, 20349, 5985, 1330, 210, 21, 1]
[1, 22, 231, 1540, 7315, 26334, 74613, 170544, 319770, 497420, 646646, 705432,
646646, 497420, 319770, 170544, 74613, 26334, 7315, 1540, 231, 22, 1]
[1, 23, 253, 1771, 8855, 33649, 100947, 245157, 490314, 817190, 1144066,
1352078, 1352078, 1144066, 817190, 490314, 245157, 100947, 33649, 8855, 1771,
253, 23, 1]
[1, 24, 276, 2024, 10626, 42504, 134596, 346104, 735471, 1307504, 1961256,
2496144, 2704156, 2496144, 1961256, 1307504, 735471, 346104, 134596, 42504,
10626, 2024, 276, 24, 1]
[1, 25, 300, 2300, 12650, 53130, 177100, 480700, 1081575, 2042975, 3268760,
4457400, 5200300, 5200300, 4457400, 3268760, 2042975, 1081575, 480700, 177100,
53130, 12650, 2300, 300, 25, 1]
[1, 26, 325, 2600, 14950, 65780, 230230, 657800, 1562275, 3124550, 5311735,
7726160, 9657700, 10400600, 9657700, 7726160, 5311735, 3124550, 1562275, 657800,
230230, 65780, 14950, 2600, 325, 26, 1]
[1, 27, 351, 2925, 17550, 80730, 296010, 888030, 2220075, 4686825, 8436285,
13037895, 17383860, 20058300, 20058300, 17383860, 13037895, 8436285, 4686825,
2220075, 888030, 296010, 80730, 17550, 2925, 351, 27, 1]
```

[43]:
```
#36. Develop a program that calculates the sum of the digits of a factorial of␣
 ↪numbers from 1 to 5 using list comprehension.

# Function to calculate the factorial of a number
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Use list comprehension to calculate the sum of digits of factorial for␣
 ↪numbers from 1 to 5
factorials = [factorial(n) for n in range(1, 6)]
sum_of_digits = [sum(int(digit) for digit in str(factorial)) for factorial in␣
 ↪factorials]

# Print the results
print("Factorials for numbers 1 to 5:", factorials)
print("Sum of digits of factorials:", sum_of_digits)
```

```
Factorials for numbers 1 to 5: [1, 2, 6, 24, 120]
Sum of digits of factorials: [1, 2, 6, 6, 3]
```

[44]:
```
#37. Write a program that finds the longest word in a sentence using list␣
 ↪comprehension.

# Input: Get a sentence from the user
sentence = input("Enter a sentence: ")

# Use list comprehension to split the sentence into words
words = sentence.split()

# Find the longest word using list comprehension
longest_word = max(words, key=len)

# Print the longest word
print("Longest word in the sentence:", longest_word)
```

```
Enter a sentence:  India is one of the largest democracies in the world.
```

```
Longest word in the sentence: democracies
```

[45]:
```
#38. Create a program that filters a list of strings to include only those with␣
 ↪more than three vowels using list comprehension.

# Define a list of strings (you can replace this with your own list)
string_list = ["hello", "world", "example", "vowel", "programming", "python"]

# Function to count vowels in a string
def count_vowels(s):
    vowels = "aeiouAEIOU"
    return sum(1 for char in s if char in vowels)

# Use list comprehension to filter strings with more than three vowels
filtered_strings = [s for s in string_list if count_vowels(s) > 3]

# Print the filtered list
print("Original list of strings:", string_list)
print("Strings with more than three vowels:", filtered_strings)
```

```
Original list of strings: ['hello', 'world', 'example', 'vowel', 'programming',
'python']
Strings with more than three vowels: []
```

[46]:
```
#39. Develop a program that calculates the sum of the digits of numbers from 1␣
 ↪to 1000 using list
#comprehension.

# Use list comprehension to calculate the sum of digits for numbers from 1 to␣
 ↪1000
```

```python
sum_of_digits = [sum(int(digit) for digit in str(num)) for num in range(1,␣
 ↪1001)]

# Print the list of sums
print("Sum of digits for numbers from 1 to 1000:", sum_of_digits)
```

```
Sum of digits for numbers from 1 to 1000: [1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3,
4, 5, 6, 7, 8, 9, 10, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 6,
7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1, 2, 3, 4,
5, 6, 7, 8, 9, 10, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 6,
7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 8, 9,
10, 11, 12, 13, 14, 15, 16, 17, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 16, 17,
18, 19, 20, 21, 22, 23, 24, 25, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 1]
```

[47]:

```
#40. Write a program that generates a list of prime palindromic numbers using␣
 ↪list comprehension.

# Function to check if a number is prime
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

# Use list comprehension to generate prime palindromic numbers
prime_palindromic_numbers = [num for num in range(1, 1000) if is_prime(num) and␣
 ↪str(num) == str(num)[::-1]]

# Print the list of prime palindromic numbers
print("Prime palindromic numbers up to 1000:", prime_palindromic_numbers)
```

```
Prime palindromic numbers up to 1000: [2, 3, 5, 7, 11, 101, 131, 151, 181, 191,
313, 353, 373, 383, 727, 757, 787, 797, 919, 929]
```

[ ]:

```
```