1.Illustrate the use of negative indexing of list with example.

Negative indexing in Python allows you to access elements in a list from the end, starting with -1 for the last element, -2 for the second-to-last, and so on.

example:

```
fruits = ['apple', 'banana', 'cherry', 'date', 'elderberry']
```

```
print(fruits[-1])   Output: elderberry (last element)
print(fruits[-2])   Output: date (second-to-last element)
print(fruits[-3])   Output: cherry (third-to-last element)
```

3a) Python program to count how many times each character appears in a given string and store the count in a dictionary with the character as the key:

```
def count_characters(input_string):
    char_count = {}
    for char in input_string:
        if char in char_count:
            char_count[char] += 1
        else:
            char_count[char] = 1

    return char_count


input_string = input("Enter a string: ")
result = count_characters(input_string)
```

```
print("Character count:", result)
```

3b) Create a function min_max() that takes a list of numbers as an argument and returns the smallest and largest numbers:

```
def min_max(numbers):

    if not numbers:
        return None, None

    smallest = min(numbers)
    largest = max(numbers)
    return smallest, largest



numbers = [int(x) for x in input("Enter numbers separated by spaces: ").split()]
smallest, largest = min_max(numbers)
print(f"The smallest number is {smallest}, and the largest number is {largest}.")
```

14a) Python program to read n integers into a list and separate the positive and negative numbers into two different lists:

```
def separate_numbers(numbers):
    positive_numbers = []
    negative_numbers = []

    for num in numbers:
        if num >= 0:
            positive_numbers.append(num)
```

```
        else:
            negative_numbers.append(num)

    return positive_numbers, negative_numbers



numbers = [int(x) for x in input("Enter numbers separated by spaces: ").split()]
positive_numbers, negative_numbers = separate_numbers(numbers)
print("Positive numbers:", positive_numbers)
print("Negative numbers:", negative_numbers)
```

14b) Create a dictionary of names and birthdays. Write a Python program that asks the user to enter a name, and the program displays the birthday of that person:

```
def get_birthday(name, birthday_dict):

    if name in birthday_dict:
        return birthday_dict[name]
    else:
        return "Birthday not found."



birthday_dict = {
    'Alice': '1990-04-25',
    'Bob': '1985-11-12',
    'Charlie': '1992-06-30'
}
```

```
name = input("Enter a name to find the birthday: ")
birthday = get_birthday(name, birthday_dict)
print(f"The birthday of {name} is: {birthday}")
```

Write Python code for the following statements

1) Write the text"PROGRAMMING IN PYTHON" to a file with name code. txt

2)then reads the text again and prints it to the screen.

Ans1)

```
With open('code.txt', 'w') as file:
   File.write('PROGRAMMING IN PYTHON')
```

Ans2)

```
With open('code.txt', 'w') as file:
   File.write('PROGRAMMING IN PYTHON')

With open('code.txt', 'r') as file:
   Content = file.read()
   Print(content)
```

JAN 2024

Q. Write a Python program to print all palindromes in a line of text.

Ans:

```
def is_palindrome(word):
   return word == word[::-1]

def find_palindromes(text):
   words = text.lower().split()
   palindromes = [word for word in words if is_palindrome(word)]
   return palindromes
```

```
text = input("Enter a line of text: ")

palindrome_words = find_palindromes(text)

if palindrome_words:
    print("Palindromes in the text:", ", ".join(palindrome_words))
else:
    print("No palindromes found in the text.")
```

Output:

Enter a line of text: Python is awesome!

No palindromes found in the text.

Q. Illustrate the use of any four dictionary methods.

Ans:

1. keys() – Returns all keys in the dictionary
2. values() – Returns all values in the dictionary
3. items() – Returns key-value pairs as tuples
4. get() – Safely retrieves a value without errors

eg:

```
student_marks = {
    "Alice": 85,
    "Bob": 92,
    "Charlie": 78,
    "David": 90
```

```
}
print("Keys:", student_marks.keys())   #  Output: dict_keys(['Alice', 'Bob', 'Charlie', 'David']
print("Values:", student_marks.values())   # Output: dict_values([85, 92, 78, 90])
print("Items:", student_marks.items())
# Output: dict_items([('Alice', 85), ('Bob', 92), ('Charlie', 78), ('David', 90)])
print("Charlie's marks:", student_marks.get("Charlie"))   # Output: 78
print("Eve's marks:", student_marks.get("Eve", "Not found"))   # Output: Not found
```

Q. Write a Python program to convert a decimal number to its binary equivalent.

Ans:

```
def decimal_to_binary(n):
    binary = ""
    while n > 0:
        remainder = n % 2
        binary = str(remainder) + binary
        n = n // 2

    return binary if binary else "0"

decimal = int(input("Enter a decimal number: "))
binary = decimal_to_binary(decimal)
print(f"Binary equivalent of {decimal} is: {binary}")
```

Output:

```
Enter a decimal number: 25
Binary equivalent of 25 is: 11001
```

Q. Write a Python program to read a text file and store the count of occurrences of each character in a dictionary.

Ans:

```
def count_characters(filename):
    char_count = {}

    try:
        with open(filename, "r", encoding="utf-8") as file:
            for line in file:
                for char in line:
                    if char in char_count:
                        char_count[char] += 1
                    else:
                        char_count[char] = 1

        return char_count

    except FileNotFoundError:
        print("Error: File not found.")
        return {}

filename = "sample.txt"

char_frequencies = count_characters(filename)
```

```
for char, count in char_frequencies.items():
    print(f"'{char}': {count}")
```

Output:

'h': 1

'e': 1

'l': 3

'o': 2

' ': 1

'w': 1

'r': 1

'd': 1

MAY 2024

1)Write a Python program to create a list of squares for the numbers from 0 to 9 using list comprehension.

Ans)

```
squares = [x**2 for x in range(10)]
print(squares)
```

Output

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

2. Write a user-defined function to generate even numbers between 1 and 25.

Ans)

```
def even_numbers():
    evens = [x for x in range(1, 26) if x % 2 == 0]
    return evens

print(even_numbers())
```

Output:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]

Essay

3.a)Describe the concept of recursive function in Python with a suitable example.

A recursive function is a function that calls itself during its execution. Recursion allows solving problems by breaking them into smaller subproblems of the same type. Every recursive function must have a base case to stop the recursion, preventing infinite loops.

Example: Factorial Calculation Using Recursion

```
def factorial(n):
    if n == 0 or n == 1:  # Base case
        return 1
    else:
        return n * factorial(n - 1)  # Recursive call

print(factorial(5))  # Output: 120
```

b)Explain how to read numeric values from a file, perform some operations, and then write the results back to the file.

Steps to Read, Process, and Write Numeric Values in a File

1. Open the file and read the numeric values.

2. Perform some mathematical operations (e.g., sum, average).

3. Write the results back to the file.

Example:

```
# Writing numbers to a file
with open("numbers.txt", "w") as file:
    file.write("10\n20\n30\n40\n50")

# Reading numbers from the file and performing operations
with open("numbers.txt", "r") as file:
    numbers = [int(line.strip()) for line in file]

sum_numbers = sum(numbers)
```

```
average = sum_numbers / len(numbers)

# Writing results back to the file
with open("results.txt", "w") as file:
    file.write(f"Sum: {sum_numbers}\n")
    file.write(f"Average: {average}\n")
```

4.a)Compare and contrast the fundamental characteristics and use cases of lists, tuples, and sets in Python.

Ans)

1. List (Ordered, Mutable, Allows Duplicates)

Use lists when you need an ordered collection of elements that can be changed dynamically.

```
# Creating a list
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")  # Adding an element
print(fruits)  # Output: ['apple', 'banana', 'cherry', 'orange']
```

2. Tuple (Ordered, Immutable, Allows Duplicates)

Use tuples when you need a fixed collection of items that should not change.

```
# Creating a tuple
coordinates = (10, 20)
# coordinates[0] = 15  # This will raise an error (tuples are immutable)
```

```
print(coordinates)  # Output: (10, 20)
```

3. Set (Unordered, Mutable, Unique Elements)

Use sets when you need a collection of unique values and do not require ordering.

```
# Creating a set
numbers = {1, 2, 3, 4, 4, 2}  # Duplicates are automatically removed
numbers.add(5)
print(numbers)  # Output: {1, 2, 3, 4,5}
```

b)Create a Python program that uses a dictionary to store the names and ages of people. Ask the user to enter a name, and the program should display the age of that person.

```
# Creating a dictionary to store names and ages
people = {
   "Alice": 25,
   "Bob": 30,
   "Charlie": 22,
   "David": 28
}

# Asking the user to enter a name
name = input("Enter a name: ")

# Displaying the age of the entered name
if name in people:
```

```
    print(f"{name} is {people[name]} years old.")
else:
    print("Name not found in the dictionary.")
```