PART - A (20 × 1 = 20 Marks)
Answer **ALL** Questions

1. Which of the following is not a relational operator?

(A) >=

(B) ==

(C) *

(D) !=

Answer : (C) *

2. Find the output of the statement:

X = 7 * 6 % 15 / 9.

(A) 1

(B) 3

(C) 2

(D) 4

Answer : (A) 1

3. What is the output of the given statement?

printf("%d", ++ + 2);

(A) 2

(B) 3

(C) 1

(D) Compiler error - Lvalue required

Answer : (D) Compiler error - Lvalue required

4. Result of 16 >> 2 is:

(A) 0

(B) 3

(C) 4

(D) 8

Answer : (C) 4

5. Which of the following type of variable is not supported by switch-case?

(A) int

(B) char

(C) float

(D) short

Answer : (C) float

6. **Correct way to initialize an array in C language is:**

(A) int a = {1, 2, 3};

(B) int a[3] = {1, 2, 3};

(C) int a[] = new int[3];

(D) int a(3) = [1, 2, 3];

Answer : (B) int a[3] = {1, 2, 3};

7. **Find the output of the following C code:**

```
int c;
for (c = 1; c <= 5;)
printf("%d", ++c);
```

(A) 1 2 3 4 5

(B) 2 3 4 5

(C) 2 3 4 5 6

(D) 1 2 3 4

Answer : (C) 2 3 4 5 6

- **++c increments c before printing.**
- **Loop starts with c = 1, so the first ++c makes c = 2, and it gets printed.**
- **Loop continues with c = 3, 4, 5, and 6 before exiting.**

8. **An array is declared as int a[] = {0, 2, 4, 6, 8, 10}. Find the value of a[a[2]].**

(A) 8

(B) 4

(C) 2

(D) 6

Answer: (D) 6

9. **Find the output of the following C code.**

```
char str[] = {'h', 'e', 'l', 'l', 'o', '\0'};
printf("%d %d", strlen(str), sizeof(str));
```

(A) 4 4

(B) 4 6

(C) 6 4

(D) 6 6

Answer: (D) 6 6

10. What is the output of the following C code?

```
char *p = "abc";
printf("%s %s", p, p + 1);
```

(A) abc abc

(B) bc bc

(C) bc c

(D) abc bc

Answer: (D) abc bc

11. Identify the error in the following function declaration statement.

```
void function(int);
```

(A) Has one argument

(B) Has no argument

(C) Not returning any values

(D) Both (A) and (C)

Answer: (D) Both (A) and (C)

- **The function accepts one argument of type int.**
- **Since the return type is void, it does not return any value.**

12. The function used to read multi word strings is _____.

(A) scanf()

(B) printf()

(C) gets()

(D) puts()

Answer: (C) gets()

13. What is the output of the following Python code?

```
X = True
Y = False
print(X & Y)
```

(A) True

(B) 1

(C) 0

(D) False

Answer: (D) False

- **X & Y performs a bitwise AND. Since X is True (1) and Y is False (0), the result is False.**

14. _____ is used for single-line comments in Python.

(A) ||

(B) !

(C) #

(D) /* */

Answer: (C) #

15. Find the output of the following Python code:

```
var = 10
print(type(var))
var = "Hello"
print(type(var))
```

(A) str int

(B) int int

(C) str str

(D) int str

Answer: (D) int str

16. Resultant data type of the sorted tuple in Python is _____.

(A) List

(B) Tuple

(C) String

(D) Int

Answer: (A) List

17. The number of elements in the series S1 is _____.

```
S1 = pd.Series(range(6))
print(S1)
```

(A) 4

(B) 5

(C) 6

(D) 7

Answer: (C) 6

18. Attribute of Numpy array is

(A) objects, type, list

(B) objects, nonvectorization

(C) unicode, shape

(D) shape, dtype, ndim

Answer: (D) shape, dtype, ndim

19. Which of the following is used to convert Numpy array into list?

(A) array.list()

(B) array.list

(C) list(array)

(D) list.array

Answer: (C) list(array)

20. Which of the data structure has both row index and column index?

(A) List

(B) Series

(C) DataFrame

(D) Tuple

Answer: (C) DataFrame

PART – B (5 × 8 = 40 Marks)
Answer **ALL** Questions

21.a.i. Write a pseudo-code to find the factorial of a given number.

Pseudo code:

```
FUNCTION factorial(INT n)
        DECLARE INT res
        SET res = 1
        FOR i FROM 1 TO n STEP 1
                res=res*i
        END FOR

BEGIN Main
        DECLARE INT num,result
        INPUT num
        SET res = factorial(num)
        DISPLAY res
END Main
```

ii. **Write the difference between pre-increment and post-increment operator in C language. Give an example.**

Pre-increment	Post-increment
1. The value is incremented by 1 right away	1. The value is incremented by 1 when it encounters the same variable again
2. If variable is : i Pre-increment :- ++i	2. If variable is : i Post-increment :- i++
3. Eg: int i=0; printf(“%d”,++i); Output: 1	3. Eg: int i=0; printf(“%d”,i++); printf(“\n%d”,i); Output: 0 1

(OR)

b. **Compare the logical operator and bitwise operator with suitable example in C language.**

Logical Operators:
Used to perform logical operations, typically on boolean values (expressions that evaluate to true or false). They are mostly used in control flow statements like if, while etc.
We have 3 types of logical opertors in C:

1. Logical AND : &&
2. Logical OR : ||
3. Logical NOT : !

&& (logical AND): Returns true if both operands are non-zero (i.e., true), otherwise returns false.
|| (logical OR): Returns true if at least one operand is non-zero (i.e., true).
! (logical NOT): Inverts the boolean value (i.e., turns true to false and false to true)

Example:

```
#include<stdio.h>
int main() {
        int a=5,b=10,c=50;
        if(a<b && a*b==c) {
                printf("%d * %d equal to %d",a,b,c);
        }
        else {
                printf("%d * %d not equal to %d",a,b,c);
        }
}
```

Here,
a=5
b=10

c=50
In the IF statement we are checking if both (a<b) and (a*b==c) conditions are true.
If the condition is true, then the print statement : 5 * 10 is equal to 50 will be executed
Else condition is false, then the print statement : 5 * 10 is not equal to 50 will be executed

Similarly,
For OR operator || ,
It will check if either conditions are true
Example:
if(a>b || a*b==c) : since 2nd condition is true , the if condition will be true

For Not Operator ! ,
It will reverse/invert the boolean value (1 or 0) returned by the condition
if(!(a>b) : returns true since a>b returns 0 but the ! operator inverts it to 1

22.a. Write a C program to find whether the given number is palindrome or not.
Note: Input - integer format.

Program to check if a number is palindrome or not

```
#include <stdio.h>
int isPalindrome(int num) {
    int original = num;
    int reversed = 0;
    // Reverse the number
    while (num > 0) {
        int digit = num % 10;
        reversed = reversed * 10 + digit;
        num /= 10;
    }
    // Check if the reversed number matches the original
    return (original == reversed);
}


int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (isPalindrome(num)) {
        printf("%d is a palindrome.\n", num);
    }
    else {
        printf("%d is not a palindrome.\n", num);
    }
    return 0;
}
```

(OR)

b. Write a C program to perform matrix addition using pointers with necessary conditions.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
	int rows, cols;
	printf("Enter the number of rows: ");
	scanf("%d", &rows);
	printf("Enter the number of columns: ");
	scanf("%d", &cols);

	// Dynamically allocate memory for matrices A, B, and result
	int **A = (int **)malloc(rows * sizeof(int *));
	int **B = (int **)malloc(rows * sizeof(int *));
	int **result = (int **)malloc(rows * sizeof(int *));

	for (int i = 0; i < rows; i++) {
		A[i] = (int *)malloc(cols * sizeof(int));
		B[i] = (int *)malloc(cols * sizeof(int));
		result[i] = (int *)malloc(cols * sizeof(int));
	}
	// Input elements for matrix A
	printf("Enter elements of matrix A:\n");
	for (int i = 0; i < rows; i++) {
		for (int j = 0; j < cols; j++) {
			printf("Enter A[%d][%d]: ", i + 1, j + 1);
			scanf("%d", &A[i][j]);
		}
	}
	// Input elements for matrix B
	printf("Enter elements of matrix B:\n");
	for (int i = 0; i < rows; i++) {
		for (int j = 0; j < cols; j++) {
			printf("Enter B[%d][%d]: ", i + 1, j + 1);
			scanf("%d", &B[i][j]);
		}
	}
	// Perform matrix addition: result = A + B
	for (int i = 0; i < rows; i++) {
		for (int j = 0; j < cols; j++) {
			result[i][j] = A[i][j] + B[i][j];
		}
	}
	// Output the result of matrix addition
	printf("\nResultant matrix (A + B):\n");
	for (int i = 0; i < rows; i++) {
		for (int j = 0; j < cols; j++) {
			printf("%d ", result[i][j]);
		}
		printf("\n");
	}
	return 0;
```

}

23.a. Explain any four string functions with suitable example in C.

Four string functions in C are:

1. atoi()
   Converts string to number
   Eg:
   char st[]=”1925”;
   int a=atoi(st);
   print(“%d”,a)

   Output:
   1925
2. strlen()
   Returns length of string till null character ‘\0’
   Eg:
   char st[]=”Hello”;
   int n=strlen(st);
   print(“Length of string is %d”,n);

   Output:
   Length of string is 5
3. strcat()
   Concatenates two string and result is updated in 1st string
   Eg:
   char s1[]=”Hello “;
   char s2[]=”World”;
   strcat(s1,s2);
   printf(“%s\n %s”,s1,s2);

   Output:
   Hello World
   World
4. strcmp()
   Compares two string and returns integer value
   If i) 0 then both string equal
   ii)>0 then string1 is greater than string2
   iii)<0 then string1 is less than string2

(OR)

b. Is passing an array to a function comes under call-by-value (or) call-by- reference? Justify your answer with examples.

Passing an array to a function falls under Call by reference.
Call by Value :

Actual Parameters : values of variables sent to function
Formal Parameters : values of variables received by variables in the function

Call by Reference :

Actual Parameters : Address of variables sent to function

Formal Parameters : Address of variables received by pointer variables in the function

When talking about arrays in C,
They are but pointers which points to different addresses that contains a value

Array Declaration:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
	int A[6];
	return 0;
}
```

Here, we declare Array of size 6

But this same thing is possible with pointers.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
	int *A;
	int size=6;
	A=(int*)malloc(size*sizeof(int));

	int i;
	for(i=0;i<size;i++) {
		printf("A[%d] : ",i+1);
		scanf("%d",&A[i]);
	}
	return 0;
}
```

Here, we first declare a pointer and a size of 6
To allocate memory of (size * 4 bytes) we use the malloc() method in stdlib.h
This way we created a 1D array similar to the normal way we declare array
Now for accessing or putting values into the array,
We use the same [i] where i : index position

Example:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
	int *A;
	int size=6;
	A=(int*)malloc(size*sizeof(int));

	int i;
	for(i=0;i<size;i++) {
		printf("A[%d] : ",i+1);
		scanf("%d",&A[i]);
	}
```

```
        return 0;
}
```

Here, we declared a sum() function:
int sum (int*,int)
The 1st parameter is pointer
The 2nd parameter is integer

In the main() function,
int a[]={1,2,3};
int n=sizeof(a)/sizeof(a[0]); //calculate size of array in bytes

During function call : int s=sum(a,n);
Here, we are passing
1st Parameter as Array a
2nd Parameter as Integer n

When passing the array itself to sum() function, the pointer variable does not copy the values in the array.
When an array is passed to a function, it **decays** into a pointer to its first element. This means you are passing the address of the first element of the array, not the entire array.

But if only the 1st element's address is passed to the pointer in sum() function,
Then how does it access the other values in array
a={1,2,3}
arr points to a[0]

When you use the indexing syntax arr[i], it's equivalent to *(arr + i). This means "take the pointer arr, move it i elements forward in memory, and dereference it to get the value at that location".

Example:

```
#include <stdio.h>
void printElement(int* arr) {
        printf("arr[0] = %d\n", arr[0]);  // Access the first element
        printf("arr[1] = %d\n", arr[1]);  // Access the second element
}
int main() {
        int a[] = {1, 2, 3};
        printElement(a);  // Pass the array to the function
        return 0;
}
```

Explanation:

1. Passing the Array: When you pass the array a to the function printElement(), it decays into a pointer to the first element of the array (a is equivalent to &a[0]).
2. Accessing Elements: arr[0] is equivalent to *(arr + 0) which accesses the first element of the array (i.e., 1). arr[1] is equivalent to *(arr + 1) which accesses the second element of the array (i.e., 2).
3. Pointer Arithmetic: The pointer arr initially points to the first element of the array. When you use arr[1], you are moving the pointer to the second element by adding 1 to it. This is how pointer arithmetic allows you to access different elements in the array.

24.a. Explain about any four list functions used in Python programming with suitable examples.

Four List functions in Python:

1. append() :
   The append() function adds an element to the end of the list.

   Example:
   my_list = [1, 2, 3]
   my_list.append(4) # Adds 4 to the end of the list
   print(my_list) # Output: [1, 2, 3, 4]

   Here, we have a list my_list = [1,2,3]
   Using append() function we are adding 4 at the end of the list my_list

   Now, 4 is located at my_list[3] position

   Hence, append() modifies the original list by adding the specified element to the end. This operation does not return any value (returns None), so it's used to directly alter the list

2. remove() :
   The remove() function removes the first occurrence of a specified element from the list.

   Example:
   my_list = [1, 2, 3, 2, 4]
   my_list.remove(2) # Removes the first occurrence of 2
   print(my_list) # Output: [1, 3, 2, 4]

   Here, we have a list my_list = [1,2,3,2,4]
   Using remove() function, we are deleting the 1st occurrence number of element 2

   Now, my_list = [1,3,2,4]
   If we do : my_list.remove(2) again
   List becomes : my_list = [1,3,4]
   remove() searches for the first occurrence of the element and removes it from the list. If the element is not found, it raises a ValueError.

3. pop() :
   The pop() function removes and returns an element from the list at the specified index. If no index is provided, it removes and returns the last element.

   Example:
   my_list = [10, 20, 30, 40]
   popped_item = my_list.pop(2) # Removes the element at index 2
   print(popped_item) # Output: 30
   print(my_list) # Output: [10, 20, 40]

   Here, we have a list my_list = [10,20,30,40]
   Since pop() works by mentioning the index at which the element is located,
   Therefore to remove 30, we mention 2 (since indexing starts from 0)

Now, my_list = [10,20,40]
pop() removes and returns the item at the given index.
If no index is provided, it removes and returns the last item in the list. If you try to pop() from an empty list, it raises an IndexError

4. sort() :
   The sort() function sorts the elements of the list in ascending order by default. It can also sort in descending order or based on a custom key function.

   Example:
   my_list = [3, 1, 4, 2]
   my_list.sort() # Sorts in ascending order
   print(my_list) # Output: [1, 2, 3, 4]

   Here, we have an unsorted list my_list = [3,1,4,2]
   Using sort() function we sort this list in ascending order

   Now, my_list = [1,2,3,4]

   But if we want to sort the list in descending order, we do the following:
   my_list = [3, 1, 4, 2]
   my_list.sort(reverse=True) # Sorts in descending order
   print(my_list) # Output: [4, 3, 2, 1]

   reverse=True argument will tell sort() function to sort this list in descending order.

   Now, my_list = [4,3,2,1]
   sort() sorts the list in place, meaning it modifies the original list and does not return a new list.
   reverse=True sorts the list in descending order.
   But we can also use key argument :
   key allows you to define a custom sorting criterion (e.g., sorting by the length of strings)

   Eg 1 : Sorting by len
   words = ["banana", "pie", "apple", "cherry"]
   words.sort(key=len) # Sort by length of the string
   print(words)

   Output:
   ['banana', 'cherry', 'apple', 'pie']

   Eg 2 : Sorting by abs
   numbers = [-10, 5, 8, -3, 7]
   numbers.sort(key=abs) # Sort by absolute value
   print(numbers)

   Output: [-3, 5, 7, 8, -10]

(OR)

b. Write a Python program to print the prime numbers between the range [0, 100] without using built-in function.

```
def is_prime(n):
        c = 0
        for i in range (2, n+1):
                if (n % i == 0):
                        c+=1
return (c == 1)

for i in range (1, 101):
        if (is_prime(i)):
                print (i, end = ' ')
```

Here, we define a is_prime(n) function to check if n is prime number or not.
Since we are iterating from 1 to 100,
We check if 'i' is prime, and print i's value if its prime number in the output:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 53 59 61 67 71 73 79 83 89 97

25.a.i. Write a python program to create pandas dataframe using list.

```
import pandas as pd
# Example data
data = [ ['Alice', 24], ['Bob', 27], ['Charlie', 22] ]

df = pd.DataFrame(data, columns=['Name', 'Age']) # Create a DataFrame

# Display the DataFrame
print(df)
```

Output: —>

```
      Name  Age
0    Alice   24
1      Bob   27
2  Charlie   22
```

ii. Compare Numpy and Pandas.

Numpy	Pandas
Primarily used for numerical operations and handling arrays/matrices.	Primarily used for data manipulation and cleaning data.
Works with ndarray, a homogeneous, multi-dimensional array.	Works with DataFrame and Series, which are more flexible, supporting heterogeneous data types (e.g., strings, integers, floats).
Faster for numerical operations on large datasets because of its lower-level array operations.	Slightly slower than NumPy for numerical operations due to higher-level abstraction

Focuses on mathematical operations like linear algebra, statistics, and array manipulation.	Provides high-level operations for data manipulation, including grouping, merging of data etc.

(OR)

b.i. Compare the elements of two pandas series using python programming.

```
# Importing pandas library
import pandas as pd

# Creating 2 pandas Series
ps1 = pd.Series([2.5, 4, 6, 8, 10, 1.75, 40])
ps2 = pd.Series([1.5, 3, 5, 7, 10, 1.75, 20])

print("Series1:")
print(ps1)
print("\nSeries2:")
print(ps2)

# Compare the series using '==' and '!='
# Relational operators
print("\nCompare the elements of the two given Series:")
print("\nEqual:")
print(ps1 == ps2)
print("\nNot Equal:")
print(ps1 != ps2)
```

Output: —>

```
Series1:
0     2.50
1     4.00
2     6.00
3     8.00
4    10.00
5     1.75
6    40.00
dtype: float64

Series2:
0     1.50
1     3.00
2     5.00
3     7.00
4    10.00
5     1.75
6    20.00
dtype: float64

Compare the elements of the two given Series:

Equal:
0    False
1    False
2    False
3    False
4     True
5     True
6    False
dtype: bool

Not Equal:
0     True
1     True
2     True
3     True
4    False
5    False
6     True
dtype: bool
```

ii. Describe about Numpy.

NumPy (Numerical Python) is a powerful library in Python used for numerical and scientific computing. It is especially well-suited for working with large, multi-dimensional arrays and matrices of numerical data.

ndarray :
The core of NumPy is the ndarray, which is a fast, memory-efficient, and flexible array object that allows you to store and manipulate large datasets.

Python's default build-in list allows heterogeneous elements to be stored.
But in ndarray, you can only store homogeneous data type elements.

Eg:

```
arr = np.array([1, 'hello', 3.5])
print(arr)
```

Output: [‘1’,’hello’,’3.5]

Here, all elements are converted to string as it can’t store more than data type. Only single data type elements can be stored in this type of array.

NumPy arrays automatically convert mixed data types to the most general type to ensure homogeneity. In most cases, this results in the array being cast to a `string` type if there is any string in the data.

PART - C (1 × 15 = 15 Marks)
Answer **ANY ONE** Question

26.i. **Write a C program to read two integer values and print true if both the numbers end with the same digit, otherwise print false.**

Example: ***Input 1: 698***
Input 2: 768
Output: true

Ans.- **Pseudocode:**

START

1. Declare two integer variables `n1` and `n2`

2. Print "Input 1: "
3. Read the first integer from the user and store it in `n1`.

4. Print "Input 2: ".
5. Read the second integer from the user and store it in `n2`.

6. Compute the last digit of `n1` by calculating `n1 % 10` and store it in `r1`.
7. Compute the last digit of `n2` by calculating `n2 % 10` and store it in `r2`.

8. Print "Output: ".

9. If `r1` is equal to `r2`:
   a. Print "true".
Else:
   b. Print "false".

10. Exit the program.

END

Program:

```
#include <stdio.h>
int main() {
        int n1, n2;
        printf("Input 1: ");
        scanf("%d",&n1);
        printf("Input 2: ");
        scanf("%d",&n2);
        int r1 = n1 % 10;
        int r2 = n2 % 10;
        printf("\nOutput: ");
        if (r1 == r2) {
                printf("true");
        }
        else {
                printf("false");
        }
        return 0;
}
```

Explanation:

1. **Variable Declaration**:
   - n1 and n2 store the two integers input by the user.
   - r1 and r2 store the last digits of n1 and n2, respectively.
2. **Input**:
   - The program prompts the user to input two integers (n1 and n2), which are read using scanf.
3. **Extracting the Last Digit**:
   - The last digit of an integer can be determined using the modulo operator (%), which gives the remainder when the number is divided by 10.
   - For example, if n1 = 123, n1 % 10 yields 3.
4. **Comparison**:
   - The program compares the last digits of the two numbers (r1 and r2) using an if statement.
   - If the last digits are equal, the program prints "true". Otherwise, it prints "false".
5. **Output**:
   - The result ("true" or "false") indicates whether the last digits of the two numbers are the same.

ii. Write a C program to count the number of vowels present in a given sentence.

Ans.- Pseudocode:

START

1. Declare a character array ‘sen’ with a size of 1000.
2. Declare two integer variables: ‘count’ (initialize to 0) and ‘i’.

3. Print a message asking the user to "Enter a sentence: ".
4. Use ‘scanf’ with format specifier ‘%[^\n]’ to read the entire sentence from the user into the array ‘sen’.

5. Loop through the characters of ‘sen’ until the null terminator (‘\0’) is encountered:
   a. Convert the current character ‘sen[i]’ to lowercase using ‘tolower’ and store it in ‘ch’.
   b. Check if ‘ch’ is one of the vowels ('a', 'e', 'i', 'o', 'u'):
      i. If true, increment ‘count’ by 1.

6. After the loop ends, print the value of ‘count’ as the number of vowels in the sentence.

7. Exit the program.

END

Program:

```
#include <stdio.h>
#include <ctype.h>
int main() {
        char sen[1000];
        int count = 0, i;
        printf("Enter a sentence: ");
        scanf("%[^\n]", sen);
        for (i = 0; sen[i] != '\0'; i++) {
                char ch = tolower(sen[i]);
                if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
                        count++;
                }
        }
        printf("\nNumber of vowels: %d\n", count);
        return 0;
}
```

Explanation:

1. **Variable Declaration:**
   - sen[1000] is used to store the input sentence (up to 999 characters, plus the null terminator).
   - count is initialized to 0 and will track the number of vowels.
   - i is a loop counter.
2. **Input Sentence:**
   - The format specifier %[^\n] in scanf reads an entire line of input, stopping at a newline (\n).
3. **Processing Each Character:**
   - The for loop iterates through the characters of sen until it finds the null terminator (\0), indicating the end of the string.
   - Each character is converted to lowercase using the tolower() function to ensure case-insensitivity when checking for vowels.
4. **Counting Vowels:**
   - If the current character (ch) matches any of the vowels (a, e, i, o, u), the count variable is incremented.
5. **Output:**
   - After the loop, the program prints the total number of vowels stored in count.
6. **Return Statement:**
   - The program returns 0, indicating successful execution.

(OR)

27.i. Write a Python program to get key and value for two different lists respectively. Merge these lists into a dictionary and print the sorted dictionary with key order.

Ans.- Pseudocode:

START
1. Input the number of key-value pairs (n).
2. Initialize an empty list `keys`.
3. Initialize an empty list `values`.

4. For i = 1 to n:
   a. Prompt the user to input a key.
   b. Append the input key to the `keys` list.

5. For i = 1 to n:
   a. Prompt the user to input a value.
   b. Append the input value to the `values` list.

6. Use the `zip` function to combine `keys` and `values` into key-value pairs.
7. Convert the key-value pairs into a dictionary.

8. Sort the dictionary by its keys using the `sorted` function.
9. Print the sorted dictionary.

END

Program:

```
n = int(input("Enter the number of key-value pairs: "))

keys = []
print("Enter the keys:")
for i in range(n):
    k = input(f"Key {i + 1}: ")
    keys.append(k)

values = []
print("Enter the values:")
for i in range(n):
    v = input(f"Value {i + 1}: ")
    values.append(v)

dictionary = dict(zip(keys, values))

sorted_dictionary = dict(sorted(dictionary.items()))

print("Sorted dictionary:", sorted_dictionary)
```

Explanation:

1. **zip(keys, values)**: Combines the keys and values lists into key-value pairs.
2. **dict()**: Converts the key-value pairs into a dictionary.
3. **sorted(dictionary.items())**: Sorts the dictionary items (key-value pairs) based on the keys in ascending order.
4. **dict(sorted(...))**: Converts the sorted items back into a dictionary.

ii. Write a Python program to create a list for the user given elements and find the sum of elements in the list.

Ans.- Pseudocode:

```
START
1. Input the number of elements in the list (n).
2. Initialize an empty list `user_list`.

3. For i = 1 to n:
   a. Prompt the user to input an element.
   b. Convert the input to a float or integer.
   c. Append the element to the `user_list`.

4. Calculate the sum of the elements in `user_list` using the `sum` function.
5. Print the sum of the elements.
END
```

Program:

```
n = int(input("Enter the number of elements in the list: "))

user_list = []
print("Enter the elements: ")
for i in range(n):
    el = float(input(f"Element {i + 1}: "))
    user_list.append(el)

list_sum = sum(user_list)

print(f"The sum of the elements in the list is: {list_sum}”)
```

Explanation:

1. **input()**: Collects user input for the number of elements and each list element.
2. **float(input())**: Converts each element to a float to handle both integers and decimals.
3. **sum(user_list)**: Calculates the total sum of the list elements.
4. **Print statement**: Displays the calculated sum to the user.
