**II B.TECH - III Semester**

| 23CSE235L | PYTHON PROGRAMMING (SKILL ENHANCEMENT COURSE) | L T P C |
|---|---|---|
| | (Common to CSE, CSM, CAI, CSD) | 0 1 2 2 |

**PRE-REQUISITES:** Nil

**COURSE EDUCATIONAL OBJECTIVES:**

1. Introduce core programming concepts of Python programming language.
2. Demonstrate about Python data structures like Lists, Tuples, Sets and dictionaries
3. Implement Functions, Modules and Regular Expressions in Python Programming and to create practical and contemporary applications using these

**UNTI-I:**

History of Python Programming Language, Thrust Areas of Python, Installing Anaconda Python Distribution, Installing and Using Jupyter Notebook.

Parts of Python Programming Language: Identifiers, Keywords, Statements and Expressions, Variables, Operators, Precedence and Associativity, Data Types, Indentation, Comments, Reading Input, Print Output, Type Conversions, the type () Function and Is Operator, Dynamic and Strongly Typed Language.

Control Flow Statements: if statement, if-else statement, if...elif…else, Nested if statement, while Loop, for Loop, continue and break Statements, Catching Exceptions Using try and except Statement.

**SAMPLE EXPERIMENTS:**

1. Write a program to find the largest element among three Numbers.
2. Write a Program to display all prime numbers within an interval
3. Write a program to swap two numbers without using a temporary variable.
4. Demonstrate the following Operators in Python with suitable examples.

i) Arithmetic Operators ii) Relational Operators iii) Assignment Operatorsiv) Logical Operators

v) Bit wise Operators vi) Ternary Operator vii) Membership Operatorsviii) Identity Operators

5. Write a program to add and multiply complex numbers
6. Write a program to print multiplication table of a given number.

**UNIT-II:**

Functions: Built-In Functions, Commonly Used Modules, Function Definition and Calling the function, return Statement and void Function, Scope and Lifetime of Variables, Default Parameters, Keyword Arguments, *args and **kwargs, Command Line Arguments.

Strings: Creating and Storing Strings, Basic String Operations, Accessing Characters in String by Index Number, String Slicing and Joining, String Methods, Formatting Strings.

Lists: Creating Lists, Basic List Operations, Indexing and Slicing in Lists, Built-In Functions Used on Lists, List Methods, del Statement.

**SAMPLE EXPERIMENTS:**

7. Write a program to define a function with multiple return values.
8. Write a program to define a function using default arguments.
9. Write a program to find the length of the string without using any library functions.
10. Write a program to check if the substring is present in a given string or not.
11. Write a program to perform the given operations on a list: addition ii. Insertion iii. Slicing
12. Write a program to perform any 5 built-in functions by taking any list.

## UNIT-III:

Dictionaries: Creating Dictionary, Accessing and Modifying key:value Pairs in Dictionaries, Built-In Functions Used on Dictionaries, Dictionary Methods, del Statement.Tuples and Sets: Creating Tuples, Basic Tuple Operations, tuple() Function, Indexing and Slicing in Tuples, Built-In Functions Used on Tuples, Relation between Tuples and Lists, Relation between Tuples and Dictionaries, Using zip() Function, Sets, Set Methods, Frozenset.

## SAMPLE EXPERIMENTS:

13. Write a program to create tuples (name, age, address, college) for at least two members and concatenate the tuples and print the concatenated tuples.
14. Write a program to count the number of vowels in a string (No control flow allowed).
15. Write a program to check if a given key exists in a dictionary or not.
16. Write a program to add a new key-value pair to an existing dictionary.
17. Write a program to sum all the items in a given dictionary.

## UNIT-IV:

Files: Types of Files, Creating and Reading Text Data, File Methods to Read and Write Data, Reading and Writing Binary Files, Pickle Module, Reading and Writing CSV Files, Python os and os.path Modules.
Object-Oriented Programming: Classes and Objects, Creating Classes in Python, Creating Objects in Python, Constructor Method, Classes with Multiple Objects, Class Attributes Vs Data Attributes, Encapsulation, Inheritance, Polymorphism.

## SAMPLE EXPERIMENTS:

18. Write a program to sort words in a file and put them in another file. The output file should have only lower-case words, so any upper-case words from source must be lowered.
19. Python program to print each line of a file in reverse order.
20. Python program to compute the number of characters, words and lines in a file.
21. Write a program to create, display, append, insert and reverse the order of the items in the array.
22. Write a program to add, transpose and multiply two matrices.
23. Write a Python program to create a class that represents a shape. Include methods to calculate its area and perimeter. Implement subclasses for different shapes like circle, triangle, and square.

## UNIT-V:

Introduction to Data Science: Functional Programming, JSON and XML in Python, NumPy with Python, Pandas.

## SAMPLE EXPERIMENTS:

24. Python program to check whether a JSON string contains complex object or not.
25. Python Program to demonstrate NumPy arrays creation using array () function.
26. Python program to demonstrate use of ndim, shape, size, dtype.
27. Python program to demonstrate basic slicing, integer and Boolean indexing.
28. Python program to find min, max, sum, cumulative sum of array
29. Create a dictionary with at least five keys and each key represent value as a list where this list contains at least ten values and convert this dictionary as a pandas data frame and explore the data through the data frame as follows:

a) Apply head () function to the pandas data frame
b) Perform various data selection operations on Data Frame

30. Select any two columns from the above data frame, and observe the change in one attribute with respect to other attribute with scatter and plot operations in matplotlib

## COURSE OUTCOMES:

| On successful completion of the course the student will be | | POs related to COs |
|---|---|---|
| **CO1** | Showcase adept command of Python syntax, deftly utilizing variables, data types, control structures, functions, modules, and exception handling to engineer robust and efficient code solutions. (L4) | PO1 |
| **CO2** | Apply Python programming concepts to solve a variety of computational problems (L3) | PO2 |
| **CO3** | Understand the principles of object-oriented programming (OOP) in Python, including classes, objects, inheritance, polymorphism, and encapsulation, and apply them to design and implement Python programs (L3) | PO3 |
| **CO4** | Become proficient in using commonly used Python libraries and frameworks such as JSON, XML, NumPy, pandas (L2) | PO4 |
| **CO5** | Exhibit competence in implementing and manipulating fundamental data structures such as lists, tuples, sets, dictionaries (L3) | PO5 |

## REFERENCE BOOKS:

1. Gowrishankar S, Veena A., Introduction to Python Programming, CRC Press.
2. Python Programming, S Sridhar, J Indumathi, V M Hariharan, 2nd Edition, Pearson, 2024
3. Introduction to Programming Using Python, Y. Daniel Liang, Pearson.

## REFERENCE WEBSITE:

1. https://www.coursera.org/learn/python-for-applied-data-science-ai
2. https://www.coursera.org/learn/python?specialization=python#syllabus

## CO-PO MAPPING:

| CO-PO | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CO1** | 3 | - | - | - | - | - | - | - | - | - | - | - |
| **CO2** | - | 3 | - | - | - | - | - | - | - | - | - | - |
| **CO3** | - | - | 3 | - | - | - | - | - | - | - | - | - |
| **CO4** | - | - | - | 3 | - | - | - | - | - | - | - | - |
| **CO5** | - | - | - | - | 3 | - | - | - | - | - | - | - |
| **CO6** | - | - | - | - | - | - | - | 3 | - | - | - | - |
| **CO7** | - | - | - | - | - | - | - | - | 3 | - | - | - |
| **CO8** | - | - | - | - | - | - | - | - | - | 3 | - | - |
| **CO9** | - | - | - | - | - | - | - | - | - | - | - | 3 |
| **CO*** | **3** | **3** | **3** | **3** | **3** | **-** | **-** | **3** | **3** | **3** | **-** | **3** |

**1. Write a program to find the largest element among three Numbers.**

**# Python program to find the largest number among the three input numbers**

**# change the values of num1, num2 and num3**
**# for a different result**
**num1 = 10**
**num2 = 14**
num3 = 12

```
# uncomment following lines to take three numbers from user
#num1 = float(input("Enter first number: "))
#num2 = float(input("Enter second number: "))
#num3 = float(input("Enter third number: "))

if (num1 >= num2) and (num1 >= num3):
   largest = num1
elif (num2 >= num1) and (num2 >= num3):
   largest = num2
else:
   largest = num3

print("The largest number is", largest)
```

OUTPUT: The largest number is 14

2. Write a Program to display all prime numbers within an interval

```
# Python program to print all
# prime number in an interval


def prime(x, y):
    prime_list = []
    for i in range(x, y):
        if i == 0 or i == 1:
            continue
        else:
            for j in range(2, int(i/2)+1):
                if i % j == 0:
                    break
            else:
                prime_list.append(i)
    return prime_list


# Driver program
starting_range = 2
```

```
ending_range = 7
lst = prime(starting_range, ending_range)
if len(lst) == 0:
    print("There are no prime numbers in this range")
else:
    print("The prime numbers in this range are: ", lst)
```

OUTPUT: The prime numbers in this range are:  [2, 3, 5]

3. Write a program to swap two numbers without using a temporary variable.

```
def swap_numbers_arithmetic(a, b):
    print(f"Before swapping: a = {a}, b = {b}")

    # Swapping using arithmetic operations
    a = a + b
    b = a - b
    a = a - b

    print(f"After swapping: a = {a}, b = {b}")

# Example usage
swap_numbers_arithmetic(5, 10)
```

OUTPUT : Before swapping: a = 5, b = 10
After swapping: a = 10, b = 5

4. Demonstrate the following Operators in Python with suitable examples.

i) Arithmetic Operators

```
a = 10
b = 5

# Addition
print("a + b =", a + b)  # 15

# Subtraction
print("a - b =", a - b)  # 5

# Multiplication
print("a * b =", a * b)  # 50

# Division
print("a / b =", a / b)  # 2.0 (float)

# Floor Division
```

```
print("a // b =", a // b)  # 2 (integer division)

# Modulus
print("a % b =", a % b)  # 0

# Exponentiation
print("a ** b =", a ** b)  # 100000
```
OUTPUT : a + b = 15
a - b = 5
a * b = 50
a / b = 2.0
a // b = 2
a % b = 0
a ** b = 100000

ii) Relational Operators
```
# Define some variables
a = 20
b = 15
c = 20

# Equal to
print("a == b:", a == b)  # False
print("a == c:", a == c)  # True

# Not equal to
print("a != b:", a != b)  # True
print("a != c:", a != c)  # False

# Greater than
print("a > b:", a > b)    # True
print("b > c:", b > c)    # False

# Less than
print("a < b:", a < b)    # False
print("b < c:", b < c)    # True

# Greater than or equal to
print("a >= b:", a >= b)  # True
print("b >= c:", b >= c)  # False

# Less than or equal to
print("a <= b:", a <= b)  # False
print("c <= a:", c <= a)  # True
```
OUTPUT:
a == b: False

a == c: True
a != b: True
a != c: False
a > b: True
b > c: False
a < b: False
b < c: True
a >= b: True
b >= c: False
a <= b: False
c <= a: True

iii) Assignment Operators

```
x = 10

# Assignment
print("x =", x)

# Add and Assign
x += 5
print("x += 5:", x)  # 15

# Subtract and Assign
x -= 3
print("x -= 3:", x)  # 12

# Multiply and Assign
x *= 2
print("x *= 2:", x)  # 24

# Divide and Assign
x /= 4
print("x /= 4:", x)  # 6.0 (float)

# Modulus and Assign
x %= 3
print("x %= 3:", x)  # 0.0 (float)

# Exponentiate and Assign
x **= 2
print("x **= 2:", x)  # 0.0 (float)
```

OUTPUT:

x = 10
x += 5: 15
x -= 3: 12

```
x *= 2: 24
x /= 4: 6.0
x %= 3: 0.0
x **= 2: 0.0
```

iv) Logical Operators

```
x = True
y = False

# AND
print("x and y:", x and y)  # False

# OR
print("x or y:", x or y)   # True

# NOT
print("not x:", not x)     # False
```

OUTPUT:

```
x and y: False
x or y: True
not x: False
```

v) Bit wise Operators

```
a = 10  # 1010 in binary
b = 4   # 0100 in binary

# AND
print("a & b =", a & b)  # 0 (0000 in binary)

# OR
print("a | b =", a | b)  # 14 (1110 in binary)

# XOR
print("a ^ b =", a ^ b)  # 14 (1110 in binary)

# NOT
print("~a =", ~a)        # -11 (inverts all bits of a)

# Left Shift
print("a << 1 =", a << 1)  # 20 (10100 in binary)

# Right Shift
print("a >> 1 =", a >> 1)  # 5 (0101 in binary)
```

OUTPUT:

a & b = 0
a | b = 14
a ^ b = 14
~a = -11
a << 1 = 20
a >> 1 = 5

vi) Ternary Operator

```
# Example 1: Basic Ternary Operator
number = 10
result = "Positive" if number > 0 else "Negative"
print("The number is:", result) # Output: The number is: Positive

# Example 2: User Input
age = int(input("Enter your age: "))
status = "Minor" if age < 18 else "Adult"
print(f"You are an {status}.")

# Example 3: Nested Ternary Operators
score = 85
grade = "A" if score >= 90 else "B" if score >= 80 else "C" if score >= 70 else "D"
print(f"Grade: {grade}")
```

OUTPUT:

```
The number is: Positive
Enter your age: 55
You are an Adult.
Grade: B
```

vii) Membership Operators

**IN**

```
person = {"name": "Alice", "age": 25, "city": "New York"}

# Check if a key exists in the dictionary
print("'name' in person:", 'name' in person)  # True

# Check if a key does not exist in the dictionary
print("'email' in person:", 'email' in person)  # False
```

OUTPUT:

```
'name' in person: True
'email' in person: False
```

**NOT IN**

```
person = {"name": "Alice", "age": 25, "city": "New York"}

# Check if a key does not exist in the dictionary
print("'email' not in person:", 'email' not in person)  # True
```

```
# Check if a key exists in the dictionary
print("'name' not in person:", 'name' not in person)  # False
```

OUTPUT:

```
'email' not in person: True
'name' not in person: False
```

viii) Identity Operators

**`is` Operator**

```
str1 = "hello"
str2 = "hello"

# Check if str1 and str2 point to the same object
print("str1 is str2:", str1 is str2)  # True (strings are interned and reused)

str3 = "world"
print("str1 is str3:", str1 is str3)  # False (different strings)
```

**OUTPUT:**

```
str1 is str2: True
str1 is str3: False
```

**`is NOT` Operator**

```
str1 = "test"
str2 = "test"

# Check if str1 and str2 do not point to the same object
print("str1 is not str2:", str1 is not str2)  # False (same object reference)

str3 = "different"
print("str1 is not str3:", str1 is not str3)  # True (different strings)
```

OUTPUT:

```
str1 is not str2: False
str1 is not str3: True
```

5. Write a program to add and multiply complex numbers

```
def add_complex(c1, c2):
    return c1 + c2

# Function to multiply two complex numbers
def multiply_complex(c1, c2):
    return c1 * c2

# Example usage
def main():
    # Define two complex numbers
```

```
    complex1 = complex(3, 4)  # 3 + 4j
    complex2 = complex(1, 2)  # 1 + 2j

    # Add the complex numbers
    sum_complex = add_complex(complex1, complex2)
    print(f"Sum of {complex1} and {complex2} is {sum_complex}")

    # Multiply the complex numbers
    product_complex = multiply_complex(complex1, complex2)
    print(f"Product of {complex1} and {complex2} is {product_complex}")

# Run the main function
if __name__ == "__main__":
    main()
```

OUTPUT:

```
Sum of (3+4j) and (1+2j) is (4+6j)
Product of (3+4j) and (1+2j) is (-5+10j)
```

6. Write a program to print multiplication table of a given number.

```
def print_multiplication_table(number, up_to=10):
    """
    Prints the multiplication table for the given number up to the specified range.

    :param number: The number for which to print the multiplication table.
    :param up_to: The range up to which to print the table (default is 10).
    """
    print(f"Multiplication Table for {number}:")
    for i in range(1, up_to + 1):
        print(f"{number} x {i} = {number * i}")

def main():
    # Get user input
    try:
        number = int(input("Enter a number to generate its multiplication table: "))
        # Optionally, get user-defined range for the multiplication table
        up_to = int(input("Enter the range up to which you want the table (default is 10): ") or 10)

        # Print the multiplication table
        print_multiplication_table(number, up_to)
    except ValueError:
        print("Please enter a valid integer.")

# Run the main function
if __name__ == "__main__":
    main()
```

OUTPUT:

Enter a number to generate its multiplication table: 8

Enter the range up to which you want the table (default is 10): 10
Multiplication Table for 8:
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80

7. Write a program to define a function with multiple return values.

```
# Function that returns multiple values
def get_values():
    integer_value = 42
    float_value = 3.14
    string_value = "Hello, Python!"
    return integer_value, float_value, string_value

# Main code to call the function and use the returned values
def main():
    int_val, float_val, str_val = get_values()

    print(f"Integer Value: {int_val}")
    print(f"Float Value: {float_val}")
    print(f"String Value: {str_val}")

# Entry point of the program
if __name__ == "__main__":
    main()
```

**OUTPUT:**

Integer Value: 42
Float Value: 3.14
String Value: Hello, Python!

8. Write a program to define a function using default arguments.

```
def greet(name, greeting="Hello", punctuation="!"):
    """
    Function to greet a person with a customizable greeting message.

    Parameters:
```

```
    - name (str): The name of the person to greet.
    - greeting (str): The greeting word (default is "Hello").
    - punctuation (str): The punctuation mark to end the greeting (default is "!").

    Returns:
    - str: The complete greeting message.
    """
    return f"{greeting}, {name}{punctuation}"

def main():
    # Call function with all arguments
    message1 = greet("Alice", "Hi", "?")
    print(message1)  # Output: Hi, Alice?

    # Call function with default greeting and punctuation
    message2 = greet("Bob")
    print(message2)  # Output: Hello, Bob!

    # Call function with default punctuation
    message3 = greet("Charlie", "Hey")
    print(message3)  # Output: Hey, Charlie!

if __name__ == "__main__":
    main()
```

**OUTPUT:**

```
Hi, Alice?
Hello, Bob!
Hey, Charlie!
```

9. Write a program to find the length of the string without using any library functions.

```
def string_length(s):
    """
    Function to calculate the length of a string without using library functions.

    Parameters:
    - s (str): The input string.

    Returns:
    - int: The length of the string.
    """
    count = 0
    for char in s:
        count += 1
    return count

def main():
    test_string = "Hello, World!"
    length = string_length(test_string)
```

```
    print(f"The length of the string '{test_string}' is {length}.")

if __name__ == "__main__":
    main()
```

**OUTPUT:**

The length of the string 'Hello, World!' is 13.

10. Write a program to check if the substring is present in a given string or not.

```
def is_substring_present(main_string, substring):
    """
    Function to check if a substring is present in a main string without using built-in substring functions.

    Parameters:
    - main_string (str): The string in which to search for the substring.
    - substring (str): The substring to search for.

    Returns:
    - bool: True if the substring is found, False otherwise.
    """
    main_length = len(main_string)
    sub_length = len(substring)

    # Iterate through the main string
    for i in range(main_length - sub_length + 1):
        # Check if the current substring matches
        if main_string[i:i + sub_length] == substring:
            return True

    return False

def main():
    main_string = "Hello, World!"
    substring = "World"

    if is_substring_present(main_string, substring):
        print(f"The substring '{substring}' is present in the main string.")
    else:
        print(f"The substring '{substring}' is not present in the main string.")

if __name__ == "__main__":
    main()
```

**OUTPUT:**

The substring 'World' is present in the main string.

11. Write a program to perform the given operations on a list: i. Addition ii. Insertion iii. slicing

```
def add_element(lst, element):
    """
    Function to add an element to the end of the list.

    Parameters:
    - lst (list): The list to which the element will be added.
    - element (any type): The element to add to the list.
    """
    lst.append(element)
    print(f"List after adding '{element}': {lst}")

def insert_element(lst, index, element):
    """
    Function to insert an element at a specific position in the list.

    Parameters:
    - lst (list): The list to which the element will be added.
    - index (int): The position at which to insert the element.
    - element (any type): The element to add to the list.
    """
    if 0 <= index <= len(lst):
        lst.insert(index, element)
        print(f"List after inserting '{element}' at index {index}: {lst}")
    else:
        print("Index out of bounds.")

def slice_list(lst, start, end):
    """
    Function to slice the list from start to end indices.

    Parameters:
    - lst (list): The list to be sliced.
    - start (int): The starting index of the slice.
    - end (int): The ending index of the slice (exclusive).

    Returns:
    - list: The sliced portion of the list.
    """
    sliced = lst[start:end]
    print(f"Sliced list from index {start} to {end}: {sliced}")
    return sliced

def main():
    # Initial list
    my_list = [1, 2, 3, 4, 5]

    print(f"Initial list: {my_list}")
```

```
    # Add an element
    add_element(my_list, 6)

    # Insert an element
    insert_element(my_list, 2, 99)

    # Slice the list
    sliced_part = slice_list(my_list, 1, 4)

if __name__ == "__main__":
    main()
```

OUTPUT:

```
Initial list: [1, 2, 3, 4, 5]
List after adding '6': [1, 2, 3, 4, 5, 6]
List after inserting '99' at index 2: [1, 2, 99, 3, 4, 5, 6]
Sliced list from index 1 to 4: [2, 99, 3]
```

12. Write a program to perform any 5 built-in functions by taking any list.

```
def add_element(lst, element):
    """
    Function to add an element to the end of the list.

    Parameters:
    - lst (list): The list to which the element will be added.
    - element (any type): The element to add to the list.
    """
    lst.append(element)
    print(f"List after adding '{element}': {lst}")

def insert_element(lst, index, element):
    """
    Function to insert an element at a specific position in the list.

    Parameters:
    - lst (list): The list to which the element will be added.
    - index (int): The position at which to insert the element.
    - element (any type): The element to add to the list.
    """
    if 0 <= index <= len(lst):
        lst.insert(index, element)
        print(f"List after inserting '{element}' at index {index}: {lst}")
    else:
        print("Index out of bounds.")

def slice_list(lst, start, end):
    """
    Function to slice the list from start to end indices.
```

```
    Parameters:
    - lst (list): The list to be sliced.
    - start (int): The starting index of the slice.
    - end (int): The ending index of the slice (exclusive).

    Returns:
    - list: The sliced portion of the list.
    """
    sliced = lst[start:end]
    print(f"Sliced list from index {start} to {end}: {sliced}")
    return sliced

def main():
    # Initial list
    my_list = [1, 2, 3, 4, 5]

    print(f"Initial list: {my_list}")

    # Add an element
    add_element(my_list, 6)

    # Insert an element
    insert_element(my_list, 2, 99)

    # Slice the list
    sliced_part = slice_list(my_list, 1, 4)

if __name__ == "__main__":
    main()
```

OUTPUT:

Initial list: [1, 2, 3, 4, 5]
List after adding '6': [1, 2, 3, 4, 5, 6]
List after inserting '99' at index 2: [1, 2, 99, 3, 4, 5, 6]
Sliced list from index 1 to 4: [2, 99, 3]

13. Write a program to create tuples (name, age, address, college) for at least two membersand concatenate the tuples and print the concatenated tuples.

```
# Creating tuples for two members

member1 = ('Alice', 21, '123 Main St, City A', 'XYZ College')

member2 = ('Bob', 22, '456 Elm St, City B', 'ABC University')


# Concatenating the tuples

combined_tuple = member1 + member2


# Printing the concatenated tuple

print("Concatenated Tuple:", combined_tuple)
```

## output

Concatenated Tuple:

Concatenated Tuple: ('Alice', 21, '123 Main St, City A', 'XYZ College', 'Bob', 22, '456 Elm St, City B', 'ABC University')

14. Write a program to count the number of vowels in a string (No control flow allowed).

```
def count_vowels(s):
    return len(list(filter(lambda x: x in 'aeiouAEIOU', s)))

# Example usage
input_string = "Hello, World!"
print(count_vowels(input_string))
```

Output:

3

15. Write a program to check if a given key exists in a dictionary or not

```
# Sample dictionary
my_dict = {
    'name': 'Alice',
    'age': 25,
    'address': '123 Main St, City A',
    'college': 'XYZ University'
}
# Ask user for the key to check
key = input("Enter the key to check: ")

# Check if the key exists in the dictionary
if key in my_dict:
    print(f"The key '{key}' exists in the dictionary.")
else:
    print(f"The key '{key}' does not exist in the dictionary.")
```

## output

```
Enter the key to check: age
The key 'age' exists in the dictionary.
```

16. Write a program to add a new key-value pair to an existing dictionary

```
# Existing dictionary
my_dict = {
    'name': 'Alice',
    'age': 30,
    'city': 'New York'
}

# Display the existing dictionary
print("Original dictionary:")
print(my_dict)

# New key-value pair to add
new_key = 'occupation'
new_value = 'Engineer'

# Adding the new key-value pair to the dictionary
my_dict[new_key] = new_value

# Display the updated dictionary
print("\nUpdated dictionary:")
print(my_dict)
```

## OUTPUT :

Original dictionary:

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

Updated dictionary:

{'name': 'Alice', 'age': 30, 'city': 'New York', 'occupation': 'Engineer'}

17. Write a program to sum all the items in a given dictionary

```
# Define a dictionary with numeric values
my_dict = {
    'apple': 10,
    'banana': 20,
    'cherry': 30,
    'date': 40,
    'elderberry': 50
}

# Sum all the values in the dictionary
total_sum = sum(my_dict.values())

# Print the result
print("Sum of all items in the dictionary:", total_sum)
```

## OUTPUT :

```
Sum of all items in the dictionary: 150
```

18.Write a program to sort words in a file and put them in another file. The output file should have only lower-case words, so any upper-case words from source must be lowered.

```
def sort_words(input_filename, output_filename):
    # Read the words from the input file
    with open(input_filename, 'r') as infile:
        # Read the file contents, split into words, and convert to lowercase
        words = infile.read().split()
        words = [word.lower() for word in words]

    # Sort the words
    words.sort()

    # Write the sorted words to the output file
    with open(output_filename, 'w') as outfile:
        for word in words:
            outfile.write(word + '\n')

# Example usage:
input_filename = 'input.txt'
output_filename = 'output.txt'

sort_words(input_filename, output_filename)
```

INPUT

Banana apple Cherry date elderberry

## OUTPUT :

apple

banana

cherry

date

elderberry

19.Python program to print each line of a file in reverse order.

```
# Function to print each line of the file in reverse order
def reverse_lines_in_file(input_file):
    # Open the input file for reading
    with open(input_file, 'r') as file:
        # Read each line of the file
        for line in file:
            # Reverse the line and print it (strip is used to remove the trailing newline)
            print(line.strip()[::-1])

# Input file path
input_file = 'sample.txt'

# Call the function to reverse the lines and print
reverse_lines_in_file(input_file)
```

INPUT

Hello, World!

Python is great.

Reverse these lines.

## OUTPUT :

!dlroW ,olleH

.taerg si nohtyP

.senil eseht esreveR

20.Python program to compute the number of characters, words and lines in a file

```
# Function to compute the number of characters, words, and lines in a file
def compute_file_statistics(input_file):
    # Initialize counters for characters, words, and lines
    char_count = 0
    word_count = 0
    line_count = 0

    # Open the file for reading
    with open(input_file, 'r') as file:
        # Read each line in the file
        for line in file:
            line_count += 1  # Increment line count
            char_count += len(line)  # Count characters in the line
            word_count += len(line.split())  # Count words by splitting the line into words

    # Print the results
    print(f"Number of lines: {line_count}")
    print(f"Number of words: {word_count}")
    print(f"Number of characters: {char_count}")

# Input file path
input_file = 'sample.txt'
```

```
# Call the function to compute file statistics
compute_file_statistics(input_file)
```

INPUT :

```
Hello, World!
This is a sample text file.
It contains multiple lines.
```

## OUTPUT :

```
Number of lines: 3
Number of words: 10
Number of characters: 66
```

21.Write a program to create, display, append, insert and reverse the order of the items in the array.

```
# Function to create, display, append, insert, and reverse an array

def array_operations():

    # 1. Create an array (list)

    arr = [10, 20, 30, 40, 50]


    # 2. Display the original array

    print("Original Array:", arr)


    # 3. Append a new item to the array

    arr.append(60)

    print("Array after appending 60:", arr)


    # 4. Insert a new item at a specific position (e.g., insert 25 at index 2)

    arr.insert(2, 25)

    print("Array after inserting 25 at index 2:", arr)


    # 5. Reverse the order of the array

    arr.reverse()

    print("Array after reversing the order:", arr)


# Call the function to perform the operations

array_operations()
```

## OUTPUT :

Original Array: [10, 20, 30, 40, 50]

Array after appending 60: [10, 20, 30, 40, 50, 60]

Array after inserting 25 at index 2: [10, 20, 25, 30, 40, 50, 60]

Array after reversing the order: [60, 50, 40, 30, 25, 20, 10]

22.Write a program to add, transpose and multiply two matrices.

```
# Function to add two matrices
def add_matrices(A, B):
    # Check if matrices are of the same size
    if len(A) != len(B) or len(A[0]) != len(B[0]):
        return "Error: Matrices must be of the same size for addition."

    # Initialize the result matrix with the same dimensions
    result = [[0 for _ in range(len(A[0]))] for _ in range(len(A))]

    # Perform element-wise addition
    for i in range(len(A)):
        for j in range(len(A[0])):
            result[i][j] = A[i][j] + B[i][j]

    return result

# Function to transpose a matrix
def transpose_matrix(A):
    # Transpose of matrix A
    result = [[A[i][j] for i in range(len(A))] for j in range(len(A[0]))]
    return result

# Function to multiply two matrices
def multiply_matrices(A, B):
```

```
    # Check if the number of columns in A is equal to the number of rows in B
    if len(A[0]) != len(B):
        return "Error: Number of columns in A must be equal to number of rows in B for multiplication."

    # Initialize the result matrix with dimensions (rows of A x columns of B)
    result = [[0 for _ in range(len(B[0]))] for _ in range(len(A))]

    # Perform matrix multiplication
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                result[i][j] += A[i][k] * B[k][j]

    return result

# Example matrices
A = [
    [1, 2, 3],
    [4, 5, 6]
]

B = [
    [7, 8, 9],
    [10, 11, 12],
    [13, 14, 15]
```

```
]

# Matrix Addition (if matrices are of the same size)
addition_result = add_matrices(A, B)
print("Matrix Addition Result:")
for row in addition_result:
    print(row)

# Matrix Transpose
transpose_result_A = transpose_matrix(A)
transpose_result_B = transpose_matrix(B)
print("\nTranspose of Matrix A:")
for row in transpose_result_A:
    print(row)

print("\nTranspose of Matrix B:")
for row in transpose_result_B:
    print(row)

# Matrix Multiplication
multiplication_result = multiply_matrices(A, B)
print("\nMatrix Multiplication Result:")
for row in multiplication_result:
    print(row)
```

## Output:

Adding matrices A and B:

Error: Matrices must be of the same size for addition.

Transposing matrix A:

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

Transposing matrix B:

[[7, 10, 13], [8, 11, 14], [9, 12, 15]]

Multiplying matrices A and B:

[[74, 88, 102], [148, 174, 200]]

23. Write a Python program to create a class that represents a shape. Include methods to calculate its area and perimeter. Implement subclasses for different shapes like circle,triangle, and square.

```
import math


# Base class representing a shape
class Shape:
    def area(self):
        raise NotImplementedError("Subclass must implement abstract method")

    def perimeter(self):
        raise NotImplementedError("Subclass must implement abstract method")

# Subclass representing a Circle
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * (self.radius ** 2)  # Area of a circle: πr^2

    def perimeter(self):
        return 2 * math.pi * self.radius  # Perimeter (circumference): 2πr

# Subclass representing a Triangle
```

```
class Triangle(Shape):
    def __init__(self, base, height, side1, side2, side3):
        self.base = base
        self.height = height
        self.side1 = side1
        self.side2 = side2
        self.side3 = side3

    def area(self):
        return 0.5 * self.base * self.height  # Area of a triangle: 1/2 * base * height

    def perimeter(self):
        return self.side1 + self.side2 + self.side3  # Perimeter: sum of all sides

# Subclass representing a Square
class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side ** 2  # Area of a square: side^2

    def perimeter(self):
        return 4 * self.side  # Perimeter of a square: 4 * side
```

```
# Function to display area and perimeter of a shape
def display_shape_info(shape):
    print(f"Area: {shape.area()}")
    print(f"Perimeter: {shape.perimeter()}")

# Example usage
if __name__ == "__main__":
    # Create objects for each shape
    circle = Circle(5)
    triangle = Triangle(3, 4, 3, 4, 5)
    square = Square(4)

    # Display information for Circle
    print("Circle:")
    display_shape_info(circle)
    print()

    # Display information for Triangle
    print("Triangle:")
    display_shape_info(triangle)
    print()

    # Display information for Square
    print("Square:")
    display_shape_info(square)
```

## output

Circle:

Area: 78.53981633974483

Perimeter: 31.41592653589793

Triangle:

Area: 6.0

Perimeter: 12

Square:

Area: 16

Perimeter: 16

## 24.Python program to check whether a JSON string contains complex object or not.

```
import json


# Function to check if a given object (dict or list) contains a complex object
def contains_complex_object(obj):
    # Check if the object is a dictionary or list, meaning it may contain complex objects
    if isinstance(obj, dict):
        # Iterate through the dictionary and check if any value is a complex object
        for value in obj.values():
            if isinstance(value, (dict, list)):  # Found a nested object or array
                return True
    elif isinstance(obj, list):
        # Iterate through the list and check if any element is a complex object
        for item in obj:
            if isinstance(item, (dict, list)):  # Found a nested object or array
                return True
    return False


# Function to check if JSON string contains a complex object
def is_complex_json(json_string):
    try:
        # Parse the JSON string into a Python object (dict or list)
        data = json.loads(json_string)


        # Check if the parsed object contains a complex object
```

```
        return contains_complex_object(data)

    except json.JSONDecodeError:
        # If JSON is invalid, return False
        print("Invalid JSON string")
        return False

# Test the function with different JSON strings
json_string_1 = '{"name": "Alice", "age": 30, "address": {"city": "Wonderland", "zipcode": "12345"}}'
json_string_2 = '{"name": "Alice", "age": 30, "friends": ["Bob", "Charlie"]}'
json_string_3 = '{"name": "Alice", "age": 30, "isEmployed": true}'
json_string_4 = '{"people": [{"name": "Alice"}, {"name": "Bob"}]}'

# Test case 1: Contains a nested object in "address"
print(is_complex_json(json_string_1))  # Output: True

# Test case 2: Contains an array in "friends"
print(is_complex_json(json_string_2))  # Output: True

# Test case 3: No nested object or array
print(is_complex_json(json_string_3))  # Output: False

# Test case 4: Contains a list of objects in "people"
print(is_complex_json(json_string_4))
```

## OUTPUT :

True

True

False

True

25.Python Program to demonstrate NumPy arrays creation using array () function.

```
import numpy as np

# Creating a 1D NumPy array using the array() function

array_1d = np.array([1, 2, 3, 4, 5])

print("1D Array:")

print(array_1d)


# Creating a 2D NumPy array (matrix) using the array() function

array_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print("\n2D Array:")

print(array_2d)


# Creating a 3D NumPy array using the array() function

array_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print("\n3D Array:")

print(array_3d)


# Creating a NumPy array from a tuple

tuple_data = (10, 20, 30, 40, 50)

array_from_tuple = np.array(tuple_data)

print("\nArray from Tuple:")

print(array_from_tuple)
```

## OUTPUT :

```
1D Array:

[1 2 3 4 5]


2D Array:

[[1 2 3]
 [4 5 6]
 [7 8 9]]


3D Array:

[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]


Array from Tuple:

[10 20 30 40 50]
```

26.Python program to demonstrate use of ndim, shape, size, dtype

```
import numpy as np

# Create a 2D numpy array

array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])


# ndim: Number of dimensions of the array

print("Number of dimensions (ndim):", array.ndim)


# shape: Shape of the array (rows, columns)

print("Shape of the array (shape):", array.shape)


# size: Total number of elements in the array

print("Total number of elements (size):", array.size)


# dtype: Data type of the elements in the array

print("Data type of the array elements (dtype):", array.dtype)
```

## OUTPUT :

```
Number of dimensions (ndim): 2

Shape of the array (shape): (3, 3)

Total number of elements (size): 9

Data type of the array elements (dtype): int64
```

27.Python program to demonstrate basic slicing, integer and Boolean indexing.

```
import numpy as np

# Create a 2D numpy array (3x3 matrix)

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])


# 1. Basic Slicing

# Slicing rows and columns using [start:stop:step] notation

print("Basic Slicing:")

print(arr[0:2, 1:3])  # Select rows 0 to 1 (not 2), and columns 1 to 2 (not 3)


# 2. Integer Indexing

# Accessing specific elements by indices

print("\nInteger Indexing:")

print(arr[1, 2])  # Access element at row 1, column 2 (5th element in a 0-based index)


# 3. Boolean Indexing

# Create a Boolean mask and use it to select elements

mask = arr > 5  # Mask where elements are greater than 5

print("\nBoolean Indexing:")

print(arr[mask])  # Select elements where the mask is True
```

## OUTPUT :

Basic Slicing:

[[2 3]

 [5 6]]

Integer Indexing:

6

Boolean Indexing:

[6 7 8 9]

28.Python program to find min, max, sum, cumulative sum of array

```
import numpy as np

# Create a 1D numpy array

arr = np.array([10, 20, 30, 40, 50])


# 1. Minimum element in the array

min_value = np.min(arr)

print("Minimum value:", min_value)


# 2. Maximum element in the array

max_value = np.max(arr)

print("Maximum value:", max_value)


# 3. Sum of all elements in the array

sum_value = np.sum(arr)

print("Sum of elements:", sum_value)


# 4. Cumulative sum of the array

cumulative_sum = np.cumsum(arr)

print("Cumulative sum:", cumulative_sum)
```

## OUTPUT :

Minimum value: 10

Maximum value: 50

Sum of elements: 150

Cumulative sum: [ 10  30  60 100 150]

29.Create a dictionary with at least five keys and each key represent value as a list where this list contains at least ten values and convert this dictionary as a pandas data frame and explore the data through the data frame as follows:

Apply head () function to the pandas data frame

a)Apply head () function to the pandas data frame

b)Perform various data selection operations on Data Frame

```
import pandas as pd


# Create a dictionary with 5 keys, each holding a list of 10 values

data = {

    'Name': ['John', 'Anna', 'Peter', 'Linda', 'James', 'Mary', 'Robert', 'Sophia', 'David', 'Emma'],

    'Age': [23, 34, 29, 45, 36, 50, 41, 22, 33, 28],

    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix', 'Philadelphia', 'San Antonio', 'San Diego', 'Dallas', 'San Jose'],

    'Salary': [55000, 62000, 72000, 45000, 54000, 78000, 67000, 55000, 64000, 71000],

    'Department': ['HR', 'Finance', 'Engineering', 'Marketing', 'Sales', 'IT', 'HR', 'Finance', 'Engineering', 'IT']

}

# Convert the dictionary into a pandas DataFrame

df = pd.DataFrame(data)


# a) Apply head() function to the DataFrame

print("First 5 rows of the DataFrame:")

print(df.head())
```

## OUTPUT :

First 5 rows of the DataFrame:

| | Name | Age | City | Salary | Department |
|---|---|---|---|---|---|
| 0 | John | 23 | New York | 55000 | HR |
| 1 | Anna | 34 | Los Angeles | 62000 | Finance |
| 2 | Peter | 29 | Chicago | 72000 | Engineering |
| 3 | Linda | 45 | Houston | 45000 | Marketing |
| 4 | James | 36 | Phoenix | 54000 | Sales |

## b) Perform various data selection operations on the DataFrame

```
# Select a single column (e.g., 'Age')
print("\nSelecting the 'Age' column:")
print(df['Age'])

# Select multiple columns (e.g., 'Name' and 'Salary')
print("\nSelecting 'Name' and 'Salary' columns:")
print(df[['Name', 'Salary']])

# Select rows using index range (e.g., first 3 rows)
print("\nSelecting first 3 rows:")
print(df.iloc[0:3])

# Filter rows based on a condition (e.g., Salary > 60000)
print("\nSelecting rows with Salary > 60000:")
print(df[df['Salary'] > 60000])

# Select rows using boolean conditions (e.g., Age > 30 and Department is 'IT')
print("\nSelecting rows with Age > 30 and Department is 'IT':")
print(df[(df['Age'] > 30) & (df['Department'] == 'IT')])

# Select a specific cell value (e.g., Salary of the 4th row)
print("\nSalary of the 4th row:")
print(df.at[3, 'Salary'])  # or df.iloc[3]['Salary']
```

## OUTPUT :

Selecting the 'Age' column:

0 23

1 34

2 29

3 45

4 36

5 50

6 41

7 22

8 33

9 28

Name: Age, dtype: int64

30. Select any two columns from the above data frame, and observe the change in one attribute with respect to other attribute with scatter and plot operations in matplotlib

```
import pandas as pd

import matplotlib.pyplot as plt


# Create a dictionary with 5 keys, each holding a list of 10 values

data = {

   'Name': ['John', 'Anna', 'Peter', 'Linda', 'James', 'Mary', 'Robert', 'Sophia', 'David', 'Emma'],

   'Age': [23, 34, 29, 45, 36, 50, 41, 22, 33, 28],

   'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix', 'Philadelphia', 'San Antonio', 'San Diego', 'Dallas', 'San Jose'],

   'Salary': [55000, 62000, 72000, 45000, 54000, 78000, 67000, 55000, 64000, 71000],

   'Department': ['HR', 'Finance', 'Engineering', 'Marketing', 'Sales', 'IT', 'HR', 'Finance', 'Engineering', 'IT']

}


# Convert the dictionary into a pandas DataFrame

df = pd.DataFrame(data)


# Select the columns 'Age' and 'Salary'

x = df['Age']

y = df['Salary']


# Create a scatter plot

plt.figure(figsize=(10, 5))

plt.scatter(x, y, color='blue', label='Salary vs Age', alpha=0.7)
```

```
# Adding title and labels
plt.title('Scatter Plot of Salary vs Age')
plt.xlabel('Age')
plt.ylabel('Salary')
plt.grid(True)
plt.legend()

# Display the scatter plot
plt.show()

# Create a line plot (just to observe trends)
plt.figure(figsize=(10, 5))
plt.plot(x, y, marker='o', color='green', linestyle='-', label='Salary vs Age')

# Adding title and labels
plt.title('Line Plot of Salary vs Age')
plt.xlabel('Age')
plt.ylabel('Salary')
plt.grid(True)
plt.legend()

# Display the line plot
plt.show()
```

## OUTPUT :

When you run the above code, you will see two plots displayed sequentially: