Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

School of Computing

Department of Computer Science and Engineering

ACADEMIC YEAR 2025 – 2026 (Summer Semester)

BONAFIDE CERTIFICATE

NAME: PIYUSH KUMAR RATRE

VTU NO: 29460

REG.NO: 24UECS0272

BRANCH: CSE

YEAR/SEM: 2nd / III sem

SLOT: S2 - 1L14

Certified that this is a bonafide record of work done by above student in the **"10211CS213 – Python Programming"** during the year 2025-2026.

SIGNATURE OF FACULTY INCHARGE **SIGNATURE OF INCHARGE**

Submitted for the Semester Model Examination held on_____at Vel Tech Rangarajan Dr. Sagunthala R & D Institute of Science and Technology.

EXAMINER 1 **EXAMINER 2**

INDEX

IYUSH KUMAR Class ..... Div. ..... Roll No. VTU29460

Name Veltech University Sub. Python Programming

Date	Particulars	Page No.	Signature
25/7/25	Task(1)-(a) Create a python program. (b) Create a python program to enter two number. (c) Create a python program to enter three number.	1-14	1/8/25
2. 1/8/25	Task(2)-(a) You are developing a Simple grade. (b) You are developing an educational program. (c) You are Working on a feature for a financial application.	14-25	8/8/25
3. 8/8/25	Task(3)-(a) You are tasked with developing a modular. (b) You are Working on a python project that requires.	25-37	22/8/25
4. 22/8/25	Task(4)-(a) You are Working on a python project. (b) You are tasked with Creating a python program. (c) You are tasked with Creating a python program.	38-44	29/8/25
5. 29/8/25	Task(5)-(a) At company stores. (b) You are developing a grade management System for a School.	45-53	12/9/25

S.No	Date	Particular	Pg.No	Sign
6	12/9/25	Task(6) – 6(1) Implement various text file operation (6.2) You have a text file containing logi of a System (6.2) You need to write a report containing the details.	54 - 63	19/9/25
7	19/9/25	Task(7.1) Utilizing 'Functions' Concepts is python programming (7.2) You are tasked with creating a small calculator application.	64 - 72	26/9/25
8	26/9/25	(8.1) Task: 8 Implement python generator and decorator (8.2) Imagine you are working on a messaging application.	73 - 81	10/10/25
9	10/10/25	Task (9.1): Implement Exceptions and Exception handing in Python (9.2) You are developing a python calculator program. (9.3) You are building a python application a determine	82 - 92	17/10/25
10	17/10/25	Task(10): 10.1 Use Matplotlib module for plotting in python. (10.2) Write a python programming to create	93 - 101	24/10/25
11	24/10/25	Task(11.1): Use Tkinter module for UI design. (11.2) Write a pytho GUI program to Create.	102 - 109	31/10/25
12.	31/10/25	Taks (12.1) Simulate Gaming Concept Using Pygame. (12.2) Write a python program to Develop a chess board using pygame.	110 - 121	31/10/25

Use Case I | 1/11/25 | | 122-126 |

Completed → 3/11/25

← Completed 3/11/25

Output :

Enter the first number : 1

Enter the Second number : 2

addition : 3
Subraction : -1
Product : 2
division : 0.5

Task - 1 : Running Python Script and Various Expression in an interactive interpreter.

Date: 25/7/25

Aim : To run python Sprit and Various in an interactive interpreter

@ Create a python program to enter two number and then performs and display the result of the following operation : Addition, Subraction, Multiplication and division.

Algorithm :

I.) Start
II. Get two numbers and Store it in Variable x & y
III. For addition do ; x+y and print it
IV. For Subraction do ; x-y and print it
V. For multiplication do ; x*y and print it
VI. For division do ; x/y and print it

Program

```
X = int (input ("Enter the first number:"))
Y = int (input ("Enter the Second number:"))
add = x+y
sub = x-y
pro = x*y
div = x/y
Print (" Addition :", add)
Print ("Subraction :", sub)
Print (" Multiplication :", pro)
Print (" Division ", div)
```

Output :

Enter the First number : 1

Enter the Second number : 2

Enter the third number : 3

@ 1 > 2 is false

1 < 2 is true

3 == 2 is false

1 != 2 is true

1 >= 2 is false

2 <= 1 is False

(b) Create a python program to enter two number and then perform and display the result of the two numbers and display the result relational expression >, <, ==, != , >=, <=

Algorithm :

I. Start

II. Get the input from the User and Store it in a, b, c

III. Perform the relational operation (i-e; >, <, ==, !=, >=, <=)

IV. Print result.

Program :

```
# initialising the Value of a,b,c
a = int (input ("Enter the first number: "))
b = int (input ("Enter the Second number: "))
c = int (input ("Enter the third number: "))

# using relational operation
Print (a ">", b, "is", a>b)
Print (a, "<", b, "is", a<b)
Print (c, "==", a "is", (==a)
Print (c, "!=", b "is" c != a)
Print (a, ">=", b, "is", a>=b)
Print (b, "<=", a, "is, b<=a)
```

Output :

Enter the first number : 1

Enter the Second number : 2

Enter the third number : 3

logical operations results

False

False

True

True

(c.) Create a python program to enter three number and then performs and display the results of the following logical operations : and , or, not ,

Algorithm :

I. Start

II. Get the input from the User

III. Perform the logical operation on the input

IV. Print result

V. Stop

Program :

```
# Performing logical operations
# Taking three number as input
a = int (input (" Enter the first number : "))
b = int (input ("Enter the Second number:"))
c = int (input(" Enter the Third number:"))
# Performing logical operation.
print ("\n logical operations results :"))
print ((a>b) and (b>c))
print ((a>b) or (b>c))
print (not (a>b))
print (not (b>c))
```

VEL TECH CSE	
X NO	1
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	
TOTAL (20)	18
N WITH DATE	

Result : Thus, the python program to run python Script and Various expression in a interactive interpreter was done Sucessfully and the output was Verified.

Output

Enter the Score : 50

the grade is F

Enter the Score : 90

the grade is A

Enter the Score : 75

the grade is C

Task - 2 Implement Conditional, Control and Looping Statements

Date: 1/8/25

Aim : To implement Conditional, Control and looping Statements using python.

a) You are developing a Simple grade management System for a School. The System needs to determine the grade of a Student based on their Score in a test the grading System follow these rules.

If the Score is 90 or above the grade is "A"

If the Score is 80 to 89, the grade is "B"

If the Score is 70 to 79, the grade is "C"

If the Score is 60 to 69, the grade is "D"

If the Score is below 60 the grade is "F"

Algorithm:

I. Start

II. Get the input marks from user

III With the use of an if - else if - Statement do

- If marks >= 90 print grade "A".
- If marks is between 80 to 89 print grade "B".
- If marks is between 70 to 79 print grad "C".
- If marks is between 60 to 69 print grade "D".
- If marks is below 60. Print Grade "F"

IV. Stop

Program :

```
Score = int (input ("Enter the Score:"))
if Score >= 90:
print ("grade is A")
elif (Score <= 89 and score >= 80):
print ("the grade is B")
elif (Score <= 89 and Score >= 80):
print ("The grade is B")
elif (Score <= 79 and score >= 70);
Print ("The grade is C")
elif (Score <= 69 and Score >= 60);
print ("The grade is D")
else
    print ("The grade is F")
```

Output :

The first 10 natural numbers are :

1
2
3
4
5
6
7
8
9
10

b. You are developing an educational program to help young students learn about natural numbers. One of the features of program is to display the first 10 natural numbers to the user. Write a python program that uses a for loop to print first 10 natural numbers

Algorithm :

I. Start
II. Display "the First 10 natural numbers are :")
III. Use loop for generating numbers
IV. Print Output
V. Stop

Program :

```
Print ("The first 10 natural numbers are :")
for i in range (1, 11):
    Print (i)
```

Output:

Enter the number : 23

The number of digits in 23 is : 2

Enter the number : 9848812

The number of digits in 9848812 is : 7

c. You are working on a feature for a financial application the involves user input. One of the requirement is to count the total number of digit in a number

Algorithm:

I. Start.
II. get the input from the user.
III. Convert the integer to string using str ( ).
IV. Use len function to find number of digits.
V. Print the output.

Program :

```
digit = int (input "Enter the number: ")
String = str (digit)
Count = len (String)
Print ("The number of digits in ", digit," is :", Count)
```

VEL TECH CSE	
EX NO	2
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	4
TOTAL (20)	
SIGN WITH DATE	

31/10

Result: Thus the python program to impliment Conditional Control and looping statements Was done Sucessfully.

Task 3 : Importing Python modules and packages
Date: 8/8/25 in python programming

Aim : To Write python demonstrating importing Python modules and packages.

a). You are tasked with developing a modular calculator application in Python. The calculator should support basic arithmetic operations: addition, subtraction, multiplication, and division. Each operation should be implemented in a separate module.

Algorithm :

I. Define function for addition, subtraction, multiplication, and division
II. Handle division by zero by raising an error if the divisor is zero.
III. Import the module (mymath) containing these functions.
IV. Initialize two numbers (a = 10, b = 5)
V. Call each function using mymath.<function name>(a,b).
VI. Print the results of all operations.

Program :

```
def add (a,b):
    return a+b
def Subtract (a, b):
    return a-b
def multiply (a,b):
    return a*b
def divide (a,b):
    if b == 0:
    raise Value Error("Cannot divide
```

Output :

a.) Addition : 15
Subtraction : 5
Multiplication : 50
division : 20

b.) Circle area (radius=7) : 153.938.
Rectangle (5 x 10) : 50
Triangle area (b=6 ; h=8) : 24.0

```
return a/b

import mymath
a = 10
b = 5
print ("Addition:", mymath.add (a,b))
print ("Subtraction :", mymath. Subtract (a,b))
print ("Multiplication :", mymath. multiply (a,b))
print ("Division:", mymath. divide (a,b))
```

Output :

b.) Circle area (radius = 7) : 153.9380

Rectangle (5x10) : 50

Triangle area (b = 6, h = 8) : 24.0

b. You are working on a Python project that requires you to perform various mathematical operation and geometric area calculations. To organize your code better, you decide to create a package named mypackage which includes sub packages pack1 and pack2 with two modules.

Algorithm:

I. Create mathfunctions.py module:

II. Create areafunctions.py module:

III. Create __init__.py files in pack1 and pack2:

IV. Create main.py:

V. Print the Output as expected.

Program:

1. Create the mathfunctions.py module

```
def add(a,b):
    return a+b
def Subtract(a,b):
    return a-b
def multiply(a,b):
    return a*b
def divide(a,b):
    if b==0:
        return "Error! Division by zero."
    return a/b
```

2. Create the areafunctions.py module

```
import math
def circle_area(radius):
    return math.pi*radius*radius
def rectangle_area(length, width):
    return length*width
def triangle_area(base, height):
    return 0.5*base*height
```

Output :

[10, 20, 30]

[10, 30]

[30]

[5, 8, 5, 19, 30, 89]

The minimum value is : 5

The maximum value is : 89

The sum is : 156

The average is : 26.0

2. (10, 'hello', 3.14, 'world')

10

hello

3.14

('Hello', 3.14)

(10, 'hello', 3.14)

(10, 'hello', 3.14, 'world', 5, 0.5)

Error -

3. {'Name': Alice', 'age': 30, 'City': 'Newyork'}

Alice

30

{'Name': 'James', 'age': 30, 'City': 'Newyork'}

{'Name': 'James', 'age' = 30}

KEY : name

KEY : Age

dict_items ([('name', 'james'), ('age', 30)])

3. Create __init__.py in each package folder (pack 1 and pack 2)

from. mathfunctions import add, Subtract, multiply, divide.

from. areafunctions import Circle_area, rectangle_area, triangle_area.

4. Create the main.py file

from pack import mathfunctions

from pack import areafunctions

Using math functions

print ("Addition :", mathfunction.add(10,5))

print ("Subtraction :", mathfunction. Subtract (10, 5))

print ("Multiplication :", mathfunctions. multiply (10, 5))

print ("Division :", mathfunctions. divide (10, 5))

Using area functions

print ("Circle Area (radius = 7): ", areafunctions. triangle_area (6, 8))

VEL TECH CSE	
EX NO	3
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	1
TOTAL (20)	14
SIGN WITH DATE	

3/10

Result : Thus, the program for Importing python modules and packages was successfully executed and the output was verified.

Task4. Use Various data types, List, Tuples
Date: 22/8/25 and Dictionary in python programming

Aim: To use various data types, List, Tuples and Dictionary in python programming

a. You are Working on a Python project that requires you to manage and manipulate a list of numbers.
Your task is to Create a python program that demonstrates the following list operations:

1. Add Elements: Add element to the list.
2. Remove Elements: Remove Specific element from the list.
3. Sort Elements: Sort the list in ascending and descending order.
4. Find Minimum and Maximum: Find the minimum and maximum elements in the list.
5. Calculate Sum and Average: Calculate the Sum and average of the elements in the list.

Alogrithm:

1. START
2. For adding elements to a list first Create a list with name "List" and assign the values within [] brackets, inorder to add a new value use the function append().
3. For removing a Specific element use

"pop (index value)" or remove (itemname)".

4. For Sorting the elements use "Sorted (list)" function.

5. For finding minimum value use "min(list)" and for maximum use "max(list)".

6. For Sum use function "Sum(list)" and for average use the formula "sum (list)/len (list)"

7. Print the Output.

8. End

Program:

```
# Add Elements: Add elements to list.
list = [10, 20]
a = 30

list.append (a)
print (list)
# Remove Element: Remove Specific Elements from the list.
list.~~remove~~ pop (I) # by index value
print (list)
list.remove (10) # by itemname
print (list)
# Sort Elements: Sort the list in ascending and descending order.
I = [5, 8, 9, 15, 30, 89]
print (Sorted (I))
# Find Minimum and Maximum: Find the minimum and maximum elements in the list.
print ("The minimum value is:", min (I))
print (" the maximum value is:", max (I))
# Calculate Sum and Average
print("The Sum ~~and~~ is:", Sum (I))
```

Output : 1

[10, 20, 30]

[10, 30]

[30]

[5, 8, 9, 15, 30, 89]

The minimum value is : 5

The maximum value is : 89

The Sum is : 158

The average is : 26.0

Print ("The average is :", ((Sum(L)/

b. You are tasked with Creating a Python program that showcases operations on tuples. Tuples are immutable sequences, Similar to lists but with the key difference that they cannot be Changed after Creation.

1. Create a Tuple : Define a tuple with elements of different data types (10, 'hello', 3.14 'World').

2. Access Elements : Access individual element and slices of the tuple.

3. Concatenate Tuples : Combine two tuples to Create a new tuple.

4. Immutable Nature : Attempt to modify elements of the tuple and handle the resulting error.

Algorithm :

1. Start
2. To Create a tuple use "tuple_name = (value)".
3. To access the elements of a tuple either use the index values (tuple_name (index_value)) or the tuple slicing tuple_name [start: end]).
4. To Concatenate tuples use the operator "+" (tuple 1 "+" tuple 2).
5. Try to modify the tuple elements by assigning the value directly like; tuple (index) = new_value, will result in an error as it is immutable
6. Print the Output
7. End.

Output : 2

(10, 'hello', 3.14, 'World')

10

hello

3.14

('hello', 3.14)

(10, 'hello', 3.14)

(10, 'hello', 3.14, 'World', 5, 0.5)

Error -

Program :

```
# Create a Tuple : Define a tuple with
                   element of different
  data types (10,'hello', 3.14, 'world')
  tuple = 10, 'hello', 3.14, 'World')
  print (tuple )
# Access Elements : Access individual elements
                    and slices of the tuple.
  for i in tuple.
      print(i)
  print (tuple [1:3] )
  print (tuple [:-1])
# Concatenate Tuples : Combine two tuples
                       to Create a new
  tuple
  t2 = (5, 0.5)
  t3 = tuple + t2
  print (t3)
# Immutable Nature : Attempt to modify
                     elements of the
  tuple and handle the resulting error
  tuple (3) = "PI" # ERROR
```

C. You are tasked with Creating a Python program that Showcases operations on dictionaries. Dictionaries in Python are unordered Collections of items. Each item is a pair Consisting of a key and a value.

1. Create a Dictionary : Define a dictionary with key-value pairs of different data types. ({'name': 'Alice', 'age': 30, 'City': 'New York'}).
2. Access Values : Access values using keys.
3. Modify Dictionary : Update values, add new key-value pairs, and remove existing pairs.
4. Iterate Over Dictionary : Use loops to iterate over keys or values.

Algorithm :

1. Start the program.
2. Define a dictionary with key-value pairs of different data types.
3. Retrieve values from the dictionary using their Corresponding keys.
4. Modify Dictionary.
5. Iterate over Dictionary.
6. Stop the program.

Program :

```
# Create a Dictionary: Define a dictionary
with key-value pairs of different
data types. ({'name': 'Alice', 'age':30,
'City': 'New York'}
dictionary = {'name': 'Alice', 'age': 30, 'City'
'New York'}
```

Output : 3

{'Name' : 'Alice', 'age' : 30, 'City' : 'Newyork'}

Alice

30

{'Name' : 'James', 'age' : 30, 'City' : 'Newyork'}

{'Name' : 'James', 'age' = 30 {

KEY : Name

KEY : Age

Print (dictionary)

Access values : Access values using keys

Print (dictionary ['name'])

print (dictionary ['age'])

Modify Dictionary : Update values, add new key-value pairs, and remove existing pairs.

dictionary ['name'] = "James"

print (dictionary)

dictionary .pop('city')

print (dictionary)

Iterate Over Dictionary : Use loops to iterate over keys or values.

for k in dictionary :

print ("KEY: ", K)

print (dictionary .items())

VEL TECH CSE	
EX NO	4
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	—
TOTAL (20)	11

WITH DATE

Result : Thus, Various data types, List, Tuples and Dictionary in python programming was used and verified sucessfully.

Output 5.1

{'id': 2, 'name': 'bob', 'department': 'Engineering'}

Task : 5 Implement Various Searching and Sorting
Date:- 29/8/25 Operations in python programming

Aim:

To Implement Various Searching and Sorting operations in python programming.

5.1. A Company stores employee records in a list of dictionaries, where each dictionary contains id, name, and department. Write a function find_employee_by_id that takes this list and a target employee with the matching ID, or None if no such employee is found.

Algorithm:

1. Input Definition:
2. Define the function Find_employee_by_id that take two parameters:
   a.) A list of dictionaries (employees), where each dictionary represents an employee record with keys id, name, and department.
   b.) An integer (target_id) representing the employee ID to be searched.
3. Iterate through the list:
   Use a for loop to iterate through each dictionary in the employees list.
4. Check for Matching ID:
   Within the loop, check if the id field of the current dictionary matches the target_id.
5. Return Matching Record:
   If a match is found, return the current dictionary.
6. Handle No Match:
   If the loop completes without finding a match

OUTPUT: 5.2

{'name': 'Alice', 'Score': 88}
{'name': 'Bob', 'Score': 95}
{'name': 'Charlie', 'Score': 75}
{'name': 'Diana', 'Score': 85}

After modification

{'name': 'Charle', 'Score': 75}
{'name': 'Diana', 'Score': 85}
{'name': 'Alice', 'Score': 88}
{'name': 'Bob', 'Score': 95}

```
, return None.
```

Program 5.1

```
def find_employee_by_id (employees, target_id):
        for employee in employees:
            if employee ['id'] == target_id:
                return employee
        return None

#Test the function
    employees = [
  {'id': 1, 'name': 'Alice', 'department': 'HR'},
  {'id': 2, 'name': 'Bob', 'department': 'Engineering'},
  {'id': 3, 'name': 'Charlie', 'department': 'Sales'},
]

print(find_employee_by_id (employees, 2)) # ~~Output:~~

#OUTPUT: {'id': 2, 'name': 'Bob', 'department':
'Engineering'}
```

5.2. You are developing a grade management System for a School. The System maintains a list of Student records, where each record is represented as a dictionary containing a Student's name and Score. The School needs to generate a report that displays Students' Scores in ascending order. Your task is to implement a feature that Sorts the Students records by their Scores using the Bubble Sort algorithm.

Algorithm:

1. Initialization:
   (i) Get the length of the Students list and Store it in n.

2.) Outer Loop:

(·) Iterate from i=0 to n-1 (inclusive). This represents the number of passes through the list.

3.) Track Swaps:

(·) Initialize a boolean variable swapped to False. This variable will track if any swaps are made in the current pass.

4.) Inner Loop:

(·) Iterate from j=0 to n-i-2 (inclusive). This loop compares adjacent elements in the list and performs swaps if necessary.

5.) Compare and Swap:

(·) For each pair of adjacent elements (i.e., students[j] and students[j+1]):

(·) Compare their score values:

(·) If students[j]['score'] > students[j+1]['score'], swap the two elements.

(·) Set swapped to True to indicate that a swap was made.

6.) Early Termination:

(·) After each pass of the inner loop, check if swapped is False. If no swaps were made during the pass, the list is already sorted, and you can break out of the outer loop early.

7.) Completion:

(·) The function modifies the students list in place, sorting it by score.

program 5.2

```
def bubble_sort_scores(students):
    n = len(students)
    for i in range(n):
        # Track if any swap is made in the pass
        swapped = False
        for j in range(0, n-i-1):
            if students[j]['score'] > students[j+1]['score']:
                # Swap if the score of the current student
                is greater than the next students[j], student[j+1]
                = students[j+1], students[j]
                swapped = True
        # If no two elements were swapped, the list is
        already sorted
        if not swapped:
            break
# Example usage
students = [
    {'name': 'Alice', 'score': 88},
    {'name': 'Bob', 'score': 95},
    {'name': 'Charlis', 'score': 75},
    {'name': 'Diana', 'score': 85}
]
print("Before Sorting:")
for student in students:
    print(student)
bubble_sort_scores(students)
print("\n After Sorting:")
for student in students:
    print(student)
```

EX No.	
PERFORMANCE (5)	
RESULT AND ANALYSIS (3)	
VIVA VOCE (3)	
RECORD (4)	
TOTAL (15)	
SIGN WITH DATE	

Result:

Thus, the program for various searching and sorting operations is executed and verified successfully.

Task : 6

Implement Various text file operation

Aim:

To Write a python program Implement Various text file operations

Problem 6.1:

You need to Write the sentence "Error objects are thrown when runtime errors occur. The Error object Can also be used as a base object for user-defined exceptions" into a text file named log.txt. Implement a function that performs this task.

Algorithm:

1.) Write to a File:

(o) Define Writefile (filename) function:

(·) open a file named "log.txt" in Write mode

(·) Write the following text to the file.

(·) "Error objects are thrown when runtime errors occur. The Error objects Can also be used as a base object for user-defined exceptions"

(·) Close the file.

2.) Read from a file:

o Define readfile (filename) function:

(·) Open the file Specified by filename in read mode using a With Statement.

(·) Read the entire Content of the file.

(·) Print the Content.

3.) Execute the program:

(·) Call Write file ("Write") to Write the predefined text to "log.txt".

(·) Call readfile ("text") to attempt to read from a file

Program 6.1

```
def Write file (filename):
    f = open("log.txt", "w")
    f.Write ("Errors objects are thrown when runtime errors occurs. The Error objects Can also be used as a base object for user-defined exceptions")
    f.close ()
def readfile (filename):
    With open(filename, "r") as file :
        Content = file.read ()
        print (Content)

Writefile ("Write")
readfile ("text")
```

OUTPUT :-

ERROR objects are thrown when runtime errors occurs. the error objects Can also be used as a base object for user-defined exceptions.

Problem 6.2

You have a text file txt containing logs of a system. Write a function that counts the number of lines containing the word "ERROR".

Algorithm:

1.) Initialize Error Counter:
   (o) Define the function count_error_lines (filename):
   (·) Initialize error_count to 0.

2.) Open and Read File:
   (o) Open the file specified by filename in read mode using a with statement

3.) Check Each Line for "ERROR":
   (o) Loop through each line in the file:
   (·) If the line contains the word "ERROR", increment error_count by 1.

4.) Return Error Count:
   (·) After reading all the lines, return the value of error_count.

5.) Execute the Program:
   (o) Call count_error_lines("log.txt") to count the number of lines with the word "ERROR" in the file "log.txt".
   (o) Print the result with the message: "Number of lines with 'ERROR': {error_lines}".

Program 6.2:

```
def count_error_lines (filename):
    error_count = 0
    with open (filename, "r") as file:
        for line in file:
            if "ERROR" in line:
                error_count += 1
    return error_count
error_lines = count_error_lines("log.txt")
print (f"Number of lines with 'ERROR':
    {error_lines}")
```

log.txt

"Error objects are thrown when runtime Error occur.
The Error object can also be used as a base object for user-defined exceptions"

Output:- 6.2

Number of lines with error : 2

Problem 6.3:

You need to write a report containing the details (Name, departments) of the employee in list. Write a Python function that writes this report to a file named: employee_report.txt.

Algorithm:

1. Create Employee Data:
   - Define the function Write_employee_report (filename):
   - Create a list employees containing dictionaries, each with "name" and "department" keys for individual employees.

2. open file for writing:
   - Open the file specified by filename in write mode using a with statement.

3. Write Employee Data to File:
   - Loop through each employee in the employees list:
   - For each employee, format a string as "Name: {employee['name']}, Department: {employee['department']}"
   - Write the formatted string to the file followed by a newline character (\n).

4. Execute the program:
   - Call Write_employee_report ("employee_report.txt") to write the employee data to the file "employee_report.txt".

Program 6.3:

```
def Write_employee_report (filename):
    employees = [
        {"name": Alice", "department": "HR"},
        {"name": "Bob", "department": "Engineering"},
        {"name": "Charlie", "department": "Finance"}
    ]
    with open (filename, "w") as file:
        for employees in employees:
            line = f"Name : {employee['name']}, Department : {employee['department']}\n"
            file.write(line)

# Example usage:
Write_employee_report ("employee_report.txt")
```

OUTPUT: 6.3

employees.txt

Name: Alice Department: HR

Name: Bob, Department: Engineering

Name: Charlie, Department: Finance

VELTECH	
EX No.	6
PERFORMANCE (5)	5
RESULT AND ANALYSIS (3)	3
VIVA VOCE (3)	3
RECORD (4)	4
TOTAL (15)	15
SIGN WITH DATE	

Result:

Thus, the python program implement various text file operations was successfully executed and the OUTPUT was verified.

Task No : 7.1

Utilizing 'Function' Concepts in python Programming

Aim: To write the python program using 'function' concepts in python programming student grades for a class projects. Write a python program that satisfies the above sorted [3, reversed(), and range()]

Algorithm:

1.) Start the program.
2.) Print a welcome message : OUTPUT a simple greeting.
3.) Determine and print the number of students uses len() to find the number of element in the student name list.
4.) Print the type of list : Uses type() to show the type of the student.
5.) Find and print highest and lowest grade: uses max() and min() to determine.
6.) Print sorted list of grades, Uses sorted() to sort the grades.
7.) Print reversed list of grades. Uses reverse [] to reverse the sorted list and convert.
8.) Generate and print a range of grade indices: Uses range [] to create a list of indicates
9.) Stop.

Program:

```
def analyze_student_grades():
    # Sample data
    students_names = ["Alice", "Bobo", "Charlie",
                      "Diana"]
    students_grades = [85, 92, 78, 90]
    # 1. print a welcome message
    print("Welcome to the Student Grades
          Analyzer!\n")
    # 2. Determine and print the number of student
    num_students = len(student_name)
    print("Number of students", num_students)
    # 3. Print the type of the student names
    list and the grade list
    print("\n Type of student_name list", type
          (student_names))
    print("type of student_grades list:", type
          (student_grades))
    # 4. Find and Print and the highest and lowest
    grade
    highest_grade = max(student_grades)
    lowest_grade = min(student_grades)
    print("\n Highest -:", highest_grade)
    print("Lowest grade", lowest_grade)
    # 5. print the list of grades sorted in
    ascending order sorted_grades = sorted
    (student_grades) print("\nSorted grads:", sorted_
    grades)
    # 6. Print the list of grades in reverse order
    reversed_grades = list(reversed(sorted_
    grades)) print("Reversed grades:", reversed_
    grades)
    # 7. Generate and print a range of grade indices
    from 1 to the number of students
    grad_indices = list(range(1, num_student + 1))
    print("\n Grade indices from 1 to number
          of students:", grade_indices)
```

OUTPUT :- 7.1

Welcome of the Student Grades Analyzer!

Numbes of Student : 4

Types of student list : < class 'list' >

Types of student_grades list : < class 'list' >

Highest grade : 92

Lowest grade : 78

Sorted grades : [78, 85, 90, 92]

Reversed grades : [92, 90, 85, 78]

Grade indices from 1 to number of students:

[1, 2, 3, 4]

O/P ✓

Run the analysis

analyze_student_grades( )

Task: 7.2 You are tasked with Creating a small calculator application to help users perform basic arithmetic operations ---- subtraction, multiplication, division.

Algorithm:

1.) Start the program

2.) User Input for Numbers : The program prompts the User to enter two numbers.

3.) User Input for Operation: The program prompts the user to choose an arithmetic operation (addition, subtraction, multiplication, division).

4.) Perform Operation: Based on the User's choice, the program performs the chosen arithmetic operation using the defined function.

5.) Display Result: The program display the results of the operation

6.) STOP

Program:

```
def add (a,b):
    """ Return the Sum of two numbers."""
    return a+b
def subtract (a,b):
    """ Return the difference between two numbers """
    return a - b
def multiply (a,b):
    """ Return the product of two numbers"""
    retun a x b
def divide (a,b):
    """ Return the quotient of two numbers. Handles
    division by zero."""
    if b != 0
```

OUTPUT : 7.2

Arithmetic operations:

Sum of 10 & 5 : 15

Difference between 10 and 5 : 5

product of 10 and 5 : 50

Quotient of 10 and 5 : 2.0

Greeting

Hello, Alice! Welcome to the program.

```
    return a/b
    else :
        return "Error : Division by zero"
    def greet (name):
"Return a greeting message for the user."""
    return f"Hello, {name}! Welcome to the program."
    def main ():
# Demonstrating the Use of User - defined functions
# Arithmetic operations
    num1 = 10
    num2 = 5
    print ("Arithmetic Operations:")
    print (f"Sum of {num1} and {num2}: ", add (num1
        , num2))
    print (f"Difference between {num1} and {num2}:",
        Subtract (num1, num2))
    print (f"Product of {num1} and {num2}:",
        multiply (num1, num2))
    print (f"Quotient of {num1} and {num2}:",
        divide (num1, num2)
// Greeting the user
    user_name = "Alice"
    print ("\n Greeting:")
    print (greet (user_name))
# Run the main function
    if __name__ == "__main__":
        main ()
```

VEL TECH-CSE	
EX NO.	7
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	15
SIGN WITH DATE	

Result :- Thus the python program using 'Functions' Concepts was Successfully executed and the output

OUTPUT : 8.1 (a)

Enter the starting number :2
Enter ending number :5
Enter step value : 1

2
3
4
5

Task-8 Implement python generator and decorators

Date: 26/9/25

Aim: Write a python program to Implement python generators and decorators.

Algorithm:

1. Define Generator Function:
   - Define the function number-sequence (start, end, step=1).
2. Initialize Current Value:
   - Set current to the value of start.
3. Generate Sequence:
   - While current is less than or equal to end:
     - Yield the current value of current
     - Increment current by step.
4. Get User Input:
   - Read the starting number (start) from user input.
   - Read the ending number (end) from user input.
   - Read the step value (step) from user input.
5. Create Generator object:
   - Create a generator object by calling number-sequence (start, end, step) with user-provided values
6. Print Generated Sequence:
   - Iterate over the values produced by the generator object.
   - Print each value.

8.1 program:

```
def number_sequence(start, end, step=1):
    current = start
    while current <= end:
        yield current
        current += step
start = int(input("Enter the starting number:"))
end = int(input("Enter the ending number:"))
step = int(input("Enter the step value:"))
# Create the generator
sequence_generator = number_sequence(start,
    end, step)
# print the generated sequence of numbers
for number in sequence_generator
    print(number)
```

Produce a default sequence of number starting from 0, ending at 10, and with a step of 1 if no values are provided

Algorithm:-

1. Start Function:
   - Define the function my_generator(n) that takes a parameter n.
2. Initialize Counter:
   - Set value to 0.
3. Generate values:
   - While value is less than n:
     - yield the current value.
     - Increment value by 1.

OUTPUT - 8.1(b)

output : 0
1
2

4. Create Generator objects:

- Call my_generator (11) to Create a generator object.

5. Iterate and print values;

- For each value produced by the generator object:
- Print value.

8.1.(b) program:

```
def my_generator(n):
    # initialize Counter
        value = 0
    # loop until Counter is less than n
        while value < n:
    # produce the Current value of the
        Counter yield value
    # increment the Counter
        value += 1
# iterate over the generator object produced
  by my_generator for value in my_
  generator (3):
# print each value produced by generator
  print (value)
```

OUTPUT : 8.2

HT, I AM Created by a function passed as an argument

HT, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT

Task 8.2 . Imagine you are working on a messaging application that needs to format message . . . . . program to implement it.

Algorithm :

1. Create Decorators :

- Define uppercase_decorator to Convert the result of a function to uppercase
- Define lowercase_decorator to Convert the result of a function to lowercase.

2. Define functions :

- Define shout function to return the input text. Apply @ uppercase_decorator to this function.
- Define whisper function to return the input text.
  Apply @ lowercase_decorator to this function.

3. Define Greet function :

- Define greet function that :
  - Accepts a function (func) as input.
  - Calls this function with the text "Hi, I am Created by a function passed as an argument."
  - Prints the results.

4. Execute the program :

- Call greet (Shout) to print the greeting in uppercase
- Call greet (whisper) to print the greeting in lowercase.

Program:

```
def uppercase_decorator (func):
    def wrapper (text):
        return func (text).upper()
    return wrapper

def lowercase_decorator (func):
    def wrapper (text):
        return func (text).lower()
    return wrapper

@uppercase_decorator
def shout (text):

@lowercase_decorator
def whisper (text):
    return text

def greet (func):
    greeting = func ("Hi, I am created by
    a function passed as an argument.
    print (greeting)
greet (shout)
greet (whisper)
```

VEL TECH - CSE	
EX NO.	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	—
TOTAL (20)	14
SIGN WITH DATE	

Result:- Thus the python program to Implement python generator and decorators was Successfully executed and the Output was verified.

OUTPUT : 9.1

Grades List : [85, 90, 78, 92, 88]

Enter the index of the grade you want to view: 3

The grade at index 3 is : 92

Task - 9.1

Date: 10/10/25

Aim: To implement Exceptions and Exceptional handling in python.

Algorithm:

1. Start the program
2. Initialize a list of grades (e.g. [85, 90, 78, 92, 88]).
3. Prompts the user to enter the index of the grade they wish to view.
4. Attempts to display the grade at the Specified index.
5. If the index is out of range, catches the Index Error and prints an error message, "Invalid index. Please enter a valid index.".

program:

```
# Initialize the list of grades
grades = [85, 90, 78, 92, 88]

# Display the grades list
print ("Grade's List:", grades)

# Prompt the user to enter the index
of the grade they want to view.
try:
    index = int (input ("Enter the index
                of the grade you want to
                view: "))
# Attempt to display the grade at the
Specified index
    print (f"The grade at index
            {index} is :
{grades [index]} ")
except Index Error:
# Handle the Case where the index is out
of range
print ("Invalid index. Please enter a valid
        index.")
except value Error:
# Handle the Case where the input is
not an integer
print ("Invalid input. Please enter a
        numerical index.")
```

OUTPUT : 9.2

Enter the ~~number~~ numerator : 12

Enter the denominator : 13

Result : 0. 9230769....

Task 9.2

You are developing a python Calculator program that performs basic .... handled properly.

Algorithm :

1. Start the program.
2. Prompts the user to enter two numbers : a numerator and a denominator.
3. Attempts to divide the numerator by the denominator.
4. If the denominator is zero, catches the zero Devision Error and displays and error message : " Error : Division by zero is not allowed."

Program :

```
# Function to perform division
def divide_numbers():
    try:
        # Prompt the user to enter the numerator
        numerator = float(input("Enter the
            numberator: "))
        # Prompt the user to enter the denominator
        denominator = float(input("Enter the
            denominator: "))
        # Attempt to perform division
        result = numerator / denominator
        print(f"Result : {result}")
    except zero Division Error:
        # Handle division by zero error
        print("Error : Division by zero is not
            allowed.")
    except Value Error:
        # Handle invalid input that is not a number
        print("Error : Please enter valid numbers.")

# Call the function to execute the division
operation divide_numbers().
```

OUTPUT : 9.3

(1) Enter a number : 32
Eligible to vote

(2) Enter a number : 3
exception occurred : invalid age.

6/pr 8

Task : You are building a Python application to determine if a person is eligible ------ 18 is entered

Algorithm :

1. Define the Custom exception.
2. Prompt the User for input.
3. Check if the age is below 18.
4. Raise an exception if the condition is met.
5. Handle the exception with a Custom error message.

Program :

```
#define Python user-defined exceptions
class InvalidAgeException (Exception):
    "Raised when the input value is less than 18"
    pass
# ~~Rai~~ you need to guess this number
number = 18

try:
    input_num = int (input ("Enter a number:"))
    if input_num < number:
        raise InvalidAgeException
    else:
        print ("Eligible to vote")
except InvalidAgeException:
    print ("Exception o ccurred : Invalid Age")
```

VEL TECH-CSE	
EX NO.	9
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	—
TOTAL (20)	15
SIGN WITH DATE	

Result : Thus the program Implement Exceptions and Exceptional handling is executed and verified Successfully.

```
        fruit_spawn = False
    else:
        snake_body.pop()

    if not fruit_spawn:
        fruit_position = [random.randrange(1, (window_x//10)) * 10,
                          random.randrange(1, (window_y//10)) * 10]
        fruit_spawn = True
    game_window.fill(black)

    for pos in snake_body:
        pygame.draw.rect(game_window, green, pygame.Rect(pos[0], pos[1], 10, 10))

# Frame Per Second / Refresh Rate
    fps.tick(snake_speed)
```

OUTPUT : 12.2

R	N	B	Q	K	B	N	R
P	P	P	P	P	P	P	P
P	P	P	P	P	P	P	P
R	N	B	Q	K	B	N	R

Date: 31/10/25

Write a python program to Develop a class board using pygame.

Algorithm:

1. Import pygame and initialize it.
2. Set Screen size and title.
3. Define Colors for the board and pieces. Define a function to draw the board by looping over rows and columns and drawing squares of different colors.
4. Define a function to draw the pieces on the board by loading images for each piece and placing them on the corresponding square.
5. Define the initial state of the board as a list of lists containing the pieces.
6. Draw the board and pieces on the Screen.
7. Start the game loop.

Program:

```
import pygame
import sys

#Initialize pygame
pygame.init()

# Screen setup
Screen_size = (640, 640)
Screen = pygame.display.set_mode
        (Screen_Size)
pygame.display.set_caption('Chess Board')
```

```
# Define colors
black = (0,0,0)
white = (255, 255, 255)

# Font for drawing piece letters
font = pygame.font.Sysfont ('arial', 50,
        bold = True)

# Draw the chessboard
def draw_(board ):
    Square_Size = 80
    for row in range (8):
        piece = board [row] [col]
        if piece != ' ':
# White pieces = uppercase (draw in white)
# black pieces = lowercase (draw in grey)
    if piece.isupper ():
        text_color = (240, 240, 240)
# white-ish
    else :
        text_color = (100, 100, 100) # grey
    for black pieces

        text_Surfaces = font.render (piece.
                        upper (), True, text_
                        Colour)

        text_rect = text_surface.get_rect
                    (Center = (col * Square_
                    Size + 40 *
                    Square_
    Screen.bilt (text_Surface, text_rect) Size + 40)

# Initial board setup
board = [
    ['r','n','b','q','k','b','n','r'],
    ['p','p','p','p','p','p','p','p'],
```

```
[ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' ],
[ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' ],
[ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' ],
[ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' ],
[ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' ],
[ 'P', 'P', 'P', 'P', 'P', 'P', 'P', 'P' ],
[ 'R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R' ]
]

# Main loop
running = True
while running:
    Screen.fill(white)
    draw_board()
    draw_pieces(board)
    pygame.display.update()

    for event in pygame.event.get()
        if event.type == pygame.QUIT:
            running = False

    pygame.quit()
    Sys.exit()
```

VEL TECH-CSE	
EX NO.	12
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	—
TOTAL (20)	15
SIGN WITH DATE	

EX NO	
PERFORMANCE (5)	
RESULT AND ANALYSIS (5)	
VIVA VOCE (5)	
RECORD (5)	
TOTAL (20)	
SIGN WITH DATE	

Result: Thus the program for pygame is executed and verified successfully.

Use Case : 1

Date: 1/11/25

Use Case - Finding the winning strategy in a card game in python

Problem Description:

Imagine a card game where each player receives a hand of cards with values. The objective is of find the best way to maximize the score for a player, assuming the players take turns drawing cards.

Assumptions:

- Each player tries to maximize their score
- Cards are represented by integers which indicate their values.

you need to design an algorithm that helps a player find the optimal strategy to guarantee the highest possible score given that the opponent is also playing optimally.

Plan:

We can solve this problem using Dynamic Programming by calculating the optimal score for every possible scenario, taking into account the best choices for both players.

Steps:

1. Define the Game: Represent the pile of cards as a list of integers.
2. Recursive Strategy: A function will recursively determine the best score player can achieve.

Use Case : 1

Date: 1/11/25

Use Case - Finding the Winning strategy in a Card game in python

Problem Description:

Imagine a Card game where each player receives a hand of Cards with values. The objective is of find the best way to maximize the Score for a player, assuming the players take turns drawing Cards.

Assumptions:

- Each player tries to maximize their Score
- Cards are represented by integers which indicate their values.

you need to design an algorithm that helps a player find the optimal strategy to guarantee the highest possible Score given that the opponent is also playing optimally.

Plan:

We Can solve this problem using Dynamic Programming by calculating the optimal Score for every possible scenario, taking into account the best choices for both players.

Steps:

1. Define the Game: Represent the pile of Cards as a list of integers.
2. Recursive Strategy: A function will recursively determine the best Score a player Can achieve.

3. Dynamic Programming (Memoization): Store intermediate results to avoid recalculating them.

4. Base Cases: When only one card is left, the current player takes it.

Program:

```
def find_optimal_strategy(Cards):
    n = len(Cards)
    # Create a memoization table to store subproblem results
    dp = [[0] * n for _ in range(n)]

# fill the table for subproblems of increasing sizes
for length in range(1, n+1):
    for i in range(n - length + 1):
        j = i + length - 1
    # If only one card is left, the player takes it
        if i == j:
            dp[i][j] = Cards[i]
        else:
            # Choose the best of two choices:
# 1. Take the left card, and the opponent plays optimally on the remaining
(i+1, j)

# 2. Take the right card, and the opponent plays optimally on the remaining
(i, j-1)
            take_left = Cards[i] - dp[i+1][j]
            take_right = Cards[j] - dp[i][j-1]
            dp[i][j] = max(take_left, take_right)
```

```
return dp[0][n-1] + sum(cards)) // 2
```

```
# Example Case
cards = [3, 9, 1, 2]
print ("First player's optimal score:",)
find_optimal_strategy (cards))
```

Explanation:

(•) Dynamic Programming Table (dp):
Each cell dp[i][j] represents the difference in score between the first player and the opponent if the game to index i to index j.

(•) Two Choices: For each move, the player can either:

1.) Pick the leftmost card cards[i],

2.) Pick the rightmost card cards[j].

(•) Recursive Relation:

The value of each subproblem is determined by maximizing the score difference between the current player and the opponent.

Example Walkthrough:

Consider the array of cards: [3, 9, 1, 2]

1.) First player (you) can choose between:

(•) Taking the leftmost card (3), leaving the card [9, 1, 2]

(•) Taking the rightmost card (2), leaving the cards [3, 9, 1]

First player's optimal Score : 5

First player's, if playing optimally, can guarantee a Score of 5 regardless of how the opponent plays.

Optimizing Strategy:

By using Dynamic Programming we ensure that the Solution is Computed efficiently, avoiding redundant calculation. The approach ensures both players play optimally, and the first player gets the highest Score possible given the opponent's best move.

VEL TECH-CSE	
EX NO.	Use Case
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	—
TOTAL (20)	15
SIGNATURE WITH DATE	