Lab 8 Lab Session - Functional Testing (Black-Box)

Name :- Jemini chaudhari
Student ID:- 202201521

Q.1.

Equivalence partitioning:-

input	outcome	Reason
(15,8,2010)	(14,8,2010)	N/A
(1,3,2015)	(28,2,2015)	N/A
(1,3,2000)	(29,2,2000)	N/A
(1,1,2000)	(31,12,1999)	N/A
(1,0,2000)	Error message	Invalid month
(1,13,2000)	Error message	Invalid month
(32,1,2000)	Error message	Invalid day
(0,1,2000)	Error message	Invalid day
(1,1,1899)	Error message	Year out of range
(1,1,2016)	Error message	Year out of range

Boundary value analysis :-

Month boundary cases

Test case	input	Expected outcome	remarks
Lower boundary	(1,1,2000)	Previous date =(31,12,1999)	valid
Upper boundary	(1,12,2000)	Previous date =(30,11,2000)	valid

Just below lower	(1,0,2000)	Error message	invalid
Just above upper	(1,13,2000)	Error message	invalid

Day boundary cases(Month with 31 days)

Test case	input	Expected outcome	remarks
Lower boundary	(1,1,2000)	Previous date =(31,12,1999)	valid
Upper boundary	(31,1,2000)	Previous date =(30,01,1999)	valid
Just below lower	(0,1,2000)	Error message	invalid
Just above upper	(32,1,2000)	Error message	invalid

Day Boundary Cases (February in Leap Year)

Test case	input	Expected outcome	remarks
Lower boundary	(1,2,2000)	Previous date =(31,01,2000)	valid
Upper boundary	(29,2,2000)	Previous date =(28,02,2000)	valid
Just below lower	(0,2,2000)	Error message	invalid
Just above upper	(30,2,2000)	Error message	invalid

Day Boundary Cases (February in Non-Leap Year)

Test case	input	Expected outcome	remarks
Lower boundary	(1,2,2001)	Previous date =(31,01,2001)	valid

Just below lower	(28,2,2001)	Previous date =(27,02,2001)	invalid
Just above upper	(29,2,2001)	Error message	invalid

Year Boundary Cases

Test case	input	Expected outcome	remarks
Lower boundary	(1,1,1900)	Previous date =(31,12,1899)	valid
Upper boundary	(31,12,2015)	Previous date =(30,12,2015)	valid
Just below lower	(1,1,1899)	Error message	invalid
Just above upper	(1,1,2016)	Error message	invalid

Program execution:-

```
def is_leap_year(year):
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

def max_day_in_month(month, year):
    if month in [4, 6, 9, 11]:
        return 30
    elif month == 2:
        return 29 if is_leap_year(year) else 28
    else:
        return 31

def previous_date(day, month, year):
    if not (1 <= month <= 12):
        return "Error: Invalid month"
    if not (1900 <= year <= 2015):
        return "Error: Invalid year"

    max_day = max_day_in_month(month, year)
    if not (1 <= day <= max_day):
```

```
        return "Error: Invalid day"

    if day > 1:
        return f"{day-1}/{month}/{year}"
    else:
        if month == 1:
            return f"31/12/{year-1}" if year > 1900 else "Error: Invalid year"
        else:
            previous_month = month - 1
            previous_day = max_day_in_month(previous_month, year)
            return f"{previous_day}/{previous_month}/{year}"

# Test suite execution
test_cases = [
    (15, 7, 2005),  # Valid
    (32, 1, 2010),  # Invalid day
    (31, 2, 2012),  # Invalid day in February
    (10, 13, 2011), # Invalid month
    (25, 5, 1800),  # Invalid year
    (29, 2, 2012),  # Valid leap year
    (1, 3, 2012),   # Valid leap year boundary
]

for day, month, year in test_cases:
    result = previous_date(day, month, year)
    print(f"Input: {day}/{month}/{year} -> Output: {result}")
```

Q.2. Programs:

P1.
Equivalence partioning

Input (v, a)	Expected Output	Equivalence Class
5, [1, 2, 3, 4, 5]	4	Class 1: Value exists

10, [1, 2, 3, 4, 5]	-1	Class 2: Value does not exist
5, []	-1	Class 3: Array is empty
-1, [1, 2, 3, 4, 5]	-1	Class 4: Invalid value

Boundary value analysis

Input (v, a)	Expected Output	Boundary Condition
1, [1]	0	Value at first index (array length 1)
5, [5]	0	Value at first index (array length 1)
1, [1, 2, 3, 4, 5]	0	Value at the first index
5, [1, 2, 3, 4, 5]	4	Value at the last index
0, [1, 2, 3, 4, 5]	-1	Value does not exist (outside bounds)
1, []	-1	Boundary condition (empty array)
6, [1, 2, 3, 4, 5]	-1	Value does not exist (edge case)

```
def linear_search(v, a):
    i = 0
    while i < len(a):
        if a[i] == v:
            return i
        i += 1
    return -1
# Example array
arr = [1, 2, 3, 4, 5]
value_to_search = 3

# Searching for the value
```

```
index = linear_search(value_to_search, arr)

if index != -1:
    print(f"Value {value_to_search} found at index {index}.")
else:
    print(f"Value {value_to_search} not found in the array.")
```

P2.

Equivalence partitioning

Test Case	Input	Expected Output	Remarks	Valid/Invalid
1	v = 5, a = [1, 2, 5, 5, 3]	2	v appears twice in the array.	Valid
2	v = 2, a = [1, 2, 3, 4, 5]	1	v appears once in the array.	Valid
3	v = 6, a = [1, 2, 3, 4, 5]	0	v does not exist in the array.	Valid
4	v = 3, a = [3]	1	Single-element array, v exists.	Valid
5	v = 3, a = [1]	0	Single-element array, v does not exist.	Valid
6	v = 5, a = []	0	Empty array, return 0.	Invalid

Boundary value analysis

Test Case	Input (v, array)	Expected Output	Remarks	Valid/Invalid
1 (Lower)	v = 1, a = [1]	1	Single element, v exists in the array.	Valid
2 (Upper)	v = 10, a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	1	v exists at the last index.	Valid
3 (Lower)	v = 1, a = [1, 2, 3, 4, 5]	1	v exists at the first index.	Valid

4 (Middle)	v = 3, a = [1, 2, 3, 4, 5]	1	v exists in the middle of the array.	Valid
5 (Multiple)	v = 5, a = [5, 5, 5, 5, 5]	5	v appears multiple times in the array.	Valid
6 (Not Found)	v = 6, a = [1, 2, 3, 4, 5]	0	v does not exist in the array.	Valid
7 (Empty)	v = 1, a = []	0	Empty array, return 0.	Invalid

```
def count_item(v, a):
    count = 0
    for i in range(len(a)):
        if a[i] == v:
            count += 1
    return count

# Test cases
# Test Case 1: Value appears multiple times
arr1 = [1, 2, 3, 1, 4, 1]
value_to_count1 = 1
# Expected Output: 3 (1 appears 3 times)

# Test Case 2: Value appears once
arr2 = [5, 5, 5, 6, 7]
value_to_count2 = 6
# Expected Output: 1 (6 appears 1 time)

# Test Case 3: Value does not appear
arr3 = [2, 3, 4, 5]
value_to_count3 = 1
# Expected Output: 0 (1 does not appear)

# Test Case 4: Empty array
arr4 = []
value_to_count4 = 2
# Expected Output: 0 (array is empty)

# Test Case 5: All elements are the same as the searched value
arr5 = [7, 7, 7, 7]
value_to_count5 = 7
# Expected Output: 4 (7 appears 4 times)
```

```
# Test Case 6: Negative numbers in the array
arr6 = [-1, -2, -1, -3, -1]
value_to_count6 = -1
# Expected Output: 3 (-1 appears 3 times)
```

P3.
Equivalence partitioning

Test Case	Input	Expected Output	Remarks	Valid/Invalid
1 (Valid)	v = 5, a = [1, 2, 5, 5, 6]	2	v appears twice in the array (first index).	Valid
2 (Valid)	v = 3, a = [1, 2, 3, 4, 5]	2	v appears once in the array.	Valid
3 (Valid)	v = 6, a = [1, 2, 3, 4, 5]	-1	v does not exist in the array.	Valid
4 (Valid)	v = 1, a = [1]	0	Single-element array, v exists.	Valid
5 (Valid)	v = 2, a = [1]	-1	Single-element array, v does not exist.	Valid
6 (Invalid)	v = 5, a = []	-1	Empty array, return -1.	Invalid
7 (Invalid)	v = 5, a = [3, 1, 2]	-1	Array not sorted, invalid search.	Invalid

Boundary Value Analysis:-

Test Case	Input (v, array)	Expected Output	Remarks	Valid/Invalid
1 (Lower)	v = 1, a = [1]	0	Single element, v exists.	Valid
2 (Upper)	v = 5, a = [1, 2, 3, 4, 5]	4	v exists at the last index.	Valid
3 (Lower)	v = 1, a = [1, 2, 3, 4, 5]	0	v exists at the first index.	Valid

4 (Middle)	v = 3, a = [1, 2, 3, 4, 5]	2	v exists in the middle of the array.	Valid
5 (Not Found)	v = 6, a = [1, 2, 3, 4, 5]	-1	v does not exist in the array.	Valid
6 (Empty)	v = 1, a = []	-1	Empty array, return -1.	Invalid

```
def binarySearch(v, a):
    lo = 0
    hi = len(a) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if v == a[mid]:
            return mid
        elif v < a[mid]:
            hi = mid - 1
        else:
            lo = mid + 1
    return -1

# Test Cases
# Test case 1: Value is present at the beginning of the array
result1 = binarySearch(3, [3, 5, 7, 9, 11])  # Expected outcome: 0 (index of 3)

# Test case 2: Value is present at the end of the array
result2 = binarySearch(11, [3, 5, 7, 9, 11])  # Expected outcome: 4 (index of 11)

# Test case 3: Value is present in the middle of the array
result3 = binarySearch(7, [3, 5, 7, 9, 11])  # Expected outcome: 2 (index of 7)

# Test case 4: Value is not present in the array (less than minimum)
result4 = binarySearch(1, [3, 5, 7, 9, 11])  # Expected outcome: -1 (1 is not found)

# Test case 5: Value is not present in the array (greater than maximum)
result5 = binarySearch(12, [3, 5, 7, 9, 11])  # Expected outcome: -1 (12 is not
found)

# Test case 6: Value is not present in the array (between two values)
result6 = binarySearch(6, [3, 5, 7, 9, 11])  # Expected outcome: -1 (6 is not found)
```

```
# Test case 7: Empty array
result7 = binarySearch(5, [])  # Expected outcome: -1 (no elements to search)

# Test case 8: Array with one element (value present)
result8 = binarySearch(5, [5])  # Expected outcome: 0 (index of 5)

# Test case 9: Array with one element (value not present)
result9 = binarySearch(3, [5])  # Expected outcome: -1 (3 is not found)

# You can print the results if needed, but it's omitted as per your request.
```

P4.
Equivalence partitioning

Test Case	Input	Expected Output	Remarks	Valid/Invalid
1 (Valid)	3, 3, 2003	0	All sides equal (equilateral).	Valid
2 (Valid)	5, 5, 2003	1	Two sides equal (isosceles).	Valid
3 (Valid)	3, 4, 2005	2	All sides different (scalene).	Valid
4 (Invalid)	1, 2, 2003	3	One side equal to the sum of the other two (invalid).	Invalid
5 (Invalid)	2, 1, 2003	3	One side equal to the sum of the other two (invalid).	Invalid
6 (Invalid)	-1, 2, 2	3	Negative side length (invalid).	Invalid
7 (Invalid)	2000, 2, 2	3	Zero side length (invalid).	Invalid
8 (Invalid)	5, 5, 10	3	One side greater than sum of other two (invalid).	Invalid

Boundary Value Analysis:-

Test Case	Input (a, b, c)	Expected Output	Remarks	Valid/Invalid
1 (Valid)	1, 1, 2001	0	Minimum positive integers (equilateral).	Valid
2 (Valid)	1, 2, 2002	1	Minimum values for isosceles.	Valid
3 (Valid)	2, 3, 2004	2	Simple scalene triangle.	Valid
4 (Invalid)	1, 1, 2002	3	One side equal to the sum of the others (invalid).	Invalid
5 (Invalid)	1, 2, 2003	3	One side equal to the sum of the others (invalid).	Invalid
6 (Invalid)	-1, 2, 2	3	Negative value (invalid).	Invalid
7 (Invalid)	0, 0, 0	3	All sides zero (invalid).	Invalid
8 (Invalid)	10^6, 10^6, 10^6	0	Large values, equilateral.	Valid
9 (Invalid)	10^6, 10^6, 2*10^6	3	One side greater than the sum of others (invalid).	Invalid

```
def triangle(a, b, c):
    EQUILATERAL = 0
    ISOSCELES = 1
    SCALENE = 2
```

```
    INVALID = 3

    if a >= b + c or b >= a + c or c >= a + b:
        return INVALID
    if a == b and b == c:
        return EQUILATERAL
    if a == b or a == c or b == c:
        return ISOSCELES
    return SCALENE

# Test Cases
# Test case 1: Equilateral triangle
result1 = triangle(3, 3, 3)  # Expected outcome: 0 (EQUILATERAL)

# Test case 2: Isosceles triangle
result2 = triangle(4, 4, 2)  # Expected outcome: 1 (ISOSCELES)

# Test case 3: Scalene triangle
result3 = triangle(5, 6, 7)  # Expected outcome: 2 (SCALENE)

# Test case 4: Invalid triangle (sum of two sides is not greater than the third)
result4 = triangle(1, 2, 3)  # Expected outcome: 3 (INVALID)

# Test case 5: Invalid triangle (one side is greater than the sum of the other two)
result5 = triangle(1, 5, 3)  # Expected outcome: 3 (INVALID)

# Test case 6: Invalid triangle (negative sides)
result6 = triangle(-1, 2, 2)  # Expected outcome: 3 (INVALID)

# Test case 7: Invalid triangle (zero length)
result7 = triangle(0, 2, 2)  # Expected outcome: 3 (INVALID)

# Test case 8: Another valid scalene triangle
result8 = triangle(8, 5, 6)  # Expected outcome: 2 (SCALENE)

# You can print the results if needed, but it's omitted as per your request.
```

P5.
Equivalence partitioning

Test Case	Input (s1, s2)	Expected Output	Remarks	Valid/Invalid
1 (Valid)	"abc", "abc"	TRUE	Exact match (s1 is a prefix of s2).	Valid
2 (Valid)	"abc", "abcdef"	TRUE	Proper prefix (s1 is a prefix of s2).	Valid
3 (Invalid)	"abcdef", "abc"	FALSE	s1 is longer than s2 (invalid prefix).	Invalid
4 (Invalid)	"abc", "def"	FALSE	No matching characters (invalid prefix).	Invalid
5 (Invalid)	"abc", "ab"	FALSE	s1 is longer than s2 (invalid prefix).	Invalid
6 (Invalid)	"abc", ""	FALSE	s2 is empty (invalid prefix).	Invalid
7 (Valid)	"", "abc"	TRUE	Empty s1 is a prefix of any s2 (valid).	Valid

Boundary Value Analysis:-

Test Case	Input (s1, s2)	Expected Output	Remarks	Valid/Invalid
1 (Valid)	"", "abc"	TRUE	Empty s1 is a prefix of any non-empty s2.	Valid
2 (Invalid)	"", ""	TRUE	Both are empty strings (valid).	Valid
3 (Invalid)	"abc", ""	FALSE	s2 is empty (invalid prefix).	Invalid
4 (Valid)	"a", "abc"	TRUE	Single character prefix (valid).	Valid
5 (Valid)	"abc", "abc"	TRUE	Exact match (valid).	Valid
6 (Invalid)	"abc", "ab"	FALSE	s1 is longer than s2 (invalid prefix).	Invalid
7 (Invalid)	"xyz", "abc"	FALSE	No matching characters (invalid prefix).	Invalid

```
def prefix(s1, s2):
    if len(s1) > len(s2):
        return False
    for i in range(len(s1)):
        if s1[i] != s2[i]:
            return False
    return True

# Test Cases
# Test case 1: s1 is a prefix of s2
result1 = prefix("pre", "prefix")  # Expected outcome: True

# Test case 2: s1 is equal to s2
result2 = prefix("same", "same")   # Expected outcome: True

# Test case 3: s1 is not a prefix of s2
result3 = prefix("not", "prefix")  # Expected outcome: False

# Test case 4: s1 is longer than s2
```

```
result4 = prefix("longer", "short") # Expected outcome: False

# Test case 5: s1 is an empty string
result5 = prefix("", "anything")    # Expected outcome: True

# Test case 6: s2 is an empty string, and s1 is not
result6 = prefix("notempty", "")    # Expected outcome: False

# Test case 7: Both s1 and s2 are empty strings
result7 = prefix("", "")            # Expected outcome: True

# Test case 8: s1 is a single character and matches the start of s2
result8 = prefix("a", "apple")      # Expected outcome: True

# Test case 9: s1 is a single character and does not match the start of s2
result9 = prefix("b", "apple")      # Expected outcome: False

# You can print the results if needed, but it's omitted as per your request.
```

P6.

a) Equivalence Classes Identification

1. Valid Equivalence Classes:
    - Equilateral Triangle: All three sides are equal (A = B = C).
    - Isosceles Triangle: Exactly two sides are equal (A = B ≠ C, A = C ≠ B, B = C ≠ A).
    - Scalene Triangle: All sides are different (A ≠ B, A ≠ C, B ≠ C).
    - Right-Angled Triangle: Satisfies Pythagorean theorem ($A^2 + B^2 = C^2$ or permutations).
2. Invalid Equivalence Classes:
    - Non-Triangle: The sum of any two sides is less than or equal to the third side (A + B ≤ C, A + C ≤ B, B + C ≤ A).
    - Non-Positive Inputs: One or more sides are non-positive (A ≤ 0, B ≤ 0, C ≤ 0).

b)Equivalence Classes

Test Case ID	Input (A, B, C)	Expected Output	Equivalence Class
TC1	3.0, 3.0, 3.0	"Equilateral Triangle"	Equilateral Triangle
TC2	4.0, 4.0, 2.0	"Isosceles Triangle"	Isosceles Triangle
TC3	3.0, 4.0, 5.0	"Scalene Triangle"	Scalene Triangle
TC4	5.0, 12.0, 13.0	"Right-Angled Triangle"	Right-Angled Triangle
TC5	1.0, 2.0, 3.0	"Not a Triangle"	Non-Triangle
TC6	-1.0, 2.0, 2.0	"Invalid Input"	Non-Positive Inputs
TC7	3.0, 0.0, 5.0	"Invalid Input"	Non-Positive Inputs
TC8	3.0, 4.0, 7.0	"Not a Triangle"	Non-Triangle

c)Boundary Condition for Scalene Triangle (A + B > C)

Test Case ID	Input (A, B, C)	Expected Output	Boundary Condition
TC9	3.0, 4.0, 5.0	"Scalene Triangle"	A + B > C
TC10	2.0, 2.0, 3.9	"Scalene Triangle"	A + B = C but not valid for triangle
TC11	2.0, 3.0, 4.0	"Scalene Triangle"	Just valid (A + B > C)
TC12	3.0, 5.0, 8.0	"Not a Triangle"	A + B = C (invalid)

d)Boundary Condition for Isosceles Triangle (A = C)

Test Case ID	Input (A, B, C)	Expected Output	Boundary Condition
TC13	3.0, 3.0, 5.0	"Isosceles Triangle"	A = B (valid)
TC14	3.0, 3.0, 6.0	"Not a Triangle"	A + B = C (invalid)
TC15	5.0, 5.0, 5.0	"Equilateral Triangle"	A = B (valid, also equilateral)
TC16	4.0, 4.0, 7.0	"Isosceles Triangle"	A = B (valid)

e) Boundary Condition for Equilateral Triangle (A = B = C)

Test Case ID	Input (A, B, C)	Expected Output	Boundary Condition
TC17	3.0, 3.0, 3.0	"Equilateral Triangle"	A = B = C
TC18	0.0, 0.0, 0.0	"Invalid Input"	Non-positive (invalid)
TC19	1.0, 1.0, 1.0	"Equilateral Triangle"	Valid minimal triangle

f) Boundary Condition for Right-Angled Triangle ($A^2 + B^2 = C^2$)

Test Case ID	Input (A, B, C)	Expected Output	Boundary Condition
TC20	3.0, 4.0, 5.0	"Right-Angled Triangle"	$A^2 + B^2 = C^2$
TC21	5.0, 12.0, 13.0	"Right-Angled Triangle"	$A^2 + B^2 = C^2$
TC22	6.0, 8.0, 10.0	"Right-Angled Triangle"	$A^2 + B^2 = C^2$
TC23	1.0, 1.0, 1.5	"Not a Triangle"	$A^2 + B^2 < C^2$

g) Test Cases for Non-Triangle Condition

Test Case ID	Input (A, B, C)	Expected Output	Remarks
TC24	1.0, 1.0, 2.0	"Not a Triangle"	$A + B \leq C$ (non-triangle)
TC25	2.0, 3.0, 6.0	"Not a Triangle"	$A + B < C$ (non-triangle)
TC26	4.0, 5.0, 10.0	"Not a Triangle"	$A + B < C$ (non-triangle)
TC27	1.0, 2.0, 3.0	"Not a Triangle"	$A + B \leq C$ (non-triangle)

h)Test Cases for Non-Positive Input

Test Case ID	Input (A, B, C)	Expected Output	Remarks
TC28	0.0, 1.0, 1.0	"Invalid Input"	Non-positive input (A is non-positive)
TC29	-1.0, 2.0, 2.0	"Invalid Input"	Non-positive input (A is negative)
TC30	2.0, -2.0, 2.0	"Invalid Input"	Non-positive input (B is negative)
TC31	1.0, 1.0, -1.0	"Invalid Input"	Non-positive input (C is negative)
TC32	0.0, 0.0, 0.0	"Invalid Input"	All sides are non-positive