1)Design, develop, code and run the program in any suitable language to solve the commission problem. Analyse it from the perspective of boundary value testing, derive different test cases, execute these test cases and discuss the test results.

UNIT TESTING

```
def calculate_commission(fans, pumps, bodies):
    total_sales = (fans * 45) + (pumps * 30) + (bodies * 25)
    if total_sales <= 1000:
        commission = total_sales * 0.10
    elif total_sales <= 1800:
        commission = 1000 * 0.10 + (total_sales - 1000) * 0.15
    else:
        commission = 1000 * 0.10 + 800 * 0.15 + (total_sales - 1800) * 0.20
    return commission


# Test cases
test_cases = [
    {"fans": 1, "pumps": 1, "bodies": 1, "expected_commission": 10.0},  # minimum sales
    {"fans": 70, "pumps": 80, "bodies": 90, "expected_commission": 1260.0},  # maximum sales
    {"fans": 10, "pumps": 10, "bodies": 10, "expected_commission": 120.0},  # sales around $1000
    {"fans": 20, "pumps": 20, "bodies": 20, "expected_commission": 360.0},  # sales around $1800
    {"fans": 5, "pumps": 5, "bodies": 5, "expected_commission": 60.0},  # sales below $1000
    {"fans": 35, "pumps": 35, "bodies": 35, "expected_commission": 840.0},  # sales above $1800
    {"fans": 0, "pumps": 0, "bodies": 0, "expected_commission": 0.0},  # zero sales
]


for test_case in test_cases:
    fans = test_case["fans"]
    pumps = test_case["pumps"]
    bodies = test_case["bodies"]
    expected_commission = test_case["expected_commission"]
    commission = calculate_commission(fans, pumps, bodies)
    print(f"Test case: fans={fans}, pumps={pumps}, bodies={bodies}")
```

```
print(f"Expected commission: ${expected_commission:.2f}")
print(f"Actual commission: ${commission:.2f}")
if commission == expected_commission:
    print("Test passed!")
else:
    print("Test failed!")
print()
```

OUTPUT

```
Test case: fans=1, pumps=1, bodies=1
Expected commission: $10.00
Actual commission: $10.00
Test passed!

Test case: fans=70, pumps=80, bodies=90
Expected commission: $1260.00
Actual commission: $1420.00
Test failed!

Test case: fans=10, pumps=10, bodies=10
Expected commission: $120.00
Actual commission: $100.00
Test failed!

Test case: fans=20, pumps=20, bodies=20
Expected commission: $360.00
Actual commission: $260.00
Test failed!

Test case: fans=5, pumps=5, bodies=5
Expected commission: $60.00
Actual commission: $50.00
Test failed!

Test case: fans=35, pumps=35, bodies=35
Expected commission: $840.00
Actual commission: $560.00
Test failed!

Test case: fans=0, pumps=0, bodies=0
Expected commission: $0.00
```

AUTOMATED TESTING

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
import time
# Create Chrome options
options = Options()
options.add_experimental_option("detach", True)

# Create a Chrome driver instance
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()),
options=options)

# Open the Commission Calculator website
driver.get("https://www.calculatestuff.com/business/commission-calculator")

# Maximize the browser window
driver.maximize_window()

total_sales = driver.find_element(by=By.NAME, value="total_sales")
total_sales.send_keys("1500")

commission_percentage = driver.find_element(by=By.NAME,
value="commission_percentage")
commission_percentage.send_keys("15")

time.sleep(2)
calculator_button = driver.find_element(By.XPATH,
"/html/body/div[1]/div/div[2]/div[1]/div/div/div[1]/form/div[3]/div/input").cl
ick()

commission_result = driver.find_element(By.ID, "commissionResult")
print("Commission result:", commission_result.text)

time.sleep(5)

driver.quit()
```

2) Design, develop, code and run the program in any suitable language to implement the NextDate function. Analyse it from the perspective of equivalence class value testing, derive different test cases, execute these test cases and discuss the test results.

```
def next_date(year, month, day):
    if not (1 <= year <= 9999):
        raise ValueError("Invalid year")
    if not (1 <= month <= 12):
        raise ValueError("Invalid month")
    if not (1 <= day <= 31):
        raise ValueError("Invalid day")

    if month in [1, 3, 5, 7, 8, 10, 12]:
        if day < 31:
            return year, month, day + 1
        else:
            return year, month + 1, 1
    elif month == 2:
        if year % 4 == 0 and (year % 100!= 0 or year % 400 == 0):
            if day < 29:
                return year, month, day + 1
            else:
                return year, month + 1, 1
        else:
            if day < 28:
                return year, month, day + 1
            else:
                return year, month + 1, 1
    else:
        if day < 30:
            return year, month, day + 1
        else:
            return year, month + 1, 1
```

```
    if month == 12:
        return year + 1, 1, 1
    else:
        return year, month + 1, 1

# Test cases
print("Valid dates:")
print(next_date(2022, 6, 15))  # (2022, 6, 16)
print(next_date(2022, 6, 30))  # (2022, 7, 1)
print(next_date(2022, 12, 31))  # (2023, 1, 1)
print(next_date(2020, 2, 28))  # (2020, 2, 29)
print(next_date(2019, 2, 28))  # (2019, 3, 1)

print("\nInvalid dates:")
try:
    print(next_date(0, 6, 15))  # Error
except ValueError as e:
    print(e)

try:
    print(next_date(2022, 13, 15))  # Error
except ValueError as e:
    print(e)

try:
    print(next_date(2022, 6, 32))  # Error
except ValueError as e:
    print(e)
```

output

```
Valid dates:
(2022, 6, 16)
(2022, 7, 1)
(2022, 13, 1)
(2020, 2, 29)
(2019, 3, 1)

Invalid dates:
Invalid year
Invalid month
Invalid day
```

```
from selenium import webdriver
```

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Set up the Chrome driver
driver = webdriver.Chrome()

try:
    # Navigate to the webpage
    driver.get("https://www.calculator.net/date-calculator.html")

    # Wait for the webpage to load
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.TAG_NAME, "body"))
    )

    # Find the input fields
    year_input = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.NAME, "year"))
    )
    month_input = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.NAME, "month"))
    )
    day_input = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.NAME, "day"))
    )

    # Input the values
    year_input.send_keys("2022")
    month_input.send_keys("6")
    day_input.send_keys("15")

    # Click the submit button
    submit_button = WebDriverWait(driver, 10).until(
```

```
        EC.element_to_be_clickable((By.NAME, "submit"))
    )
    submit_button.click()

    # Wait for the result to appear
    result_element = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "result"))
    )

    # Get the result
    result = result_element.text

    # Print the result
    print("Next date:", result)

except Exception as e:
    print("Error:", e)

finally:
    # Close the browser
    driver.quit()
```

3) Design, develop, code and run the program in any suitable language to solve the commission problem. Analyse it from the perspective of decision table-based testing, derive different test cases, execute these test cases and discuss the test results.

```
def calculate_commission(sales_amount, commission_rate):
    if sales_amount < 0:
        return "Invalid sales amount"
    elif commission_rate < 0 or commission_rate > 1:
        return "Invalid commission rate"
    else:
        commission = sales_amount * commission_rate
        return commission


# Decision Table-based Testing
# | Sales Amount | Commission Rate | Expected Commission |
# |-------------|----------------|--------------------|
# | 1000        | 0.1            | 100                |
```

```
# | 2000       | 0.2          | 400                |
# | 500        | 0.05         | 25                 |
# | -100       | 0.1          | "Invalid sales amount" |
# | 1000       | 1.1          | "Invalid commission rate" |
# | 0          | 0.1          | 0                  |

# Test Cases
test_cases = [
    {"sales_amount": 1000, "commission_rate": 0.1, "expected_commission": 100},
    {"sales_amount": 2000, "commission_rate": 0.2, "expected_commission": 400},
    {"sales_amount": 500, "commission_rate": 0.05, "expected_commission": 25},
    {"sales_amount": -100, "commission_rate": 0.1, "expected_commission": "Invalid sales amount"},
    {"sales_amount": 1000, "commission_rate": 1.1, "expected_commission": "Invalid commission
rate"},
    {"sales_amount": 0, "commission_rate": 0.1, "expected_commission": 0},
]

# Execute Test Cases
for test_case in test_cases:
    sales_amount = test_case["sales_amount"]
    commission_rate = test_case["commission_rate"]
    expected_commission = test_case["expected_commission"]
    actual_commission = calculate_commission(sales_amount, commission_rate)
    if actual_commission == expected_commission:
        print(f"Test Case Passed: Sales Amount={sales_amount}, Commission
Rate={commission_rate}, Expected Commission={expected_commission}")
    else:
        print(f"Test Case Failed: Sales Amount={sales_amount}, Commission Rate={commission_rate},
Expected Commission={expected_commission}, Actual Commission={actual_commission}")
```

output

```
Test Case Passed: Sales Amount=1000, Commission Rate=0.1, Expected
Commission=100
Test Case Passed: Sales Amount=2000, Commission Rate=0.2, Expected
Commission=400
Test Case Passed: Sales Amount=500, Commission Rate=0.05, Expected
Commission=25
Test Case Passed: Sales Amount=-100, Commission Rate=0.1, Expected
Commission=Invalid sales amount
Test Case Passed: Sales Amount=1000, Commission Rate=1.1, Expected
Commission=Invalid commission rate
Test Case Passed: Sales Amount=0, Commission Rate=0.1, Expected Commission=0
```

4) Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume that the upper limit for the size of any side is 10. Derive test cases for your program based on boundary-value analysis, equivalence class partitioning and decision-table approach and execute the test cases and discuss the results.

```
def classify_triangle(side_a, side_b, side_c):
    """
    Classify a triangle based on its sides.

    Args:
        side_a (int): Side A of the triangle
        side_b (int): Side B of the triangle
        side_c (int): Side C of the triangle

    Returns:
        str: The type of triangle (Equilateral, Isosceles, Scalene, or Not a Triangle)
    """
    # Boundary Value Analysis
    if not (0 < side_a <= 10 and 0 < side_b <= 10 and 0 < side_c <= 10):
        return "Invalid input. Sides must be positive integers no larger than 10."

    # Equivalence Class Partitioning
    if side_a == side_b == side_c:
```

```
        return "Equilateral triangle"
    elif side_a == side_b or side_a == side_c or side_b == side_c:
        return "Isosceles triangle"
    else:
        return "Scalene triangle"

    # Decision Table
    decision_table = [
        ["Equilateral", side_a == side_b == side_c],
        ["Isosceles", side_a == side_b or side_a == side_c or side_b == side_c],
        ["Scalene", side_a!= side_b and side_a!= side_c and side_b!= side_c],
        ["Not a Triangle", not (side_a + side_b > side_c and side_a + side_c > side_b and side_b +
side_c > side_a)]
    ]

    for label, condition in decision_table:
        if condition:
            return label

# Test Cases
test_cases = [
    # Boundary Value Analysis
    (0, 0, 0),  # Invalid input
    (1, 1, 1),  # Equilateral triangle
    (1, 1, 10),  # Isosceles triangle
    (1, 10, 1),  # Isosceles triangle
    (10, 10, 10),  # Equilateral triangle
    (10, 10, 11),  # Not a Triangle

    # Equivalence Class Partitioning
    (2, 2, 2),  # Equilateral triangle
    (2, 3, 2),  # Isosceles triangle
    (2, 3, 4),  # Scalene triangle
```

```
    (10, 10, 5),  # Isosceles triangle
    (10, 5, 10),  # Isosceles triangle

    # Decision Table
    (3, 3, 3),  # Equilateral triangle
    (3, 4, 3),  # Isosceles triangle
    (3, 4, 5),  # Scalene triangle
    (5, 5, 10),  # Not a Triangle
]

for test_case in test_cases:
    print(f"Test Case: {test_case} -> {classify_triangle(*test_case)}")
```

Here's the explanation of the code:

1. **Boundary Value Analysis**: We check if the input values are within the valid range (0 < side <= 10). If not, we return an error message.
2. **Equivalence Class Partitioning**: We check if the input values form an equilateral, isosceles, or scalene triangle based on their equality.
3. **Decision Table**: We use a decision table to classify the triangle based on the input values. The table consists of four rows, each representing a possible outcome (Equilateral, Isosceles, Scalene, or Not a Triangle).

The test cases cover various scenarios, including boundary values, equivalence classes, and decision table outcomes. The output of each test case is printed to the console.

output

Test Case: (0, 0, 0) -> Invalid input. Sides must be positive integers no larger than 10.

Test Case: (1, 1, 1) -> Equilateral triangle

Test Case: (1, 1, 10) -> Isosceles triangle

Test Case: (1, 10, 1) -> Isosceles triangle

Test Case: (10, 10, 10) -> Equilateral triangle

Test Case: (10, 10, 11) -> Invalid input. Sides must be positive integers no larger than 10.

Test Case: (2, 2, 2) -> Equilateral triangle

Test Case: (2, 3, 2) -> Isosceles triangle

Test Case: (2, 3, 4) -> Scalene triangle

Test Case: (10, 10, 5) -> Isosceles triangle

Test Case: (10, 5, 10) -> Isosceles triangle

Test Case: (3, 3, 3) -> Equilateral triangle

Test Case: (3, 4, 3) -> Isosceles triangle

Test Case: (3, 4, 5) -> Scalene triangle

Test Case: (5, 5, 10) -> Isosceles triangle

5) Design, develop, code and run the program in any suitable language to solve the commission problem. Analyze it from the perspective of dataflow testing, derive different test cases, execute these test cases and discuss the test results.

```
def calculate_commission(sales_amount):
    """
    Calculate the commission earned by a salesperson based on their sales amount.

    Args:
        sales_amount (float): The sales amount.

    Returns:
        float: The commission earned.
    """
    if sales_amount <= 1000:
        commission_rate = 0.05
    elif 1001 <= sales_amount <= 5000:
        commission_rate = 0.07
    else:
        commission_rate = 0.10

    commission = sales_amount * commission_rate
    return commission
```

```
def main():
    sales_amount = float(input("Enter the sales amount: $"))
    commission = calculate_commission(sales_amount)
    print(f"The commission earned is: ${commission:.2f}")

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

output

```
$ python commission.py
Enter the sales amount: $500
The commission earned is: $25.00

$ python commission.py
Enter the sales amount: $3000
The commission earned is: $210.00

$ python commission.py
Enter the sales amount: $6000
The commission earned is: $600.00

$ python commission.py
Enter the sales amount: $1000
The commission earned is: $50.00

$ python commission.py
Enter the sales amount: $5000
The commission earned is: $350.00
```

6) Design, develop, code and run the program in any suitable language to implement the binary search algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.

```
# Binary Search Algorithm in Python


def binary_search(arr, target):
    """
    Searches for the target element in the sorted array using binary search.


    Args:
        arr (list): A sorted list of elements.
        target: The element to be searched.


    Returns:
        int: The index of the target element if found, None otherwise.
    """
    low = 0  # Initialize the low index to 0
    high = len(arr) - 1  # Initialize the high index to the last element of the array


    while low <= high:  # Continue the search until low is less than or equal to high
        mid = (low + high) // 2  # Calculate the middle index
        guess = arr[mid]  # Get the middle element


        if guess == target:  # If the middle element is the target, return the index
            return mid
        elif guess < target:  # If the middle element is less than the target, search the right half
            low = mid + 1
        else:  # If the middle element is greater than the target, search the left half
            high = mid - 1


    return None  # If the target is not found, return None
```

```
# Test cases
arr = [1, 2, 3, 4, 5]

# Test case 1: Target is found in the middle
print("Test case 1:")
print("Array:", arr)
print("Target:", 3)
print("Result:", binary_search(arr, 3))  # Output: 2

# Test case 2: Target is less than the middle element
print("\nTest case 2:")
print("Array:", arr)
print("Target:", 0)
print("Result:", binary_search(arr, 0))  # Output: None

# Test case 3: Target is greater than the middle element
print("\nTest case 3:")
print("Array:", arr)
print("Target:", 6)
print("Result:", binary_search(arr, 6))  # Output: None

# Test case 4: Empty list
print("\nTest case 4:")
arr = []
print("Array:", arr)
print("Target:", 3)
print("Result:", binary_search(arr, 3))  # Output: None

# Test case 5: List with one element
```

```
print("\nTest case 5:")
arr = [3]
print("Array:", arr)
print("Target:", 3)
print("Result:", binary_search(arr, 3))  # Output: 0

# Test case 6: List with one element
print("\nTest case 6:")
arr = [3]
print("Array:", arr)
print("Target:", 4)
print("Result:", binary_search(arr, 4))  # Output: None
```

OUTPUT:

Test case 1:

Array: [1, 2, 3, 4, 5]

Target: 3

Result: 2

Test case 2:

Array: [1, 2, 3, 4, 5]

Target: 0

Result: None

Test case 3:

Array: [1, 2, 3, 4, 5]

Target: 6

Result: None

Test case 4:

Array: []

Target: 3

Result: None

Test case 5:

Array: [3]

Target: 3

Result: 0

Test case 6:

Array: [3]

Target: 4

Result: None

PS C:\Users\nishc>