# LAB-8 IT314

## Software Testing

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

**Dharmi Patel 202201467**

# Q-1

The program's input consists of a triple: day, month, and year. The valid ranges are:

- **Day**: 1≤day≤31
- **Month**: 1≤month≤121
- **Year**: 1900≤year≤2015

The program should return either the previous date if the input is valid or an error message for invalid inputs.

### Test Design Strategy

We will use **Equivalence Partitioning (EP)** and **Boundary Value Analysis (BVA)** techniques to design test cases.

1. **Equivalence Partitioning (EP)**: The input space is divided into valid and invalid partitions. For example:
    - **Valid**: Dates within the range (e.g., 15/7/2010, 31/12/2015)
    - **Invalid**: Dates outside the range (e.g., day > 31, month > 12, year < 1900)
2. **Boundary Value Analysis (BVA)**: The boundaries of the input ranges are tested. For example:
    - **Valid boundaries**: 1 for day, 1 for month, 1900 for year
    - **Invalid boundaries**: 0 for day, 13 for month, 2016 for year, etc.

**Equivalence Partitioning (EP):**

| Test Case ID | Input Data (Day, Month, Year) | Expected Outcome | Valid/Invalid Input |
|---|---|---|---|
| EP-01 | 15, 7, 2010 | 14/7/2010 | Valid |
| EP-02 | 32, 7, 2010 | Error: Invalid Day | Invalid |
| EP-03 | 1, 3, 2000 | 29/2/2000 | Valid |
| EP-04 | 15, 13, 2010 | Error: Invalid Month | Invalid |
| EP-05 | 15, 7, 1899 | Error: Invalid Year | Invalid |

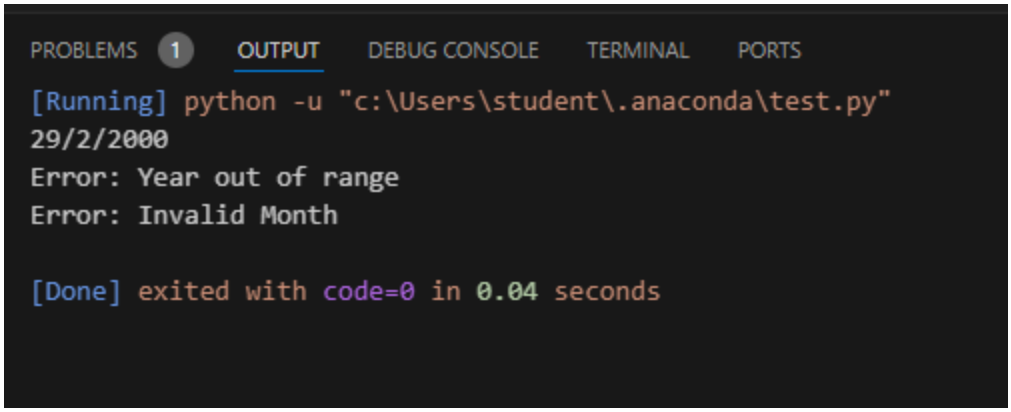
**Boundary Value Analysis (BVA):**

| Test Case ID | Input Data (Day, Month, Year) | Expected Outcome | Valid/Invalid Input |
|---|---|---|---|
| BVA-01 | 1, 7, 2010 | 30/6/2010 | Valid |
| BVA-02 | 31, 12, 2010 | 30/12/2010 | Valid |
| BVA-03 | 15, 1, 2010 | 14/1/2010 | Valid |
| BVA-04 | 15, 12, 2010 | 14/12/2010 | Valid |
| BVA-05 | 1, 1, 1900 | 31/12/1899 (Error) | Invalid |
| BVA-06 | 31, 12, 2015 | 30/12/2015 | Valid |
| BVA-07 | 0, 7, 2010 | Error: Invalid Day | Invalid |

| | | | |
|---|---|---|---|
| BVA-08 | 15, 13, 2010 | Error: Invalid Month | Invalid |

**Code:**

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

def previous_date(day, month, year):
    # Days in each month
    days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

    if year < 1900 or year > 2015:
        return "Error: Invalid Year"

    if month < 1 or month > 12:
        return "Error: Invalid Month"

    if is_leap_year(year):
        days_in_month[1] = 29
    if day < 1 or day > days_in_month[month - 1]:
        return "Error: Invalid Day"

    # Calculate previous day
    day -= 1
    if day == 0:
        month -= 1
        if month == 0:
            month = 12
            year -= 1
            if year < 1900:
                return "Error: Year out of range"
        day = days_in_month[month - 1]

    return f"{day}/{month}/{year}"

# Example test execution
print(previous_date(1, 3, 2000))  # Output: "29/2/2000" (Leap year)
print(previous_date(1, 1, 1900))  # Output: "Error: Year out of range"
print(previous_date(15, 13, 2010))  # Output: "Error: Invalid Month"
```




# Q.2. Programs:

**P1. The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.**

**Equivalence Partitioning Test Cases**

| Test Case ID | Input Array a | Search Value v | Expected Outcome | Valid/Invalid |
|---|---|---|---|---|
| EP-01 | [1, 2, 3, 4, 5] | 3 | 2 | Valid |
| EP-02 | [1, 2, 3, 4, 5] | 6 | -1 | Valid |
| EP-03 | [] | 1 | -1 | Valid |
| EP-04 | None | 5 | Error (or exception) | Invalid |
| EP-05 | [1, 2, 3] | "3" | Error (or exception) | Invalid |
| EP-06 | [1, 2, 3] | None | Error (or exception) | Invalid |
| EP-07 | "12345" | 3 | Error (or exception) | Invalid |
| EP-08 | [1, 2, 3] | [2, 3] | Error (or exception) | Invalid |

| | | | | |
|---|---|---|---|---|
| EP-09 | [1, 2, None, 4, 5] | 5 | Error (or exception) | Invalid |
| EP-10 | [10, 20, 30] | 20 | 1 | Valid |

**Boundary Value Analysis Test Cases**

| Test Case ID | Input Array | Search Value v | Expected Outcome | Valid/Invalid |
|---|---|---|---|---|
| **BVA-01** | [1] | 1 | 0 | Valid |
| **BVA-02** | [1, 2, 3, 4, 5] | 1 | 0 | Valid |
| **BVA-03** | [1, 2, 3, 4, 5] | 5 | 4 | Valid |
| **BVA-04** | [1, 2, 3, 4, 5] | 6 | -1 | Valid |
| **BVA-05** | [] | 1 | -1 | Valid |
| **BVA-06** | [5, 10, 15, 20] | 20 | 3 | Valid |
| **BVA-07** | [1, 2, 3, 4, 5] | 0 | -1 | Valid |
| **BVA-08** | [1, 2, 3] | 3 | 2 | Valid |
| **BVA-09** | [1, 2, 3] | 2 | 1 | Valid |
| **BVA-10** | [1, 2, 3, 4] | 4 | 3 | Valid |

**Code:**

```
def linearSearch(v, a):
    if a is None or not isinstance(a, list):
        raise ValueError("Input array must be a list and cannot be None.")

    for i in range(len(a)):
        if a[i] == v:
            return i
    return -1
```

```
linearSearch([1,2,3,4,],"123")
```

```
[Running] python -u "c:\Users\student\.anaconda\tempCodeRunnerFile.py"
Traceback (most recent call last):
  File "c:\Users\student\.anaconda\tempCodeRunnerFile.py", line 12, in <module>
    linearSearch([1,2,3,4,],"123")
  File "c:\Users\student\.anaconda\tempCodeRunnerFile.py", line 3, in linearSearch
    raise ValueError("Input array must be a list and cannot be None.")
ValueError: Input array must be a list and cannot be None.

[Done] exited with code=1 in 0.047 seconds
```

**P2. The function countItem returns the number of times a value v appears in an array of integers a.**

## Table for Equivalence Partitioning (EP)

| Test ID | Input Array | Search Value | Expected Outcome | Validity |
|---|---|---|---|---|
| EP-01 | [1, 2, 3, 1, 2, 1] | 1 | 3 | Valid |
| EP-02 | [1, 2, 3, 4, 5] | 6 | 0 | Valid |
| EP-03 | [] | 1 | 0 | Valid |
| EP-04 | None | 5 | Error | Invalid |
| EP-05 | [1, 2, 3] | "3" | Error | Invalid |

## Table for Boundary Value Analysis (BVA)

| Test ID | Input Array | Search Value | Expected Outcome | Validity |
|---|---|---|---|---|
| BVA-01 | [1] | 1 | 1 | Valid |
| BVA-02 | [1, 2, 3, 4, 5] | 1 | 1 | Valid |

| | | | | |
|---|---|---|---|---|
| **BVA-03** | **[1, 2, 3, 4, 5]** | **5** | **1** | **Valid** |
| **BVA-04** | **[1, 2, 3, 4, 5]** | **6** | **0** | **Valid** |
| **BVA-05** | **[]** | **1** | **0** | **Valid** |
| **BVA-06** | **[1, 1, 1, 1]** | **1** | **4** | **Valid** |
| **BVA-07** | **[1, 2, 3, 4, 5]** | **0** | **0** | **Valid** |
| **BVA-08** | **[1, 2, 3]** | **3** | **1** | **Valid** |
| **BVA-09** | **[1, 2, 3]** | **2** | **1** | **Valid** |
| **BVA-10** | **[1, 2, 3, 4]** | **4** | **1** | **Valid** |

**Code:**

```
def countItem(v, a):
    if a is None or not isinstance(a, list):
        raise ValueError("Input array must be a list and cannot be None.")

    count = 0
    for item in a:
        if item == v:
            count += 1
    return count



countItem([1,2,3,4,5,6],4)
countItem([1,2,3,4,5,6],"4")
```

```
[Running] python -u "c:\Users\student\.anaconda\test.py"
Traceback (most recent call last):
  File "c:\Users\student\.anaconda\test.py", line 12, in <module>
    countItem([1,2,3,4,5,6],4)
  File "c:\Users\student\.anaconda\test.py", line 3, in countItem
    raise ValueError("Input array must be a list and cannot be None.")
ValueError: Input array must be a list and cannot be None.

[Done] exited with code=1 in 0.038 seconds
```

**P3. The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.**

**Table for Equivalence Partitioning (EP)**

| Test ID | Input Array | Search Value | Expected Outcome | Validity |
|---|---|---|---|---|
| EP-01 | [1, 2, 3, 4, 5] | 3 | 2 | Valid |
| EP-02 | [1, 2, 3, 4, 5] | 6 | -1 | Valid |
| EP-03 | [] | 1 | -1 | Valid |
| EP-04 | None | 5 | Error | Invalid |
| EP-05 | [1, 2, 3] | "3" | Error | Invalid |

**Table for Boundary Value Analysis (BVA)**

| Test ID | Input Array | Search Value | Expected Outcome | Validity |
|---|---|---|---|---|
| BVA-01 | [1] | 1 | 0 | Valid |
| BVA-02 | [1, 2, 3, 4, 5] | 1 | 0 | Valid |
| BVA-03 | [1, 2, 3, 4, 5] | 5 | 4 | Valid |
| BVA-04 | [1, 2, 3, 4, 5] | 0 | -1 | Valid |
| BVA-05 | [1, 2, 3, 4, 5] | 6 | -1 | Valid |
| BVA-06 | [1, 1, 1, 1] | 1 | 0 | Valid |
| BVA-07 | [1, 2, 3, 4, 5] | 2 | 1 | Valid |
| BVA-08 | [1, 2, 3, 4, 5] | 4 | 3 | Valid |
| BVA-09 | [1, 2, 3, 4, 5] | 3 | 2 | Valid |
| BVA-10 | [] | 1 | -1 | Valid |

**Code:**

```
def binarySearch(v, a):
```

```
  if a is None or not isinstance(a, list):
     raise ValueError("Input array must be a list and cannot be None.")

  lo, hi = 0, 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


binarySearch([],7)
binarySearch([1,2,3,4,545],5)
```

```
[Running] python -u "c:\Users\student\.anaconda\test.py"
Traceback (most recent call last):
  File "c:\Users\student\.anaconda\test.py", line 19, in <module>
    binarySearch([],7)
  File "c:\Users\student\.anaconda\test.py", line 3, in binarySearch
    raise ValueError("Input array must be a list and cannot be None.")
ValueError: Input array must be a list and cannot be None.

[Done] exited with code=1 in 0.04 seconds
```

**P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal), scalene (no lengths equal), or invalid (impossible lengths).**

**Equivalence Partitioning (EP)**

| Test ID | Input Values | Expected Outcome | Validity |
|---|---|---|---|
| EP-01 | (3, 3, 3) | 0 | Valid |
| EP-02 | (3, 4, 3) | 1 | Valid |
| EP-03 | (3, 4, 5) | 2 | Valid |
| EP-04 | (1, 2, 3) | 3 | Valid |
| EP-05 | (0, 0, 0) | 3 | Valid |
| EP-06 | (-1, 2, 3) | 3 | Valid |
| EP-07 | (5, “5”, -5) | 3 | Invalid |

**Boundary Value Analysis (BVA)**

| Test ID | Input Values | Expected Outcome | Validity |
|---|---|---|---|
| BVA-01 | (1, 1, 1) | 0 | Valid |
| BVA-02 | (2, 2, 3) | 1 | Valid |
| BVA-03 | (2, 3, 4) | 2 | Valid |
| BVA-04 | (1, 1, 2) | 3 | Valid |
| BVA-05 | (0, 0, 1) | 3 | Valid |
| BVA-06 | (0, 0, 0) | 3 | Valid |
| BVA-07 | (1, 2, 2) | 1 | Valid |
| BVA-08 | (1, 2, 3) | 3 | Valid |
| BVA-09 | (3, 3, 6) | 3 | Valid |
| BVA-10 | (5, 5, 5) | 0 | Valid |

```
def triangle(a, b, c):
    # Ensure that the inputs are integers
```

```
    if not all(isinstance(x, int) for x in [a, b, c]):
        raise ValueError("All inputs must be integers.")

    # Check for the triangle inequality and classify the triangle
    if a >= b + c or b >= a + c or c >= a + b:
        return 3  # INVALID
    if a == b and b == c:
        return 0  # EQUILATERAL
    if a == b or a == c or b == c:
        return 1  # ISOSCELES
    return 2  # SCALENE


triangle("8",8,9)
```

```
[Running] python -u "c:\Users\student\.anaconda\test.py"
Traceback (most recent call last):
  File "c:\Users\student\.anaconda\test.py", line 16, in <module>
    triangle("8",8,9)
  File "c:\Users\student\.anaconda\test.py", line 4, in triangle
    raise ValueError("All inputs must be integers.")
ValueError: All inputs must be integers.
```

**P5. The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).**

## Equivalence Partitioning (EP) Test Cases

| Test ID | Input Values | Expected Outcome | Validity |
|---|---|---|---|
| EP-01 | ("pre", "prefix") | true | Valid |
| EP-02 | ("prefix", "pre") | false | Valid |
| EP-03 | ("", "hello") | true | Valid |
| EP-04 | ("hello", "hello") | true | Valid |
| EP-05 | ("he", "hello") | true | Valid |

| | | | |
|---|---|---|---|
| EP-06 | ("world", "hello") | false | Valid |
| EP-07 | ("longprefix", "short") | false | Valid |
| EP-08 | ("hello", "") | false | Valid |
| EP-09 | (123, 1234) | Error | Invalid |
| EP-10 | (123, "abc") | Error | Invalid |

## Boundary Value Analysis (BVA) Test Cases

| Test ID | Input Values | Expected Outcome | Validity |
|---|---|---|---|
| BVA-01 | ("a", "a") | true | Valid |
| BVA-02 | ("ab", "abc") | true | Valid |
| BVA-03 | ("abc", "abc") | true | Valid |
| BVA-04 | ("abc", "abcd") | true | Valid |
| BVA-05 | ("abcd", "abc") | false | Valid |
| BVA-06 | ("", "") | true | Valid |
| BVA-07 | ("long", "longprefix") | false | Valid |
| BVA-08 | ("prefix", "") | false | Valid |
| BVA-09 | ("a", "ab") | true | Valid |
| BVA-10 | ("abc", "a") | false | Valid |

```
def prefix(num1, num2):
    # Convert integers to strings for prefix comparison
    s1 = str(num1)
    s2 = str(num2)

    # Check if s1 is longer than s2
    if len(s1) > len(s2):
        return False  # num1 cannot be a prefix of num2

    # Check for prefix match
```

```
    for i in range(len(s1)):
        if s1[i] != s2[i]:
            return False  # Mismatch found
    return True  # num1 is a prefix of num2


if prefix(123,1234):
    print("true")
else:
    print("false")
```

```
[Running] python -u "c:\Users\student\.anaconda\test.py"
true

[Done] exited with code=0 in 0.042 seconds
```

**P6: Consider again the triangle classification program (P4) with a slightly different specification: The program reads floating values from the standard input. The three values A, B, and C are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral, or right angled. Determine the following for the above program:**

**a) Identify the Equivalence Classes for the System**

1. **Equivalence Class for Valid Triangles:**
    - **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, B ≠ C, A ≠ C).**
    - **Right Triangle: Fulfills the Pythagorean theorem ($A^2 + B^2 = C^2$ or any permutation).**
2. **Equivalence Class for Invalid Triangles:**
    - **Non-Triangle: The sides do not satisfy the triangle inequality (A + B ≤ C, A + C ≤ B, B + C ≤ A).**

- **Non-Positive Values: One or more sides are less than or equal to zero ($A \leq 0$, $B \leq 0$, $C \leq 0$).**

## b) Identify Test Cases to Cover the Identified Equivalence Classes

| Test Case ID | Input Values | Expected Outcome | Equivalence Class |
|---|---|---|---|
| TC-01 | (3.0, 3.0, 3.0) | Equilateral Triangle | Valid Equilateral |
| TC-02 | (3.0, 3.0, 4.0) | Isosceles Triangle | Valid Isosceles |
| TC-03 | (3.0, 4.0, 5.0) | Scalene Triangle | Valid Scalene |
| TC-04 | (5.0, 5.0, 5.0) | Equilateral Triangle | Valid Equilateral |
| TC-05 | (5.0, 5.0, 8.0) | Isosceles Triangle | Valid Isosceles |
| TC-06 | (3.0, 4.0, 6.0) | Scalene Triangle | Valid Scalene |
| TC-07 | (1.0, 2.0, 3.0) | Non-Triangle | Invalid Non-Triangle |
| TC-08 | (3.0, 4.0, 7.0) | Non-Triangle | Invalid Non-Triangle |
| TC-09 | (0.0, 4.0, 5.0) | Non-Positive Input | Invalid Non-Positive |
| TC-10 | (-1.0, 2.0, 3.0) | Non-Positive Input | Invalid Non-Positive |

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

| Test Case ID | Input Values | Expected Outcome |
|---|---|---|
| BCA-01 | (2.0, 3.0, 4.0) | Scalene Triangle |
| BCA-02 | (2.0, 2.0, 4.0) | Non-Triangle |
| BCA-03 | (1.0, 1.0, 2.0) | Non-Triangle |

## d) Boundary Condition: A = C (Isosceles Triangle)

| Test Case ID | Input Values | Expected Outcome |
|---|---|---|
| BCB-01 | (3.0, 3.0, 4.0) | Isosceles Triangle |
| BCB-02 | (4.0, 3.0, 4.0) | Isosceles Triangle |
| BCB-03 | (1.0, 1.0, 2.0) | Non-Triangle |

## e) Boundary Condition: A = B = C (Equilateral Triangle)

| Test Case ID | Input Values | Expected Outcome |
|---|---|---|
| BCC-01 | (3.0, 3.0, 3.0) | Equilateral Triangle |
| BCC-02 | (2.0, 2.0, 2.0) | Equilateral Triangle |
| BCC-03 | (0.0, 0.0, 0.0) | Non-Positive Input |

## f) Boundary Condition: $A^2 + B^2 = C^2$ (Right Triangle)

| Test Case ID | Input Values | Expected Outcome |
|---|---|---|
| BCD-01 | (3.0, 4.0, 5.0) | Right Triangle |
| BCD-02 | (5.0, 12.0, 13.0) | Right Triangle |
| BCD-03 | (1.0, 1.0, 1.0) | Non-Triangle |

## g) Non-Triangle Case: Test Cases to Explore the Boundary

| Test Case ID | Input Values | Expected Outcome |
|---|---|---|
| NTC-01 | (1.0, 2.0, 3.0) | Non-Triangle |
| NTC-02 | (3.0, 3.0, 7.0) | Non-Triangle |
| NTC-03 | (5.0, 10.0, 20.0) | Non-Triangle |

## h) Non-Positive Input: Identify Test Points

| Test Case ID | Input Values | Expected Outcome |
|---|---|---|
| NPI-01 | (0.0, 5.0, 6.0) | Non-Positive Input |
| NPI-02 | (-1.0, 4.0, 5.0) | Non-Positive Input |
| NPI-03 | (1.0, 0.0, 1.0) | Non-Positive Input |
| NPI-04 | (1.0, 1.0, -1.0) | Non-Positive Input |