PSG COLLEGE OF TECHNOLOGY, COIMBATORE-4

DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES

BSc COMPUTER SYSTEMS & DESIGN 6 Semester

15X602 SOFTWARE TESTING

Problem Sheet-I CODE REVIEW PROCESS

You are a programmer in AppSoftk software solutions. Write the coding for the requirements given below. Review the code, Test your code with Valid and Invalid input sequences for all the applications.

PROBLEM 1

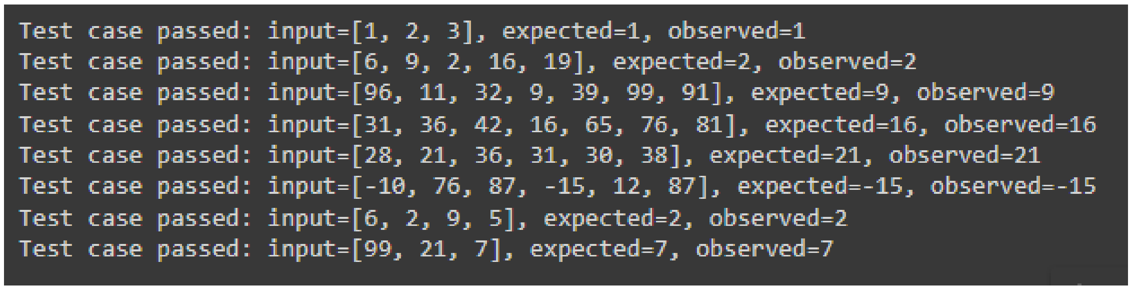
1.1 To find the minimum number among a list of array values.

Requirement Specification

The program termed minimum reads a set of integers and prints the smallest integer. Based on the critical situations given below modify and review your code for a code review process. The inputs are stored in a file. Design the test cases to read the input from the file and show if the observed output is matched with the expected output, and if so treat the test case as pass, and otherwise fail.

Test case	Size of inputs	Input values	Expected output	Observed output	Test case Pass/Fail
1.	3	1 2 3	1	1	PASS
1	5	6,9,2,16,19	2	2	PASS
2	7	96,11,32,9,39,99,91	9	9	PASS
3	7	31,36,42,16,65,76,81	16	16	PASS
4	6	28,21,36,31,30,38	21	21	PASS
5	6	-10,76,87,-15,12,87	-15	-15	PASS
6	4	6,2,9,5	2	2	PASS

7	3	99,21,7	7	7	PASS



```
Test case passed: input=[1, 2, 3], expected=1, observed=1
Test case passed: input=[6, 9, 2, 16, 19], expected=2, observed=2
Test case passed: input=[96, 11, 32, 9, 39, 99, 91], expected=9, observed=9
Test case passed: input=[31, 36, 42, 16, 65, 76, 81], expected=16, observed=16
Test case passed: input=[28, 21, 36, 31, 30, 38], expected=21, observed=21
Test case passed: input=[-10, 76, 87, -15, 12, 87], expected=-15, observed=-15
Test case passed: input=[6, 2, 9, 5], expected=2, observed=2
Test case passed: input=[99, 21, 7], expected=7, observed=7
```

Some typical and critical situations are given below

(i) A very short list (of inputs) with the size of 1,2, or 3 elements

(ii) An empty list i.e of size 0

(iii) A list where the minimum element is negative

(iv) A list where the minimum element are the first or last element

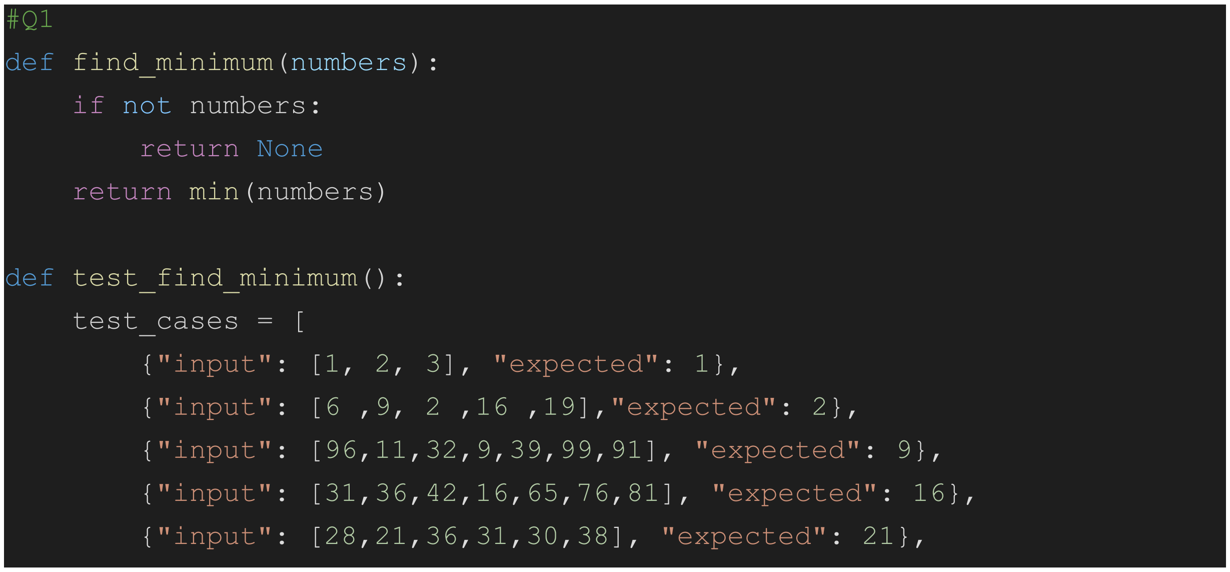
(v) A list where all elements are negative

(vi) A list where some elements are real numbers

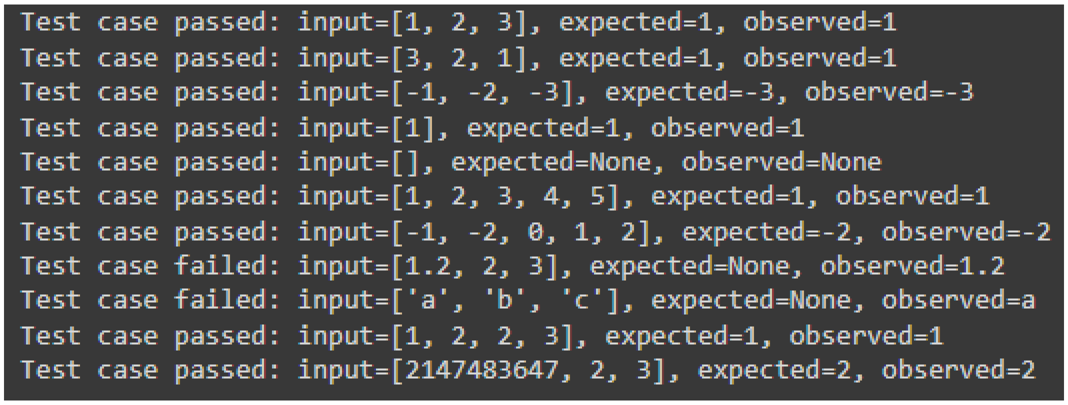
(vii) A list where some elements are alphabetic characters

(viii) A list with duplicate elements

(ix) A list where one element has a value greater than the maximum permissible of an integer



```
#Q1
def find_minimum(numbers):
    if not numbers:
        return None
    return min(numbers)

def test_find_minimum():
    test_cases = [
        {"input": [1, 2, 3], "expected": 1},
        {"input": [6 ,9, 2 ,16 ,19],"expected": 2},
        {"input": [96,11,32,9,39,99,91], "expected": 9},
        {"input": [31,36,42,16,65,76,81], "expected": 16},
        {"input": [28,21,36,31,30,38], "expected": 21},
        {"input": [-10,76,87,-15,12,87],"expected": -15},
        {"input": [6,2,9,5], "expected": 2},
        {"input": [99,21,7], "expected": 7},

        {"input": [1, 2, 3], "expected": 1},
        {"input": [3, 2, 1], "expected": 1},
        {"input": [-1, -2, -3], "expected": -3},
        {"input": [1], "expected": 1},
        {"input": [], "expected": None},
        {"input": [1, 2, 3, 4, 5], "expected": 1},
        {"input": [-1, -2, 0, 1, 2], "expected": -2},
        {"input": [1.2, 2, 3], "expected": None},
        {"input": ["a", "b", "c"], "expected": None},
        {"input": [1, 2, 2, 3], "expected": 1},
        {"input": [2147483647, 2, 3], "expected": 2},
    ]

    for test_case in test_cases:
        input_list = test_case["input"]
        expected_output = test_case["expected"]
        observed_output = find_minimum(input_list)

        if observed_output == expected_output:
            print(f"Test case passed: input={input_list},
expected={expected_output}, observed={observed_output}")
        else:
            print(f"Test case failed: input={input_list},
expected={expected_output}, observed={observed_output}")

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



```
Test case passed: input=[1, 2, 3], expected=1, observed=1
Test case passed: input=[3, 2, 1], expected=1, observed=1
Test case passed: input=[-1, -2, -3], expected=-3, observed=-3
Test case passed: input=[1], expected=1, observed=1
Test case passed: input=[], expected=None, observed=None
Test case passed: input=[1, 2, 3, 4, 5], expected=1, observed=1
Test case passed: input=[-1, -2, 0, 1, 2], expected=-2, observed=-2
Test case failed: input=[1.2, 2, 3], expected=None, observed=1.2
Test case failed: input=['a', 'b', 'c'], expected=None, observed=a
Test case passed: input=[1, 2, 2, 3], expected=1, observed=1
Test case passed: input=[2147483647, 2, 3], expected=2, observed=2
```

1.2 To find the Maximum number among the list of array values. Test cases and critical situations are same, compute for maximum as above problem. The test cases are stored in file. After, in your program read the test case file with observed output is match, say test case should be passing, and otherwise fail.

PROBLEM 2

A Simple example of code is given. Write an improved code by identifying the defects

```
public class Account {
double principal,rate; int daysActive,accountType;
public static final int STANDARD=0, BUDGET=1, PREMIUM=2,
PREMIUM_PLUS=3;
}
...
public static double calculateFee (Account[] accounts)
{
double totalFee = 0.0;
Account account;
for (int i=0;i<accounts.length;i++) {
account=accounts[i];
if(account.accountType==Account.PREMIUM|| account.accountType
== Account.PREMIUM_PLUS )
totalFee += .0125 * ( account.principal*Math.pow
(account.rate,(account.daysActive/365.25))
- account.principal);
}
return totalFee;
}
```

Problems and defects that is to be discovered and removed:

1. Many comments are missing or not informative enough.
2. The indentation style is not consistent and makes it more difficult to read the code.
3. Some variables look like class names.
4. Some comments are misleading.
5. Many methods are too large and should be broken into smaller methods.
6. Some variables are never used.
7. Perhaps regular expressions should be assigned to strings to improve the readability.

Reference: Code Review (Book: "Applied Software Project Management" by Andrew Stellman and Jennifer Greene)

PROBLEM 3

Develop a program for computing the Factorial of 'N' Number. Review your code to generate positive and negative test cases.

Requirement Specification:

- Factorials are represented by n!

1. Examples: 3! = 1 * 2 * 3 = 6
   4! = 1 * 2 * 3 * 4 = 24
   5! = 1 * 2 * 3 * 4 * 5 = 120

2. A factorial program is easy to test because you are simply testing the input value for an integer variable related to a single text box and then verifying its output value.

3. Determine the maximum integer value that the computer can calculate its factorial. This value becomes the upper boundary.

4. **POSITIVE TEST CASES**

   1) Submit integer value at lower boundary [0]
   2) Submit integer value at lower boundary +1 [1]
   3) Submit integer value at upper boundary -1
   4) Submit integer value at upper boundary

5. **NEGATIVE TEST CASES**

   5) Submit integer value at upper boundary +1
   6) Submit integer value with leading 0
   7) Submit integer value with leading plus sign [+]
   8) Submit integer value with leading minus sign [-]
   9) Submit integer value with leading space

10) Submit number with decimal point
12) Submit number with the letter 'e' in it
13) Submit alpha character(s)
14) Submit special character(s) [ex: !@#$^]
15) Submit with only space
16) Submit with nothing entered

Problem Sheet-2

WHITE BOX TESTING

Problem 1:

Consider the following software code to perform white box testing which involves testing of

- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis

Sample code

```
void countChar (int& vocNumber, int& totalNumber){
char chr;
cin>> chr;
while ((chr >=`A´) &&
(chr <=`Z´) &&
(totalNumber <INT_MAX)){
totalNumber +=1;
if ((chr ==`A´)||
(chr ==`E´) ||
(chr ==`I´) ||
(chr ==`O´) ||
(chr ==`U´)){
```

```
vocNumber +=1;
}
cin>> chr
}
}
```

- Identify the above code and generate all possible valid and invalid test cases
- Execute all statements in a program at least once under the test cases to compute 100% statement coverage.
- Identify the independent path for path coverage.
- Each atomic condition must be true at least once and false at least once to achieve 100% Decision coverage.

Problem 2:

Consider the program given below. Draw the program (Flow) graph. Derive test cases so that100% statement coverage and path coverage is achieved.

- Each independent path in the code is taken for testing.
- Every possible statement in the code to be tested at least once

```
/*Program to validate input data*/
#include<stdio.h>
#include<string.h>
#include<conio.h>
1. void main()
2. {
3. charfname[30],address[100],Email[100];
4. int valid=1,flag=1;
5. clrscr();
6. printf("Enter first name:");
7. scanf("%s",fname);
8. printf("\nEnter address:");
```

```c
scanf("%s",address);
printf("\nEnter Email:");
scanf("%s",Email);
if(strlen(fname)<4||strlen(fname)>30){
printf("\nInvalid first name");
valid=0;
}
if(strlen(address)<4||strlen(address)>100){
printf("\nInvalid address length");
valid=0;
}
if(strlen(Email)<8||strlen(Email)>100){
printf("\nInvalid Email length");
flag=0;
valid=0;
}
if(flag==1){
if(strchr(Email,'.')==0||strchr(Email,'@')==0){
printf("\nEmail must contain . and @ characters");
valid=0;
}
}
if(valid) {
printf("\nFirst name: %s \t Address: %s \t Email: %s",fname,address,Email);
}
getch();

}
```

Problem 3:

Consider a program for classification of a triangle based on Figure 3.1. Its input is a triple of positive integers (a,b,c) and the input parameters are greater than zero and less than or equal to 100.

Right angled triangle : $c^2=a^2+b^2$ or $a^2=b^2+c^2$ or $b^2=c^2+a^2$

Obtuse angled triangle : $c^2>a^2+b^2$ or $a^2>b^2+c^2$ or $b^2>c^2+a^2$

Acute angled triangle : $c^2<a^2+b^2$ or $a^2<b^2+c^2$ or $b^2<c^2+a^2$

1.	Right angled triangle
2.	Obtuse angled triangle
3.	Acute angled triangle
4.	Invalid triangle
5.	Input values out of range

1. Generate test case that represents a valid right angled triangle, obtuse triangle and Acute angled triangle?
2. Test case with three integers such that the sum of two is equal to the third?
3. Test case in which one side has a zero value?
4. Test case in which one side has a negative value?
5. Test case with non integer value
6. Test case with wrong number of values (2 or less, four or more).

Write a set of test requirements that achieves statement and path coverage

Reference : G.J. Myers, The Art of Software Testing ,John Wiley and Sons, 1979.

Problem 4:

There are 100 pirates(a person who sails in a ship and attacks other ships in order to steal from them) on the ship. In statistical terms this means we have a population of 100. If we know the amount of gold coins each of the 100 pirates have, we use the **standard deviation equation for an entire population.** Suppose if we don't know the amount of gold coins each of the 100 pirates have? For example, we only had enough time to ask 5 pirates how many gold coins they have. In statistical terms this means we have a sample size of 5 and in this case we use the **standard deviation equation for a sample of a population**

$$s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{(n - 1)}}$$

s = standard deviation x_i = each value in the sample $\bar{x}$ = the arithmetic mean

n = the number of values (the sample size) $\sum (x_i - \bar{x})^2$ = The sum of $(x_i - \bar{x})^2$ for all values

Write a code to accept input for calculating the standard deviation. It has consciously been seeded with defects. If you were asked to perform white box testing on this program, identify some of the defects in the program. Also list the methodology you used to identify these defects.

- For the above program ,draw the flow chart and flow graph
- Identify the independent path from the flowgraph
- Compute the cyclomatic complexity of the program using all the three methods.

Problem 5:

Write a program for determination of day of the week. Its input is at triple of positive Integers (day, month and year) from the interval

$1 \leq days \leq 31$

$1 \leq month \leq 12$

$2000 \leq 2019$ The output may be the day of a week.

Find all du-paths and identify those du-paths that are definition clear. Also find all du-paths all-uses, all-definitions and generate test cases for these paths.

Problem 6:

Write a code for inserting and deleting elements from a doubly linked list. Suggest a set of test data to cover each and every statement of the program

- Draw a control graph (to determine different program paths)
- Calculate Cyclomatic complexity (metrics to determine the number of independent paths)
- Find a basis set of paths
- Generate test cases to exercise each path

Problem Sheet - 3

BLACK BOX TESTING

Problem 1

Password field accepts minimum 6 characters and maximum 12 characters. [6-12]

Write Test Cases considering values from valid region and each invalid region and values which define exact boundary.

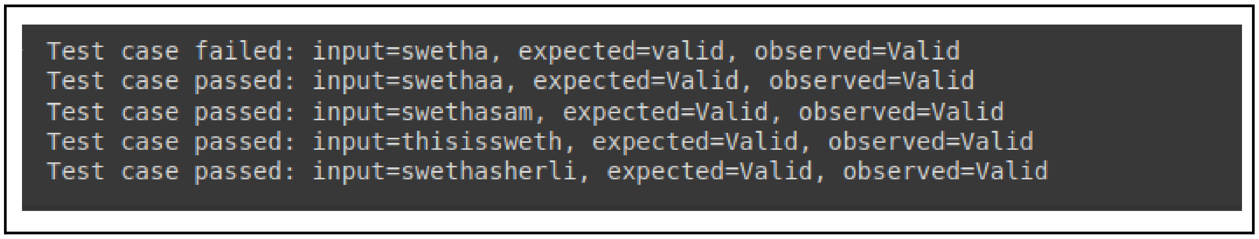
You need to execute 5 Test Cases for the given problem.

1. Consider password length less than 6
2. Consider password of length exactly 6
3. Consider password of length between 7 and 11
4. Consider password of length exactly 12
5. Consider password of length more than 12

Design the problem using Boundary Value Analysis (BVA) Testing

Test case	Sizeof inputs	Input values	Expected output	Observed output	Test case Pass/Fail
1.	6	swetha	Valid	Valid	PASS
2.	7	swethaa	Valid	Valid	PASS
3	9	swethasam	valid	valid	PASS
4	11	thisissweth	valid	valid	PASS
5	12	swethasherli	valid	valid	PASS

```python
# Test Case 1: Password length 6
test_case_1 = {"input": "swetha", "expected": "valid"}

# Test Case 2: Password length exactly 7
test_case_2 = {"input": "swethaa", "expected": "Valid"}

# Test Case 3: Password length 9
test_case_3 = {"input": "swethasam", "expected": "Valid"}

# Test Case 4: Password length 11 (less than 12)
test_case_4 = {"input": "thisissweth", "expected": "Valid"}

# Test Case 5: Password length exactly 12 (boundary)
test_case_5 = {"input": "swethasherli", "expected": "Valid"}


test_cases = [test_case_1, test_case_2, test_case_3, test_case_4,
test_case_5]

def validate_password(password):
    if 6 <= len(password) <= 12:
        return "Valid"
    else:
        return "Invalid"

for i in test_cases:
    input_password = i["input"]
    expected_output = i["expected"]
    observed_output = validate_password(input_password)

    if observed_output == expected_output:
        print(f"Test case passed: input={input_password},
expected={expected_output}, observed={observed_output}")
    else:
        print(f"Test case failed: input={input_password},
expected={expected_output}, observed={observed_output}")
```



```
Test case failed: input=swetha, expected=valid, observed=Valid
Test case passed: input=swethaa, expected=Valid, observed=Valid
Test case passed: input=swethasam, expected=Valid, observed=Valid
Test case passed: input=thisissweth, expected=Valid, observed=Valid
Test case passed: input=swethasherli, expected=Valid, observed=Valid
```

Problem2

A store in the city offers different discounts depending on the purchases made by the individual. In order to test the software that calculates the discounts, we can identify the ranges of purchase values that earn the different discounts. For example, if a purchase is in the range of $1 up to $50 has no discounts, a purchase over $50 and up to $200 has a 5% discount, and purchases of $201 and up to $500 have a 10% discounts, and purchases of $501 and above have a 15% discounts.

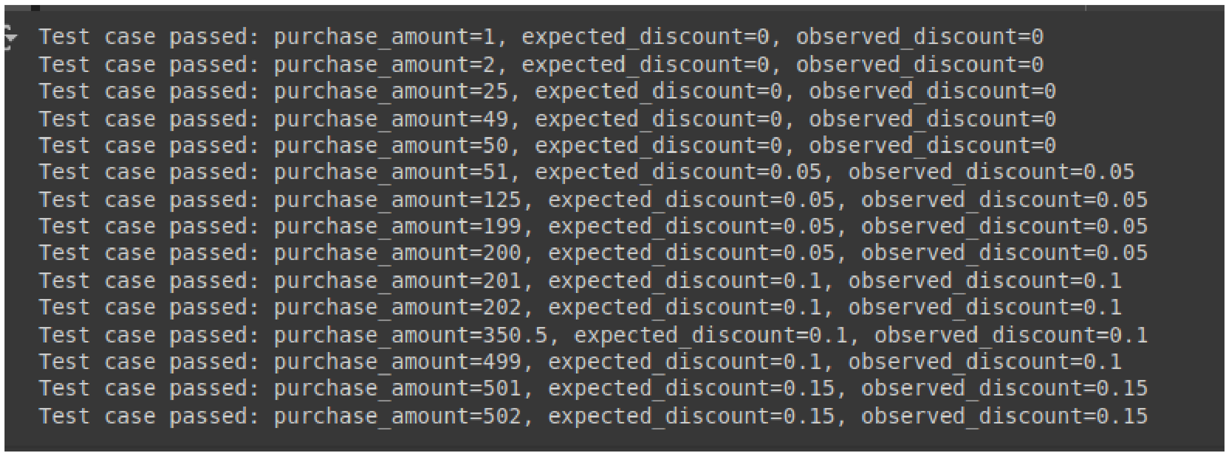
Design the test case using Boundary Value Analysis (BVA) Testing.

Test case	Input values	Expected output	Observed output	Test case Pass/Fail
1.	1	0	0	PASS
2.	2	0	0	PASS
3	25	0	0	PASS
4	49	0	0	PASS
5	50	0	0	PASS
6	51	0.05	0.05	PASS
7	125	0.05	0.05	PASS
8	199	0.05	0.05	PASS
9	200	0.05	0.05	PASS
10	201	0.10	0.10	PASS

11	202	0.10	0.10	PASS
12	305.5	0.10	0.10	PASS
13	499	0.10	0.10	PASS
14	501	0.15	0.15	PASS
15	502	0.15	0.15	PASS

```python
test_cases = [
   {"purchase_amount": 1, "expected_discount": 0},
   {"purchase_amount": 2, "expected_discount": 0},
   {"purchase_amount": 25, "expected_discount": 0},
   {"purchase_amount": 49, "expected_discount": 0},
   {"purchase_amount": 50, "expected_discount": 0},

   {"purchase_amount": 51, "expected_discount": 0.05},
   {"purchase_amount": 125, "expected_discount": 0.05},
   {"purchase_amount": 199, "expected_discount": 0.05},
   {"purchase_amount": 200, "expected_discount": 0.05},

   {"purchase_amount": 201, "expected_discount": 0.10},
   {"purchase_amount": 202, "expected_discount": 0.10},
   {"purchase_amount": 350.5, "expected_discount": 0.10},
   {"purchase_amount": 499, "expected_discount": 0.10},

   {"purchase_amount": 501, "expected_discount": 0.15},
   {"purchase_amount": 502, "expected_discount": 0.15},

]

def calculate_discount(purchase_amount):
   if purchase_amount < 50:
       return 0
   elif 50 <= purchase_amount <=200:
       return 0.05
   elif 201 <= purchase_amount < 500:
       return 0.10
```

```python
    else:
        return 0.15

# Run the test cases and compare expected vs. actual discounts
for test_case in test_cases:
    purchase_amount = test_case["purchase_amount"]
    expected_discount = test_case["expected_discount"]
    observed_discount = calculate_discount(purchase_amount)

    if observed_discount == expected_discount:
        print(f"Test case passed: purchase_amount={purchase_amount},
expected_discount={expected_discount},
observed_discount={observed_discount}")
    else:
        print(f"Test case failed: purchase_amount={purchase_amount},
expected_discount={expected_discount},
observed_discount={observed_discount}")
```



```
Test case passed: purchase_amount=1, expected_discount=0, observed_discount=0
Test case passed: purchase_amount=2, expected_discount=0, observed_discount=0
Test case passed: purchase_amount=25, expected_discount=0, observed_discount=0
Test case passed: purchase_amount=49, expected_discount=0, observed_discount=0
Test case passed: purchase_amount=50, expected_discount=0, observed_discount=0
Test case passed: purchase_amount=51, expected_discount=0.05, observed_discount=0.05
Test case passed: purchase_amount=125, expected_discount=0.05, observed_discount=0.05
Test case passed: purchase_amount=199, expected_discount=0.05, observed_discount=0.05
Test case passed: purchase_amount=200, expected_discount=0.05, observed_discount=0.05
Test case passed: purchase_amount=201, expected_discount=0.1, observed_discount=0.1
Test case passed: purchase_amount=202, expected_discount=0.1, observed_discount=0.1
Test case passed: purchase_amount=350.5, expected_discount=0.1, observed_discount=0.1
Test case passed: purchase_amount=499, expected_discount=0.1, observed_discount=0.1
Test case passed: purchase_amount=501, expected_discount=0.15, observed_discount=0.15
Test case passed: purchase_amount=502, expected_discount=0.15, observed_discount=0.15
```

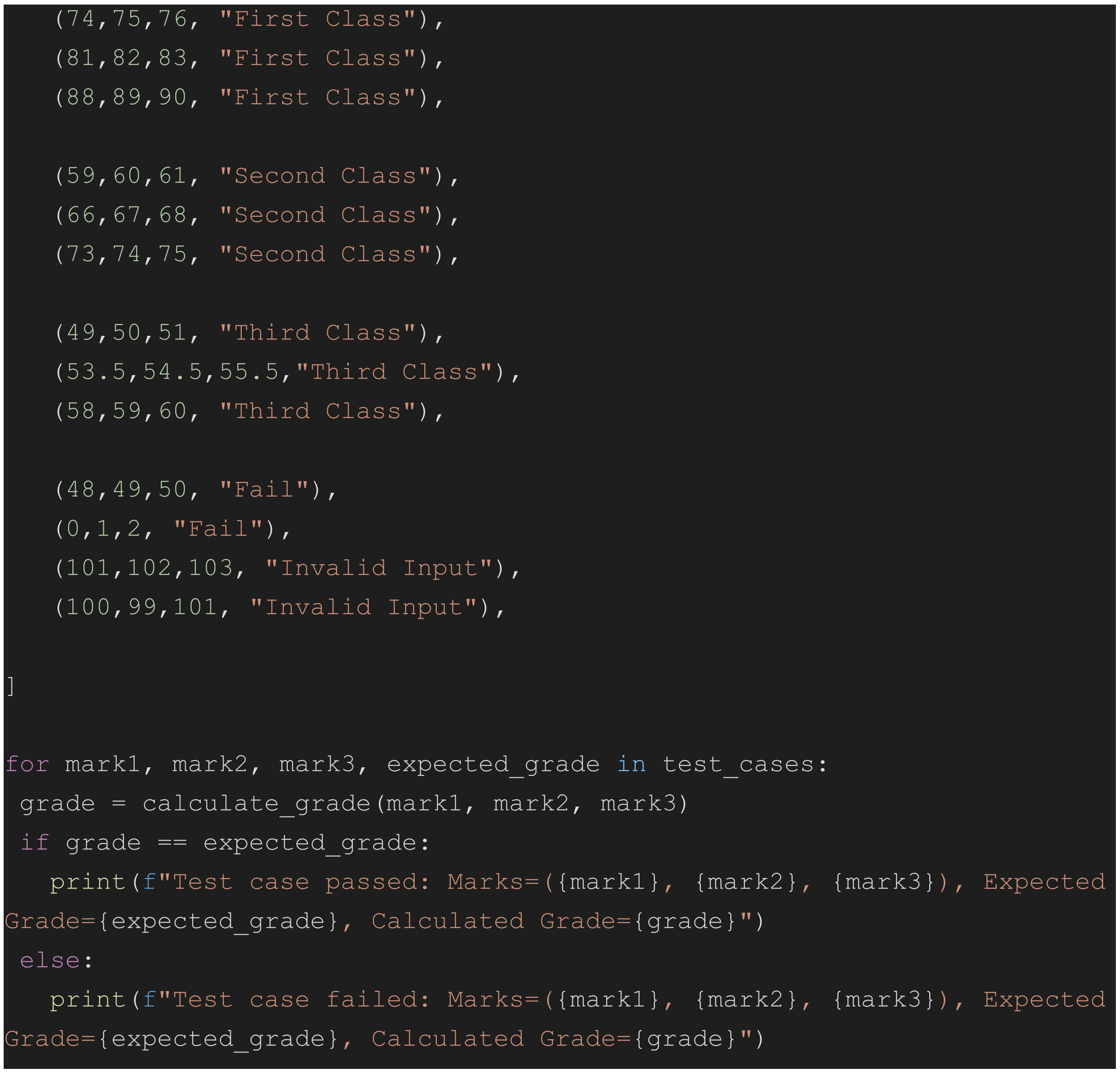
Problem 3

Consider a program for determining the grade of a student based on the marks in three subjects. Its input is a triple of positive integers (mark1, mark2, mark3) and values for each of these may be from interval [0-100]. The total marks are the average of marks obtained in three subjects. The grade is calculated based on the condition given in the table

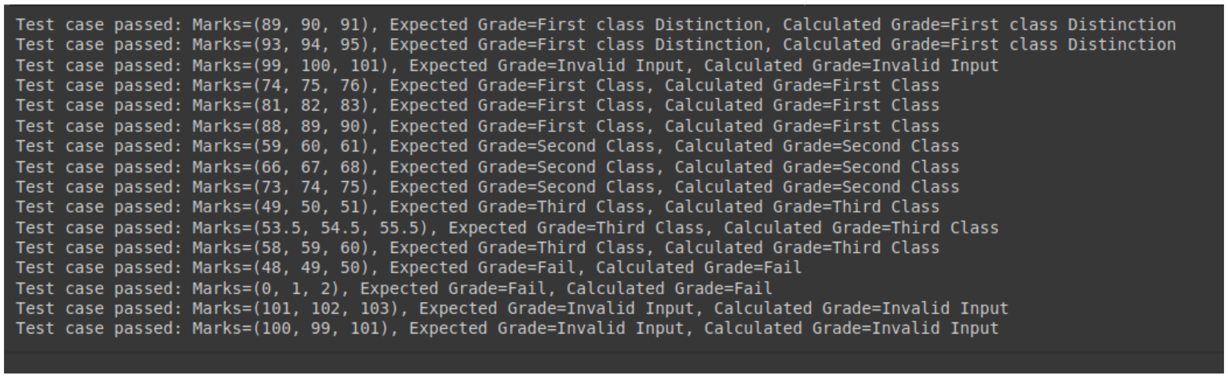
Marks Obtained	**Grade**
90-100	First class Distinction
75 - 89	First Class
60 - 74	Second Class
50- 59	Third Class
Below 50	Fail

Design the test case using Robustness Testing

```
def calculate_grade(mark1, mark2, mark3):

 if 0 <= mark1 <= 100 and 0 <= mark2 <= 100 and 0 <= mark3 <= 100:
   total_marks = (mark1 + mark2 + mark3) / 3

   if 90<= total_marks <=100:
     return "First class Distinction"
   elif 75<= total_marks <= 89:
     return "First Class"
   elif 60<= total_marks <=74:
     return "Second Class"
   elif 50<= total_marks <= 59:
     return "Third Class"
   elif 0<= total_marks < 50:
     return "Fail"
 else:
   return "Invalid Input"

# Example usage and test cases
test_cases = [
   (89,90, 91, "First class Distinction"),
   (93,94,95, "First class Distinction"),
   (99,100,101, "Invalid Input"),
```



```
    (74,75,76, "First Class"),
    (81,82,83, "First Class"),
    (88,89,90, "First Class"),

    (59,60,61, "Second Class"),
    (66,67,68, "Second Class"),
    (73,74,75, "Second Class"),

    (49,50,51, "Third Class"),
    (53.5,54.5,55.5,"Third Class"),
    (58,59,60, "Third Class"),

    (48,49,50, "Fail"),
    (0,1,2, "Fail"),
    (101,102,103, "Invalid Input"),
    (100,99,101, "Invalid Input"),

]

for mark1, mark2, mark3, expected_grade in test_cases:
 grade = calculate_grade(mark1, mark2, mark3)
 if grade == expected_grade:
   print(f"Test case passed: Marks=({mark1}, {mark2}, {mark3}), Expected
Grade={expected_grade}, Calculated Grade={grade}")
 else:
   print(f"Test case failed: Marks=({mark1}, {mark2}, {mark3}), Expected
Grade={expected_grade}, Calculated Grade={grade}")
```



```
Test case passed: Marks=(89, 90, 91), Expected Grade=First class Distinction, Calculated Grade=First class Distinction
Test case passed: Marks=(93, 94, 95), Expected Grade=First class Distinction, Calculated Grade=First class Distinction
Test case passed: Marks=(99, 100, 101), Expected Grade=Invalid Input, Calculated Grade=Invalid Input
Test case passed: Marks=(74, 75, 76), Expected Grade=First Class, Calculated Grade=First Class
Test case passed: Marks=(81, 82, 83), Expected Grade=First Class, Calculated Grade=First Class
Test case passed: Marks=(88, 89, 90), Expected Grade=First Class, Calculated Grade=First Class
Test case passed: Marks=(59, 60, 61), Expected Grade=Second Class, Calculated Grade=Second Class
Test case passed: Marks=(66, 67, 68), Expected Grade=Second Class, Calculated Grade=Second Class
Test case passed: Marks=(73, 74, 75), Expected Grade=Second Class, Calculated Grade=Second Class
Test case passed: Marks=(49, 50, 51), Expected Grade=Third Class, Calculated Grade=Third Class
Test case passed: Marks=(53.5, 54.5, 55.5), Expected Grade=Third Class, Calculated Grade=Third Class
Test case passed: Marks=(58, 59, 60), Expected Grade=Third Class, Calculated Grade=Third Class
Test case passed: Marks=(48, 49, 50), Expected Grade=Fail, Calculated Grade=Fail
Test case passed: Marks=(0, 1, 2), Expected Grade=Fail, Calculated Grade=Fail
Test case passed: Marks=(101, 102, 103), Expected Grade=Invalid Input, Calculated Grade=Invalid Input
Test case passed: Marks=(100, 99, 101), Expected Grade=Invalid Input, Calculated Grade=Invalid Input
```

PROBLEM4

- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 are considered valid. A success message is shown.

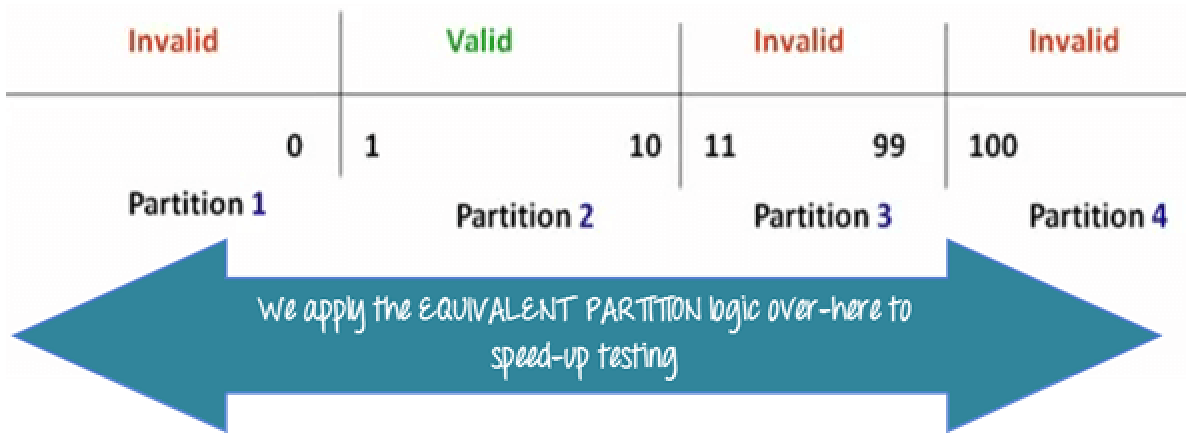
  While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Order PIZZA Submit

Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field (let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

You cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown in the below figure where the system behavior can be considered the same.



PROBLEM 5

Generate Equivalence Partitioning (valid and invalid) Test Cases for a Tax Calculation Problem.

The pay of employee in an organization (“Pay” as an input) having values 12000 to 35000 in the valid range. The program calculates the corresponding tax with following assumptions:

1) Pay up to Rs. 15000 Tax is Zero
2) Pay between 15001 and 25000 Tax is 18 % of Pay
3) Pay above 25000 Tax is 20% of Pay