

# Rajalakshmi Engineering College

Name: Ishwaryaa N  
Email: 240701196@rajalakshmi.edu.in  
Roll no: 240701196  
Phone: 9655727550  
Branch: REC  
Department: I CSE AH  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## NeoColab\_REC\_CS23221\_Python Programming

### REC\_Python\_Week 6\_CY

Attempt : 1  
Total Mark : 40  
Marks Obtained : 40

### Section 1 : Coding

#### 1. Problem Statement

Implement a program that checks whether a set of three input values can form the sides of a valid triangle. The program defines a function `is_valid_triangle` that takes three side lengths as arguments and raises a `ValueError` if any side length is not a positive value. It then checks whether the sum of any two sides is greater than the third side to determine the validity of the triangle.

#### ***Input Format***

The first line of input consists of an integer A, representing side1.

The second line of input consists of an integer B, representing side2.

The third line of input consists of an integer C, representing side3.

### **Output Format**

The output prints either "It's a valid triangle" if the input side lengths form a valid triangle,

or "It's not a valid triangle" if they do not.

If there is a ValueError, it should print "ValueError: <error\_message>".

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 3

4

5

Output: It's a valid triangle

### **Answer**

# You are using Python

```
def is_valid_triangle(a, b, c):
```

```
    if a <= 0 or b <= 0 or c <= 0:
```

```
        raise ValueError("Side lengths must be positive")
```

```
    if a + b > c and a + c > b and b + c > a:
```

```
        return "It's a valid triangle"
```

```
    else:
```

```
        return "It's not a valid triangle"
```

```
try:
```

```
    side1 = int(input())
```

```
    side2 = int(input())
```

```
    side3 = int(input())
```

```
    result = is_valid_triangle(side1, side2, side3)
```

```
    print(result)
```

```
except ValueError as e:
```

```
    print(f"ValueError: {e}")
```

**Status : Correct**

**Marks : 10/10**

## 2. Problem Statement

Write a program to obtain the start time and end time for the stage event show. If the user enters a different format other than specified, an exception occurs and the program is interrupted. To avoid that, handle the exception and prompt the user to enter the right format as specified.

Start time and end time should be in the format 'YYYY-MM-DD HH:MM:SS'. If the input is in the above format, print the start time and end time. If the input does not follow the above format, print "Event time is not in the format "

### ***Input Format***

The first line of input consists of the start time of the event.

The second line of the input consists of the end time of the event.

### ***Output Format***

If the input is in the given format, print the start time and end time.

If the input does not follow the given format, print "Event time is not in the format".

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 2022-01-12 06:10:00

2022-02-12 10:10:12

Output: 2022-01-12 06:10:00

2022-02-12 10:10:12

### ***Answer***

```
# You are using Python
from datetime import datetime
```

```
def validate_datetime_format(date_str):
    try:
        datetime.strptime(date_str, "%Y-%m-%d %H:%M:%S")
```

```
        return True
    except ValueError:
        return False

start_time = input()
end_time = input()

if validate_datetime_format(start_time) and
validate_datetime_format(end_time):
    print(start_time)
    print(end_time)
else:
    print("Event time is not in the format")
```

**Status :** Correct

**Marks :** 10/10

### 3. Problem Statement

In the enchanted realm of Academia, you, the Academic Alchemist, are bestowed with a magical quill and a parchment to weave the grades of aspiring students into a tapestry of academic brilliance.

The mission is to craft a Python program that empowers faculty members to enter student grades for any two subjects, stores these magical grades in a mystical file, and then, with a wave of your virtual wand, calculates the GPA to unveil the true essence of academic achievement.

#### ***Input Format***

The input format is a string representing the student's name, any two subjects, and corresponding grades.

After entering grades, they can type 'done' when prompted for the student's name.

#### ***Output Format***

The output should display the (average of grades) calculated GPA with a precision of two decimal places.

The magical grades will be saved in a mystical file named "magical\_grades.txt".

Refer to the sample output for format specifications.

### **Sample Test Case**

Input: Alice

Math

95

English

88

done

Output: 91.50

### **Answer**

```
# You are using Python
```

```
file_name = "magical_grades.txt"
```

```
with open(file_name, "w") as file:
```

```
    while True:
```

```
        student_name = input()
```

```
        if student_name.lower() == "done":
```

```
            break
```

```
        subject1 = input()
```

```
        grade1 = input()
```

```
        subject2 = input()
```

```
        grade2 = input()
```

```
    try:
```

```
        grade1 = float(grade1)
```

```
        grade2 = float(grade2)
```

```
    if not (0 <= grade1 <= 100 and 0 <= grade2 <= 100):
```

```
        print("Grades must be between 0 and 100.")
```

```
        continue
```

```
    gpa = (grade1 + grade2) / 2
```

```
    file.write(f"{student_name}, {subject1}: {grade1}, {subject2}: {grade2}, GPA:  
{gpa:.2f}\n")
```

```
    print(f"{gpa:.2f}")
```

```
except ValueError:  
    print("Invalid grade input. Please enter numeric values.")
```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

Write a program to read the Register Number and Mobile Number of a student. Create user-defined exception and handle the following:

If the Register Number does not contain exactly 9 characters in the specified format(2 numbers followed by 3 characters followed by 4 numbers) or if the Mobile Number does not contain exactly 10 characters, throw an `IllegalArgumentException`. If the Mobile Number contains any character other than a digit, raise a `NumberFormatException`. If the Register Number contains any character other than digits and alphabets, throw a `NoSuchElementException`. If they are valid, print the message 'valid' or else print an Invalid message.

##### **Input Format**

The first line of the input consists of a string representing the Register number.

The second line of the input consists of a string representing the Mobile number.

##### **Output Format**

The output should display any one of the following messages:

If both numbers are valid, print "Valid".

If an exception is raised, print "Invalid with exception message: ", followed by the specific exception message.

Refer to the sample output for the formatting specifications.

##### **Sample Test Case**

Input: 19ABC1001

9949596920

Output: Valid

**Answer**

# You are using Python

import re

```
class IllegalArgumentException(Exception):  
    pass
```

```
class NumberFormatException(Exception):  
    pass
```

```
class NoSuchElementException(Exception):  
    pass
```

```
def validate_register_number(register_number):  
    if len(register_number) != 9:  
        raise IllegalArgumentException("Register Number should have exactly 9  
characters.")
```

```
    if not re.match(r'^\d{2}[A-Za-z]{3}\d{4}$', register_number):  
        raise IllegalArgumentException("Register Number should have the format: 2  
numbers, 3 characters, and 4 numbers.")
```

```
    if not all(char.isalnum() for char in register_number):  
        raise NoSuchElementException("Register Number should contain only digits  
and alphabets.")
```

```
def validate_mobile_number(mobile_number):  
    if len(mobile_number) != 10:  
        raise IllegalArgumentException("Mobile Number should have exactly 10  
characters.")
```

```
    if not mobile_number.isdigit():  
        raise NumberFormatException("Mobile Number should only contain digits.")
```

```
try:  
    register_number = input().strip()  
    mobile_number = input().strip()  
  
    validate_register_number(register_number)
```

```
validate_mobile_number(mobile_number)
```

```
print("Valid")
```

```
except (IllegalArgumentException, NumberFormatException,  
NoSuchElementException) as e:
```

```
    print(f"Invalid with exception message: {e}")
```

**Status :** Correct

**Marks :** 10/10