

# Rajalakshmi Engineering College

Name: Jhanani shree  
Email: 240701215@rajalakshmi.edu.in  
Roll no: 240701215  
Phone: 7373333511  
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

Alex is creating an account and needs to set up a password. The program prompts Alex to enter their name, mobile number, chosen username, and desired password. Password validation criteria include:

Length between 10 and 20 characters. At least one digit. At least one special character from !@#\$%^&\* set. Display "Valid Password" if criteria are met; otherwise, raise an exception with an appropriate error message.

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

The first line of the input consists of the name as a string.

The second line of the input consists of the mobile number as a string.

The third line of the input consists of the username as a string.

The fourth line of the input consists of the password as a string.

### **Output Format**

If the password is valid (meets all the criteria), it will print "Valid Password"

If the password is weak (fails any one or more criteria), it will print an error message accordingly.

Refer to the sample outputs for the formatting specifications.

### **Sample Test Case**

Input: John  
9874563210

john  
john1#nhøj

Output: Valid Password

### **Answer**

# You are using Python

```
def validate_password(password):
```

```
    if len(password) < 10 or len(password) > 20:
```

```
        return "Should be a minimum of 10 characters and a maximum of 20  
characters"
```

```
    if not any(char.isdigit() for char in password):
```

```
        return "Should contain at least one digit"
```

```
    if not any(char in "!@#$$%^&*" for char in password):
```

```
        return "It should contain at least one special character"
```

```
    return "Valid Password"
```

```
def main():
```

```
    name = input().strip()
```

```
    mobile_number = input().strip()
```

```
    username = input().strip()
```

```
    password = input().strip()
```

```
result = validate_password(password)
print(result)

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

**Status :** Correct

**Marks :** 10/10

## 2. 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(side1, side2, side3):
    if side1 <= 0 or side2 <= 0 or side3 <= 0:
        raise ValueError("Side lengths must be positive")

    # Check the triangle inequality theorem
    if (side1 + side2 > side3) and (side1 + side3 > side2) and (side2 + side3 >
side1):
        return True
    else:
        return False

def main():
    try:
        side1 = int(input().strip())
        side2 = int(input().strip())
        side3 = int(input().strip())

        if is_valid_triangle(side1, side2, side3):
            print("It's a valid triangle")
        else:
            print("It's not a valid triangle")

    except ValueError as e:
        print(f"ValueError: {e}")

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

**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
```

```
def calculate_gpa(grades):  
    return sum(grades) / len(grades)
```

```
def main():  
    grades = []
```

```
while True:
```

```
    student_name = input("").strip()
    if student_name.lower() == 'done':
        break
```

```
    subject1 = input().strip()
    grade1 = float(input(f" ").strip())
```

```
    subject2 = input().strip()
    grade2 = float(input(f"").strip())
```

```
    # Validate grades
```

```
    if not (0 <= grade1 <= 100) or not (0 <= grade2 <= 100):
        print("Grades must be between 0 and 100.")
        continue
```

```
    grades.append((student_name, grade1, grade2))
```

```
    # Calculate GPA
```

```
    gpa = calculate_gpa([grade1, grade2])
    print(f"{gpa:.2f}")
```

```
    # Save to file
```

```
    with open("magical_grades.txt", "a") as file:
        file.write(f"{student_name}, {subject1}, {grade1}, {subject2}, {grade2}\n")
```

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

**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

```
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 (register_number[:2].isdigit() and register_number[2:5].isalpha() and
register_number[5:].isdigit()):
        raise IllegalArgumentException("Register Number should have the format: 2
numbers, 3 characters, and 4 numbers.")
    if not register_number.isalnum():
        raise NoSuchElementException("Register Number contains invalid
characters.")
```

```
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.")
```

```
def main():
    register_number = input().strip()
    mobile_number = input().strip()

    try:
        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}")

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

**Status :** Correct

**Marks :** 10/10