Building Super Fun Applications

Logic Building & Basic Backend Code

CSE110: Programming Language I

Table of Contents

Contest Rules and Regulations

Two programming contests will be held, one before the midterm and the other after the midterm.

Team Building Rules

- There will be six to eight teams. Six to eight horsemen/horsewomen will be selected by the instructor. The selection will be based on the class performance of the students. They will form teams according to their preference. Every student must participate.
- Each team will have 5 members. Each team must have at least one male and one female student.
- Each team must have at least one person with knowledge on the following:

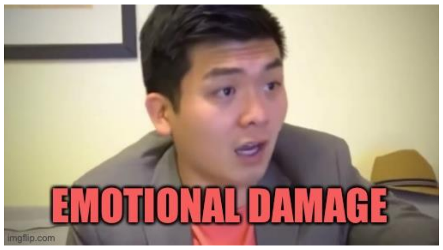
1. Tinder
2. Basic Mathematical Operations
3. Rock-Paper-Scissors
4. Tic-Tac-Toe
5. Ludo
6. Football Penalty Shootout
7. Geometry
8. Scientific Calculator
9. *Dictator* the Movie
10. The Plot of *Mr. Robot* the Series
11. Pokémon Types and Battle Systems
12. *Snakes and Ladders* Game
13. *Attack on Titan* Anime
14. Family Tree and Bootstrap Paradox of the *Dark* Series
15. *Set* Theories
16. Different Types of *Mathematical Series*
17. *Spotify* Playlists Modifier
18. *Minecraft* Mechanics

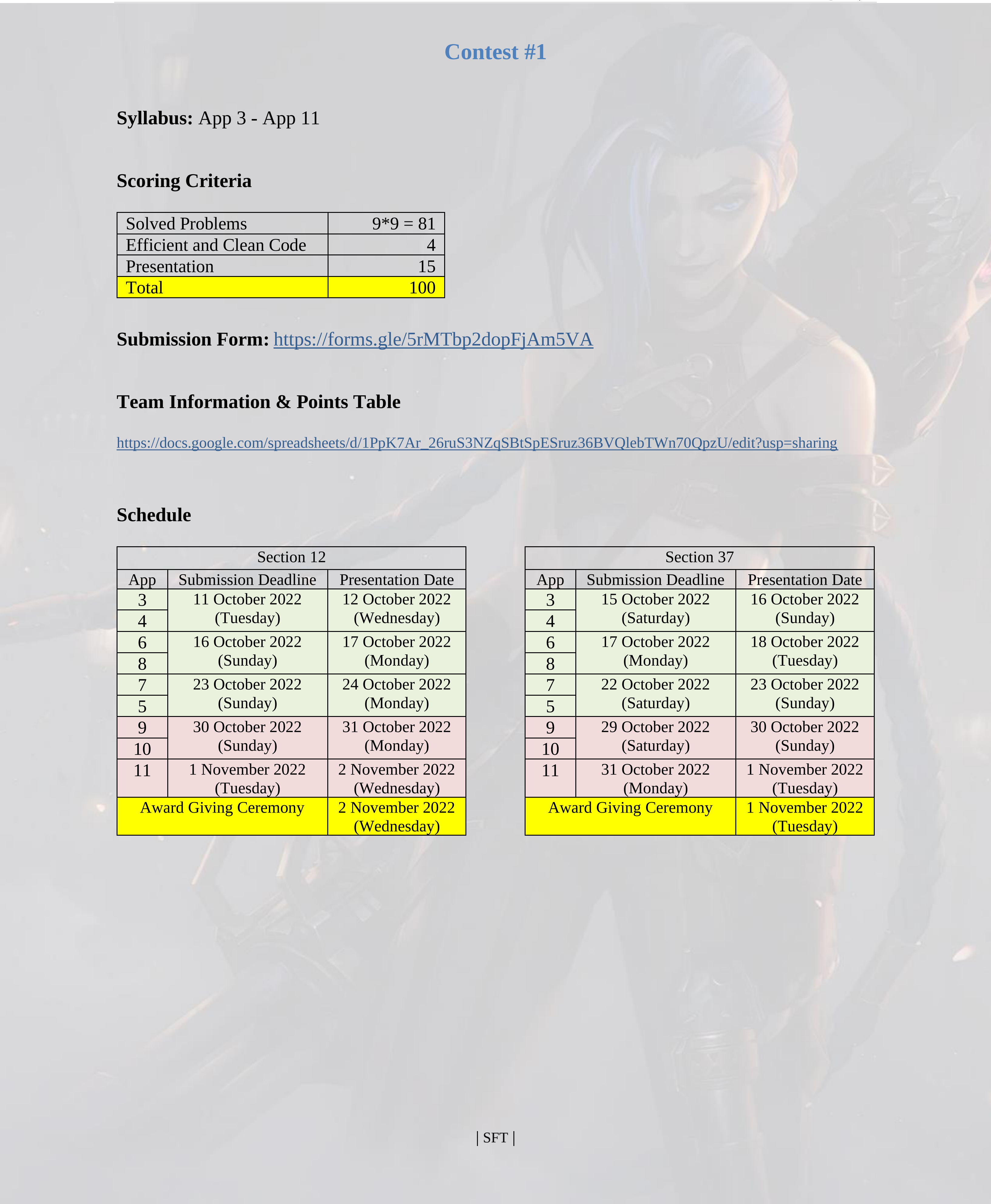
Contest Rules

1. The outputs must match exactly as shown in the question.
2. The code should work for every sample input/output combination.
3. During the presentation, the instructor may ask any member of any team to present any of the contest problem solutions and 15 marks will be allocated for it.
4. One team should not aid any other team. If a team is struggling with something, that team can request help from the instructor.
5. One person from each team will submit the **.ipynb** file in the Google form.
6. All the problems must be submitted by one member of each team before the deadline.
7. Submitting different solutions to a same application may result in some additional score. Therefore, multiple submissions of a same application is granted.

Perks and Benefits

Winning Team of Each Contest	+1.5 Points
2nd, 3rd and 4th Teams of Each Contest	+1 Points
5th – 8th Teams of Each Contest	-1 Points
From each section, top 8 individual performers of each contest will receive exclusive rewards	





Contest #1

Syllabus: App 3 - App 11

Scoring Criteria

Solved Problems	9*9 = 81
Efficient and Clean Code	4
Presentation	15
Total	100

Submission Form: https://forms.gle/5rMTbp2dopFjAm5VA

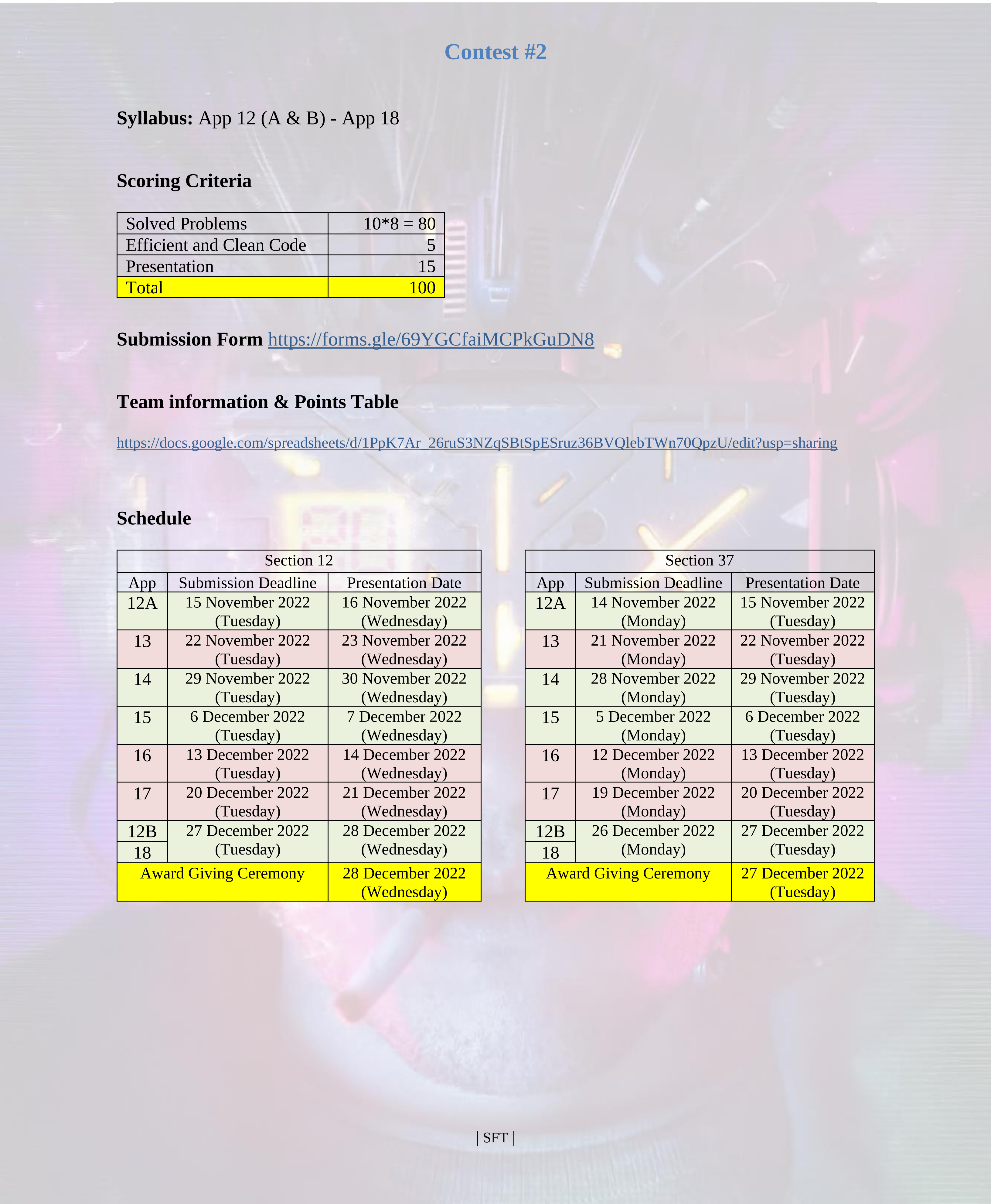
Team Information & Points Table

https://docs.google.com/spreadsheets/d/1PpK7Ar_26ruS3NZqSBtSpESruz36BVQlebTWn70QpzU/edit?usp=sharing

Schedule

Section 12		
App	Submission Deadline	Presentation Date
3	11 October 2022 (Tuesday)	12 October 2022 (Wednesday)
4		
6	16 October 2022 (Sunday)	17 October 2022 (Monday)
8		
7	23 October 2022 (Sunday)	24 October 2022 (Monday)
5		
9	30 October 2022 (Sunday)	31 October 2022 (Monday)
10		
11	1 November 2022 (Tuesday)	2 November 2022 (Wednesday)
Award Giving Ceremony		2 November 2022 (Wednesday)

Section 37		
App	Submission Deadline	Presentation Date
3	15 October 2022 (Saturday)	16 October 2022 (Sunday)
4		
6	17 October 2022 (Monday)	18 October 2022 (Tuesday)
8		
7	22 October 2022 (Saturday)	23 October 2022 (Sunday)
5		
9	29 October 2022 (Saturday)	30 October 2022 (Sunday)
10		
11	31 October 2022 (Monday)	1 November 2022 (Tuesday)
Award Giving Ceremony		1 November 2022 (Tuesday)



Contest #2

Syllabus: App 12 (A & B) - App 18

Scoring Criteria

Solved Problems	10*8 = 80
Efficient and Clean Code	5
Presentation	15
Total	100

Submission Form https://forms.gle/69YGCfaiMCPkGuDN8

Team information & Points Table

https://docs.google.com/spreadsheets/d/1PpK7Ar_26ruS3NZqSBtSpESruz36BVQlebTWn70QpzU/edit?usp=sharing

Schedule

Section 12		
App	Submission Deadline	Presentation Date
12A	15 November 2022 (Tuesday)	16 November 2022 (Wednesday)
13	22 November 2022 (Tuesday)	23 November 2022 (Wednesday)
14	29 November 2022 (Tuesday)	30 November 2022 (Wednesday)
15	6 December 2022 (Tuesday)	7 December 2022 (Wednesday)
16	13 December 2022 (Tuesday)	14 December 2022 (Wednesday)
17	20 December 2022 (Tuesday)	21 December 2022 (Wednesday)
12B	27 December 2022 (Tuesday)	28 December 2022 (Wednesday)
18		
Award Giving Ceremony		28 December 2022 (Wednesday)

Section 37		
App	Submission Deadline	Presentation Date
12A	14 November 2022 (Monday)	15 November 2022 (Tuesday)
13	21 November 2022 (Monday)	22 November 2022 (Tuesday)
14	28 November 2022 (Monday)	29 November 2022 (Tuesday)
15	5 December 2022 (Monday)	6 December 2022 (Tuesday)
16	12 December 2022 (Monday)	13 December 2022 (Tuesday)
17	19 December 2022 (Monday)	20 December 2022 (Tuesday)
12B	26 December 2022 (Monday)	27 December 2022 (Tuesday)
18		
Award Giving Ceremony		27 December 2022 (Tuesday)

App #1A: Tinder Prototype [Difficulty Level: Walk in the Park]

Human beings are such interesting creatures. Even after seeing the red flags, humans tend to go down the wrong way or towards the wrong people. Sometimes, these lead to disasters, sometimes these miraculously work out, although the chances of it working out is very thin. Suppose you are designing a dating app quite similar to Tinder with an additional feature of suggesting whether you should swipe left (ignore) or swipe right (approach).

Now in your app you shall be matching two people up based on the following conditions:

1. If the age gap is less than or equal to 5 years: 1 point
2. If both are from the same state: 1 point
3. If both have the same music taste: 2 points
4. If both have the same hobby: 1 point
5. If both are working people: 2 points. Else if one of them is a working person: 1 point. If both of them are jobless: -1 point
6. If both of them have the same first language: 1 point
7. If both are vegetarian or both are non-vegetarian: 2 points

Now, if the total point is greater than or equals to 5, your app should print "Swipe right". Otherwise, it should print "Swipe left". If any of them is a minor (Age<18), print "Swipe left" without considering anything else.

Sample Input #1:	**Sample Input #2:**
Person 1 Details Age: 25 State: Brooklyn Music Genre: Rock Hobby: Watching Movies Jobholder: Yes First Language: Bangla Food Habit: Vegetarian Person 2 Details Age: 21 State: Arizona Favourite Music Genre: Classical Hobby: Watching Movies Jobholder: Yes First Language: Bangla Food Habit: Non-Vegetarian	Person 1 Details Age: 16 State: Chittagong Music Genre: Bangla Rap Hobby: Doing Tiktok Jobholder: No First Language: Bangla Food Habit: Non-Vegetarian Person 2 Details Age: 24 State: Dholaikhal Favourite Music Genre: Rock Hobby: Doing Tiktok Jobholder: No First Language: Bangla Food Habit: Non-Vegetarian
Output #1: Swipe Right	**Output #2:** Swipe Left

App #1B: Tinder Prototype using Functions [Difficulty Level: Walk in the Park]

Suppose you are designing a dating app quite similar to Tinder with an additional feature of suggesting whether you should swipe left (ignore) or swipe right (approach) **using Functions**.

In this app you shall be matching two people up based on the conditions specified in App #1A.

Your job is to write two functions:

- **get_info()** which takes inputs from the user and **returns** a dictionary containing all these information (Name, age, state, music_genre, hobby, job_status, language, food_habit)
- **compare_and_conclude (dict_1, dict_2)** which takes two dictionaries as parameters, then compares between these two dictionaries to determine and **print** whether either of those two persons should "Swipe Left" or "Swipe Right".

#Driver Code
p1= get_info()
p2 = get_info()
compare_and_conclude (p1, p2)

Sample Input #1: Person 1 Details Age: 25 State: Brooklyn Music Genre: Rock Hobby: Watching Movies Jobholder: Yes First Language: Bangla Food Habit: Vegetarian Person 2 Details Age: 21 State: Arizona Favourite Music Genre: Classical Hobby: Watching Movies Jobholder: Yes First Language: Bangla Food Habit: Non-Vegetarian	**Sample Input #2:** Person 1 Details Age: 16 State: Chittagong Music Genre: Bangla Rap Hobby: Doing Tiktok Jobholder: No First Language: Bangla Food Habit: Non-Vegetarian Person 2 Details Age: 24 State: Dholaikhal Favourite Music Genre: Rock Hobby: Doing Tiktok Jobholder: No First Language: Bangla Food Habit: Non-Vegetarian
Output #1: Swipe Right	**Output #2:** Swipe Left

Programers who write dating apps
I guide others to a treasure that I cannot possess

App #2A: Super Calculator [Difficulty Level: Easy but Lengthy]

Design a super calculator that takes two/one numbers as inputs what operation is going to be performed in between these numbers. Whether you shall be taking two numbers or only one number as input will depend on the operation you need to perform. Your super calculator should be able to perform:

- Addition
- Subtraction
- Multiplication
- Division
- Floor division
- Exponentiation/Root Operation
- Modulus
- Negation (Sign conversion, Single input)
- Compare (Whether first number is equal to, grater or less than the second number)
- State of the number (Whether the number is positive, negative or zero. Single input)

Your program should be able to handle invalid inputs such as:

- Division by 0
- Root of negative numbers
- Any input other than digits
- The first input not being within 0-9

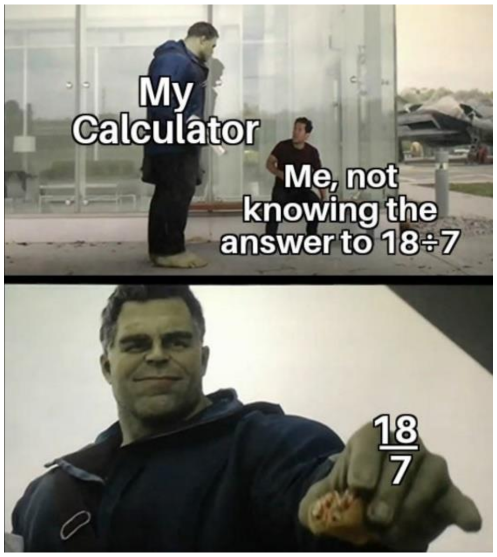
Sample Input #1 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 0 First Number: 5 Second Number: -2 **Output:** Addition: 3	**Sample Input #10** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 5 First Number: -25 Second Number: 0.5 **Output:** Invalid Input: Root of negative number
Sample Input #2 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 1 First Number: 5 Second Number: -2 **Output:** Subtraction: 7	**Sample Input #11** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 6 First Number: 25 Second Number: 0 **Output:** Invalid Input: Division by zero
Sample Input #3 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 2 First Number: 5 Second Number: -2 **Output:** Multiplication: -10.0	**Sample Input #12** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 6 First Number: 25 Second Number: 6 **Output:** Modulus: 1

Sample Input #4 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 3 First Number: 5 Second Number: 0 **Output:** Invalid Input: Division by zero	**Sample Input #13** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 7 Number: 15 **Output:** Negation: -15
Sample Input #5 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 3 First Number: 5 Second Number: -2 **Output:** Division: -2.50	**Sample Input #14** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 7 Number: -15 **Output:** Negation: 15
Sample Input #6 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 4 First Number: 5 Second Number: 2 **Output:** Floor division: 2	**Sample Input #15** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 8 First Number: 25 Second Number: 12 **Output:** First Number is greater than second number
Sample Input #7 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 4 First Number: 5 Second Number: 0 **Output:** Invalid Input: Division by zero	**Sample Input #16** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 8 First Number: 25 Second Number: 25 **Output:** Both numbers are equal
Sample Input #8 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 5 First Number: 5 Second Number: 2 **Output:** Exponentiation/Root: 25.0	**Sample Input #17** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 9 Number: 25 **Output:** The number is positive
Sample Input #9 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 5 First Number: 25 Second Number: 0.5 **Output:** Exponentiation/Root: 5.0	**Sample Input #18** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 10 **Output:** Invalid operation choice

App #2B: Super Calculator using Functions [Difficulty Level: Easy but Lengthy]

Design a super calculator that takes two/one numbers as inputs what operation is going to be performed in between these numbers. Whether you shall be taking two numbers or only one number as input will depend on the operation you need to perform. Your super calculator should have the following functions:

- addition(n1,n2)
- subtraction(n1,n2)
- multiplication(n1,n2)
- division(n1,n2)
- floor division(n1,n2)
- exponentiation(n1,n2)
- modulus(n1,n2)
- negation(n1)
- compare(n1,n2)
- state(n1)



Your program should be able to handle invalid inputs such as:

- Division by 0
- Root of negative numbers
- Any input other than digits
- The first input not being within 0-9

Depending on the choice of operation (1-9), a specific function will be called.

Sample Input #1 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 0 First Number: 5 Second Number: -2 **Output:** Addition: 3	**Sample Input #10** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 5 First Number: -25 Second Number: 0.5 **Output:** Invalid Input: Root of negative number
Sample Input #2 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 1 First Number: 5 Second Number: -2 **Output:** Subtraction: 7	**Sample Input #11** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 6 First Number: 25 Second Number: 0 **Output:** Invalid Input: Division by zero
Sample Input #3 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 2 First Number: 5 Second Number: -2 **Output:** Multiplication: -10.0	**Sample Input #12** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 6 First Number: 25 Second Number: 6 **Output:** Modulus: 1

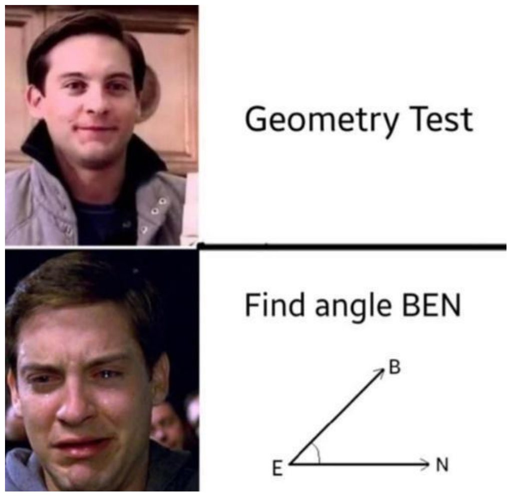
Sample Input #4 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 3 First Number: 5 Second Number: 0 **Output:** Invalid Input: Division by zero	**Sample Input #13** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 7 Number: 15 **Output:** Negation: -15
Sample Input #5 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 3 First Number: 5 Second Number: -2 **Output:** Division: -2.50	**Sample Input #14** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 7 Number: -15 **Output:** Negation: 15
Sample Input #6 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 4 First Number: 5 Second Number: 2 **Output:** Floor division: 2	**Sample Input #15** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 8 First Number: 25 Second Number: 12 **Output:** First Number is greater than second number
Sample Input #7 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 4 First Number: 5 Second Number: 0 **Output:** Invalid Input: Division by zero	**Sample Input #16** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 8 First Number: 25 Second Number: 25 **Output:** Both numbers are equal
Sample Input #8 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 5 First Number: 5 Second Number: 2 **Output:** Exponentiation/Root: 25.0	**Sample Input #17** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 9 Number: 25 **Output:** The number is positive
Sample Input #9 Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 5 First Number: 25 Second Number: 0.5 **Output:** Exponentiation/Root: 5.0	**Sample Input #18** Which Operation you want to perform? (0=Addition, 1=Subtraction, 2=Multiplication, 3=Division, 4= Floor division, 5= Exponentiation/ Root Operation, 6= Modulus, 7= Negation, 8= Compare, 9= State of the number): 10 **Output:** Invalid operation choice

App #3: Triangle/Quadrilateral Validator using Sides [Difficulty Level: Walk in the Park]

Imagine you are trying to draw a quadrilateral or a triangle. So, you are taking the lengths of each of the sides of the shape. Now, what shape you need to draw depends on the first input (3/4). After deciding what your shape will be, your job is to verify whether the shape can be drawn using the lengths given by the user.

a. If no of sides: 3, a triangle is needed to be drawn. To draw a triangle, the sum of the lengths of any two sides must be greater than the third.
b. If no of sides: 4, a quadrilateral is needed to be drawn. To draw a quadrilateral, the sum of the lengths of any three sides must be greater than the fourth.
c. For any other input for No of angles, print "Invalid input"

Sample Input 1: No of sides: 1 **Output:** Invalid input	**Sample Input 2:** No of sides: -2 **Output:** Invalid input	**Sample Input 3:** No of sides: 3 Length of sides: 2 3 1 **Output:** Invalid input
Sample Input 4: No of sides: 3 Length of sides: 2 3 2 **Output:** A triangle can be drawn	**Sample Input 5:** No of sides: 4 Length of sides: 2 3 1 4 **Output:** A quadrilateral can be drawn	**Sample Input 6:** No of sides: 4 Length of sides: 2 3 1 7 **Output:** Invalid input



When the geometry teacher starts talking about cubes and you remember how you once solved one color of a Rubik's cube
You know, I'm something of a scientist myself.

App #4: Triangle/ Quadrilateral Detector using Angles [Difficulty Level: Walk in the Park]

Imagine you are trying to draw a quadrilateral or a triangle. So, you are taking the angles of the shape. Now, what shape you need to draw depends on the first input (3 or 4). After deciding what your shape will be, your job is to verify whether the shape can be drawn using the angles given by the user. You also have to specify what kind of triangle (equilateral/isosceles/scalene and acute/right/obtuse) or quadrilateral (rectangle/parallelogram/square/rhombus/regular quadrilateral) the shape is.

a. If no of angles: 3, a triangle is needed to be drawn. To draw a triangle, the sum of the angles must be equal to 180 degrees.
b. In the triangle if all the angles are equal it is an equilateral triangle, if two of the angles are equal it is an isosceles triangle, and if none of the angles are equal, it is a scalene triangle. If one of the angles is equal to 90 degrees, it is a right triangle. If one of the angles is greater than 90 degrees, it is an obtuse triangle. If none of the angles are greater than 90 degrees, it is an acute triangle.
c. If no of angles: 4, a quadrilateral is needed to be drawn. To draw a quadrilateral, the sum of the angles must be equal to 360 degrees.
d. In the quadrilateral, if all the angles are equal (90 degrees) it is either a square or a parallelogram, if the first-third and second-fourth angles are equal, it is a rhombus or a rectangle. Otherwise, it is a regular quadrilateral.
e. Angles cannot be zero or negative.
f. For any other input for No of angles, print "Invalid input".

Sample Input #1: No of angles: 1 **Output:** Invalid input	**Sample Input 2:** No of sides: 4 Angles: 80 100 80 100 **Output:** A rhombus or a rectangle can be drawn	**Sample Input 3:** No of angles: 3 Angles: 60 90 50 **Output:** Invalid input
Sample Input 4: No of sides: 3 Angles: 60 90 30 **Output:** A scalene right triangle can be drawn	**Sample Input 5:** No of sides: 4 Angles: 60 80 70 150 **Output:** A regular quadrilateral can be drawn	**Sample Input 6:** No of sides: 4 Angles: 60 80 70 130 **Output:** Invalid input

App #5: Password Validator [Difficulty Level: Walk in the Park]

Imagine you developed a split personality called Mr. Robot who is an occasional hacker. To infiltrate a large company and gain their trust, you must create a simple password validator program for them that checks if a password is valid or not.

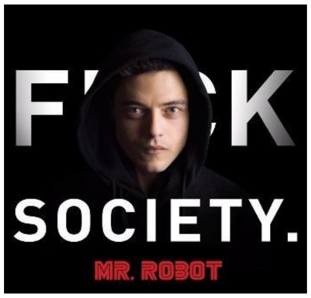
The password must:

a. Be of at least length 8.
b. Contain at least one digit (0-9)
c. Contain at least **one** upper case letter (A-Z)
d. Contain at least **two** smaller case letters (a-z).

If one of the criteria is missing,

specify which one is missing.

Sample Input #1 Password: BabaSifatErTobarok21 **Output:** Password Valid!	**Sample Input #2** Password: OmShantiOm **Output:** Password Invalid! Digits missing.	**Sample Input #3** Password: sigmamale **Output:** Password Invalid! Digits missing. Upper-case missing.
Sample Input #4 Password: GGWPEZ **Output:** Password Invalid! Insufficient length. Digits missing. Insufficient lower-case.	**Sample Input #5** Password: (ノಠ益ಠ) (ง'̀-'́)ง (͡° ͜ʖ ͡°) ¯_(ツ)_/¯ (=^ェ^=) (✿◠‿◠) **Output:** Password Invalid! Digits missing. Upper-case missing. Insufficient lower-case.	**Sample Input #6** Password: Dipjol#2169621 **Output:** Password Valid!



App #6: Aladeen Date Validator [Difficulty Level: Aladeen]

Imagine you are the chief scheduling officer of Admiral General Aladeen and are in charge of handling all his schedules.
You have to build a date validator which has to be accurate and impeccable, otherwise you will be executed and lose your head.

Take one date from the user and find out whether that date is valid (Aladeen) or not.

a. You have to consider leaps years too.
b. The date format (Day/Month/Year) must be followed strictly. For example, 08/02/2022.
c. Invalid inputs must be handled properly. For example, 31/02/2018, 12/13/2012, -12/5/2014 and 5/0/2000 etc. are invalid dates.

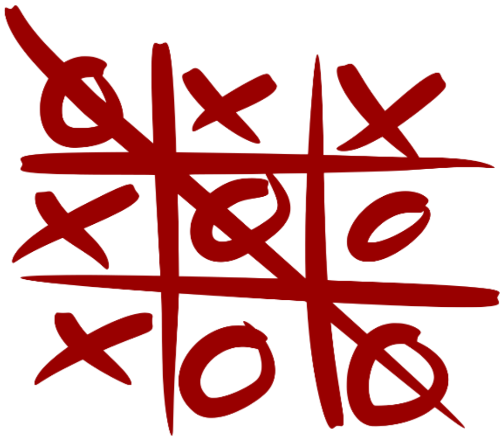
Sample Input #1: Date: 12/05/1996 **Output:** Aladeen Date	**Sample Input #2:** Date: 12/13/1996 **Output:** Invalid Date	**Sample Input #3:** Date: -12/12/1996 **Output:** Invalid Date
Sample Input #4: Date: 29/02/2003 **Output:** Invalid Date	**Sample Input #5:** Date: 29/02/2004 **Output:** Aladeen Date	**Sample Input #6:** Date: 02/02/2004 **Output:** Aladeen Date

App #7: Tic-Tac-Toe [Difficulty Level: Walk in the Park]

Suppose, one day you ask your parents for a new phone/laptop and like traditional brown parents they start roasting you. Now, in order to protect your dignity, you have to show them what you learned studying CSE. So, you quickly decide to design a tic-tac-toe game to reestablish your dignity and ask for a MacBook/iPhone.

The rules of the game are:

- The game is played on a grid that's 3 squares by 3 squares.
- The first player to get 3 of their marks in a row (up, down, across, or diagonally) is the winner.
- When all 9 squares are full, but no winner is decided, the game ends in a tie.
- The valid input range is between 1 to 9. The same input cannot be given more than once.

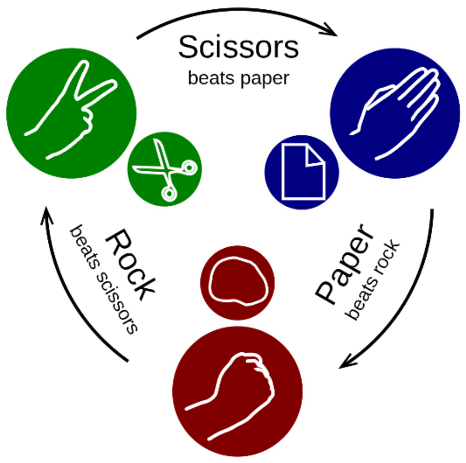


Sample Input #1	**Sample Input #2**	**Sample Input #3**
Player 1: 1 Player 2: 5 Player 1: 7 Player 2: 1 Cannot enter a number that is already taken. Enter again. Player 2: 6 Player 1: 4 **Output:** Player 1 wins!	Player 1: 2 Player 2: 5 Player 1: 0 Invalid Input. Enter again. Player 1: 3 Player 2: 1 Player 1: 7 Player 2: 9 **Output:** Player 2 wins!	Player 1: 2 Player 2: 7 Player 1: 3 Player 2: 1 Player 1: 4 Player 2: 5 Player 1: 9 Player 2: 6 Player 1: 8 **Output:** It's a tie!

We know, even after seeing that your program works, your parents still aren't convinced…
Sad Unemployment Noises

App #8: Rock-Paper-Scissor [Difficulty Level: Walk in the Park]

Imagine you are building a virtual rock-paper-scissor game playable among 3 players. Now write a python program that will take "rock", "paper" or "scissor" as inputs and decide the winner using the following conditions.



Paper: *not invented yet*
Scissors: *not invented yet*
Cavemen when deciding on something:

A winner is decided if:

a) One player picks rock and the other two pick scissor
Or b) One player picks scissor and the other two pick paper
Or c) One player picks paper and the other two picks rock.

Remember that there **can only be one winner**. Until a winner is decided, the game will continue.

Sample Input #1:	**Sample Input #2:**
Player 1: rock Player 2: paper Player 3: rock **Output:** Player 2 wins Explanation: Here condition (c) was satisfied and hence player 2 became the winner.	Player 1: rock Player 2: paper Player 3: paper Player 1: scissor Player 2: scissor Player 3: paper Player 1: paper Player 2: paper Player 3: paper Player 1: paper Player 2: paper Player 3: scissor **Output:** Player 3 wins Explanation: Here the game continued until having paper, paper and scissor inputs, the condition (b) was satisfied and player 3 became the winner.

App #9: Ludo [Difficulty Level: Nose to the Grindstone]

Imagine you are designing a minimalistic version of the game ludo with the following conditions:

a. The player who crosses throughout the board of exactly 25 boxes wins.
b. Each of the players get to roll the dice and can get from 1-6. If a player scores 6, he/she gets another immediate turn. If a player scores 6 three times in a row, all three turns get cancelled and the player stays stationary.
c. If a player crosses 24 boxes, he/she must get exactly 1 after rolling the dice in order to win. Otherwise, he/she will stay in the same place.
d. The one who gets to the 25th box first will win.
e. Until a winner is decided, the dice will be rolled between the players and the game will continue.
f. If one player gets to a box which is already occupied by another player, the one who occupied the box gets eaten and has to start from box no 1 again.

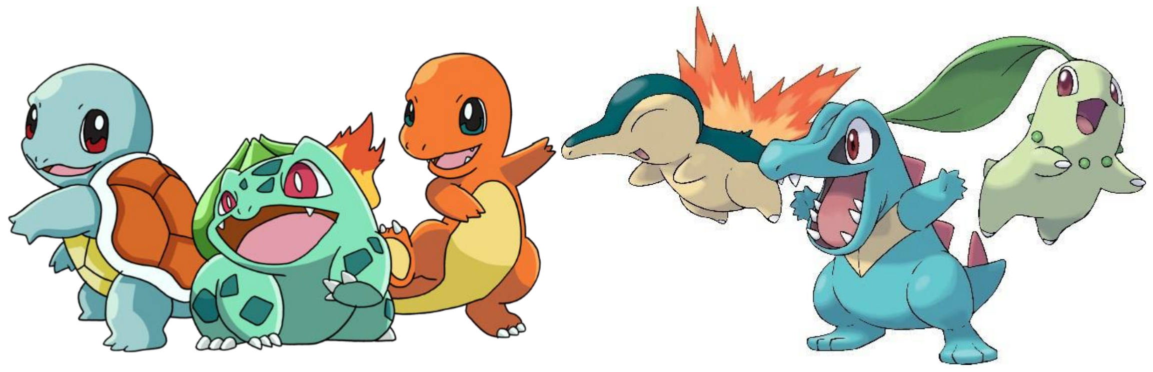
Sample Input #1:	**Sample Input #2:**	**Sample Input #3:**	**Sample Input #4:**
1. Player 1: 5	1. Player 1: 5	1. Player 1: 2	1. Player 1: 4
2. Player 2: 4	2. Player 2: 3	2. Player 2: 3	2. Player 2: 5
3. Player 1: 5	3. Player 1: 5	3. Player 1: 6	3. Player 1: 4
4. Player 2: 5	4. Player 2: 2	4. Player 1: 6	4. Player 2: 5
5. Player 1: 6	5. Player 1: 5	5. Player 1: 6	5. Player 1: 4
6. Player 1: 1	6. Player 2: 4	6. Player 2: 3	6. Player 2: 5
7. Player 2: 4	7. Player 1: 5	7. Player 1: 5	7. Player 1: 4
8. Player 1: 4	8. Player 2: 5	8. Player 2: 3	8. Player 2: 5
9. Player 2: 6	9. Player 1: 3	9. Player 1: 5	9. Player 1: 4
10. Player 2: 6	10. Player 2: 4	10. Player 2: 6	10. Player 2: 5
	11. Player 1: 5	11. Player 2: 6	11. Player 1: 4
	12. Player 2: 4	12. Player 2: 4	12. Player 2: 5
	13. Player 1: 1		13. Player 1: 1
	14. Player 2: 6		
	15. Player 1: 3		
	16. Player 2: 4		
	17. Player 1: 1		
Output: Player 2 wins	**Output:** Player 1 wins	**Output:** Player 2 wins	**Output:** Player 1 wins
		[Hint: Player 1 hits 3 6s in a row and therefore his turn gets cancelled, and he stays stationary]	[Hint: Player 2 gets eaten in turn no 9 and starts from box 1 again]

How I console myself every time I lose in ludo.
LIT
Joh haarta hai,
wahi jeetne ka matlab jaanta hai

App #10: Pokémon Battle Simulator [Difficulty Level: Nose to the Grindstone]

Imagine on one morning you woke up in the world of Pokémon as Ash Ketchum. Professor Oak then asked you to choose one from the following 6 starter Pokémon. After you chose your Pokémon, your arch nemesis Gary Oak chose one from the remaining Pokémon.

Now Gary being the bully that he is, challenges you to a battle. However, you are a CSE student in real life and therefore, decided to design a Pokémon battle simulator to determine your winning chances.



- Charmander and Cyndaquil are fire type, Squirtle and Totodile are water type, and Bulbasaur and Chikorita are grass type.
- Fire types are strong against grass type and do double damage, water types are strong against fire type and do double damage, and grass types are strong against water types and do double damage. In these cases, type affinity = 2
- Fire types are weak against water type and do half damage, water types are weak against grass types and do half damage, grass types are weak against fire types and do half damage. In these cases, type affinity = 0.5
- For same types (fire-fire/water-water/grass-grass), type affinity = 1
- Assume all 6 Pokémon have 50 hp and 20 attack power.
- The battle goes on until one of the Pokémon's hp goes down to zero (or negative). Equation used to calculate remaining hp after each turn:

  Remaining hp = previous hp – attack power * type affinity

- When the battle is over, declare who is the winner.

Your task is to create a Pokémon battle simulator that determines who wins between you and your rival.

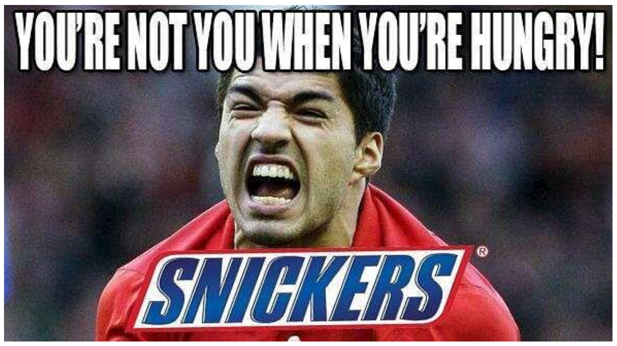
Sample Input #1	**Sample Input #2**	**Sample Input #3**
You Choose: Charmander Gary Chooses: Totodile **Output:** === Turn 1 === Charmander has 10 hp left Totodile has 40 hp left === Turn 2 === Charmander has 0 hp left Totodile has 30 hp left Totodile and Gary wins!	You Choose: Squirtle Gary Chooses: Cyndaquil **Output:** === Turn 1 === Squirtle has 40 hp left Cyndaquil has 10 hp left === Turn 2 === Squirtle has 30 hp left Cyndaquil has 0 hp left Squirtle and Ash wins!	You Choose: Chikorita Gary Chooses: Bulbasaur **Output:** === Turn 1 === Chikorita has 30 hp left Bulbasaur has 30 hp left === Turn 2 === Chikorita has 10 hp left Bulbasaur has 10 hp left === Turn 3 === Chikorita has 0 hp left Bulbasaur has 0 hp left It's a tie!

App #11: Penalty Shootout [Difficulty Level: Nose to the Grindstone]

Imagine two teams are playing football and the scores are tied after 120 minutes of play. Now the winners will be decided through a penalty shootout. Now write a python program that takes in "goal" or "miss" as inputs and prints the winner and the score using the following conditions.

1. Each team will get at most 5 chances to shoot. The team scoring more goals wins.
2. If the goals are tied again, a draw is declared.
3. A Team may be able to secure victory even before all 5 chances are used. For example, if team 1 scores the first 3 goals and team 2 misses the first 3 goals, the game ends instantly and Team 1 wins.

Sample Input #1:	**Sample Input #2:**	**Sample Input #3:**
Team 1: goal	Team 1: goal	Team 1: miss
Team 2: goal	Team 2: miss	Team 2: goal
Team 1: goal	Team 1: goal	Team 1: goal
Team 2: goal	Team 2: goal	Team 2: goal
Team 1: miss	Team 1: miss	Team 1: miss
Team 2: goal	Team 2: goal	Team 2: goal
Team 1: goal	Team 1: miss	Team 1: miss
Team 2: miss	Team 2: goal	
Team 1: goal	Team 1: goal	**Output:**
Team 2: miss	Team 2: miss	Team 2 wins by 3-1
Output:	**Output:**	Explanation:
Team 1 wins by 4-3	Draw by 3-3	Since team 1 only had one chance left but team 2 is already leading by 3-1. Therefore, no further input is needed and Team 2 is declared the winner.
Explanation:	Explanation:	
After 5 turns, team 1 scored 4 goals and team 2 scored 3. Therefore, team 1 are the winners.	After 5 turns, both teams scored an equal number of goals (3 goals each). Therefore, a draw is declared.	



App #12A: Set Operations [Difficulty Level: As Easy as Maths]

Design a set operation application that asks the user which set operation the user wants to perform. Then take one/two **string** inputs from the user and convert these into list(s). After performing the appropriate set operation, print the resultant list.

- .sort() function is allowed
- Take conceptual help from: https://www.cuemath.com/algebra/operations-on-sets/

Sample Input #1: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 1 A: [10, 20, 30] B: [20, 30, 40, 50] **Output:** A U B: [10, 20, 30, 40, 50]	**Sample Input #2:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 2 A: [10, 20, 30] B: [20, 30, 40, 50] **Output:** A ∩ B : [20, 30]
Sample Input #3: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 1 A: [ ] B: [ ] **Output:** A U B: [ ]	**Sample Input #4:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 2 A: [10, 20, 30] B: [ ] **Output:** A ∩ B: [ ]
Sample Input #5: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 3 A: [10, 20, 30] Universal Set, U: [10, 20, 30, 40, 50] **Output:** A' : [40, 50]	**Sample Input #6:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 3 A: [10, 20, 30, 70] Universal Set, U: [10, 20, 30, 40, 50] **Output:** Error: First Set Containing Elements that are not a Part of the Universal Set.
Sample Input #7: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 4 A: [10, 20, 30, 70, 80] B: [20, 30, 40, 50] **Output:** A – B: [70, 80]	**Sample Input #8:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 4 A: [10, 20, 30, 70, 80] B: [40, 50] **Output:** A – B: [10, 20, 30, 70, 80]
Sample Input #7: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 5 A: [10, 20, 30] B: [30, 40] **Output:** A X B: [[10, 30], [10, 40], [20, 30], [20,40], [30, 30], [30, 40]]	**Sample Input #8:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 5 A: [10, 20, 30] B: [] **Output:** A X B: [[10, None], [20, None], [30, None]]

App #12B: Set Operations using Functions [Difficulty Level: As Easy as Maths]

Design a set operation application that asks the user which set operation the user wants to perform. Then take one/two **string** inputs from the user and convert these into list(s). After performing the appropriate set operation, print the resultant list.

- .sort() function is allowed

You have to implement the following functions:

1. union(A,B)
2. intersection(A,B)
3. complement(A,universal_set)
4. difference(A,B)
5. cross_product(A,B)

Depending on the choice of operation (1-5), a specific function will be called.

Sample Input #1: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 1 A: [10, 20, 30] B: [20, 30, 40, 50] **Output:** A U B: [10, 20, 30, 40, 50]	**Sample Input #2:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 2 A: [10, 20, 30] B: [20, 30, 40, 50] **Output:** A ∩ B : [20, 30]
Sample Input #3: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 1 A: [ ] B: [ ] **Output:** A U B: [ ]	**Sample Input #4:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 2 A: [10, 20, 30] B: [ ] **Output:** A ∩ B: [ ]
Sample Input #5: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 3 A: [10, 20, 30] Universal Set, U: [10, 20, 30, 40, 50] **Output:** A' : [40, 50]	**Sample Input #6:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 3 A: [10, 20, 30, 70] Universal Set, U: [10, 20, 30, 40, 50] **Output:** Error: First Set Containing Elements that are not a Part of the Universal Set.
Sample Input #7: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 4 A: [10, 20, 30, 70, 80] B: [20, 30, 40, 50] **Output:** A – B: [70, 80]	**Sample Input #8:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 4 A: [10, 20, 30, 70, 80] B: [40, 50] **Output:** A – B: [10, 20, 30, 70, 80]
Sample Input #7: Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 5 A: [10, 20, 30] B: [30, 40] **Output:** A X B: [[10, 30], [10, 40], [20, 30], [20,40], [30, 30], [30, 40]]	**Sample Input #8:** Set Operation (1=Union, 2=Intersection, 3=Complement, 4=Difference, 5=Cross Product): 5 A: [10, 20, 30] B: [] **Output:** A X B: [[10, None], [20, None], [30, None]]

App #13: Snakes and Ladders [Difficulty Level: Walk in the Park]

Imagine you are building the snakes and ladders game by yourself for two players. The following are the rules of the game:

- Both players will start from box 0 and only get to box 1 after scoring a 1. Otherwise, they will stay in box 0.
- The one reaching box 30 first is the winner
- Any player who reaches the head of a snake will be eaten and sent to the tail of that snake.
- Any player who reaches the bottom of the ladder gets elevated to the top of that ladder.
- If a player gets to box 29, he/she must score 1 to win.

The following two dictionaries are the snakes and ladders positions respectfully. In the first dictionary, the keys are the heads and the values are the tails of the snakes. In the second dictionary, the keys are the bottom of the ladders and the values are the top of the ladders. You can copy these two dictionaries directly onto your code.

Snakes= {15: 5, 19: 12, 25: 16, 27: 21}

Ladders= {9: 14, 13: 18, 17: 26}

You have to find the winner and print the position of the two players after each turn. You also have to show whenever a player gets a ladder or gets eaten by a snake. You have to continuously take P1 and P2 inputs until a winner is decided.

Sample Input/Output 1:	**Sample Input/Output 2:**
P1: 1 P2: 4 P1 in box 1 and P2 in box 0 P1: 5 P2: 1 P1 in box 6 and P2 in box 1 P1: 3 P1 got a ladder! P2: 4 P1 in box 14 and P2 in box 5 P1: 5 P1 got eaten by a snake! P2: 6 P1 in box 12 and P2 in box 11 P1: 5 P1 got a ladder! P2: 4 P2 got eaten by a snake! P1 in box 26 and P2 in box 5 P1: 3 P2: 6 P1 in box 29 and P2 in box 11 P1: 3 P2: 6 P2 got a ladder! P1 in box 29 and P2 in box 26 P1: 1 P1 Wins!	P1: 4 P2: 6 P1 in box 0 and P2 in box 0 P1: 1 P2: 1 P1 in box 1 and P2 in box 1 P1: 4 P2: 5 P1 in box 5 and P2 in box 6 P1: 4 P1 got a ladder! P2: 6 P1 in box 14 and P2 in box 12 P1: 5 P1 got eaten by a snake! P2: 5 P2 got a ladder! P1 in box 12 and P2 in box 26 P1: 5 P1 got a ladder! P2: 6 P1 in box 26 and P2 in box 26 P1: 3 P2: 4 P2 Wins!

App #14: Mathematical Series Builder [Difficulty Level: As Easy as Maths]

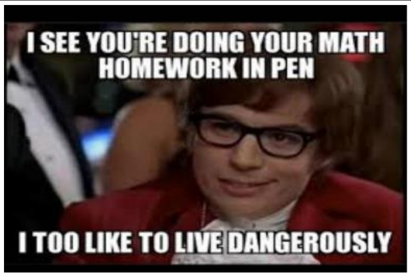
Imagine you got frustrated with CSE and now looking to shift into MNS. Some of your friends encouraged you to do a double major in both CSE and MNS. You, being a stubborn child, decided to have a go at it. Now, you want to build a program that blends both CSE and MNS altogether.

Design an application that asks which series you want to build and provides the sum of the series depending on the **extent** (n) of that particular series. Value of n must be grater than 0. Assume all the inputs are integers.

Your application should be able to build the following types of series:

1. Common Difference Series [a, (a+d), (a+2d), ... , (a+(n-1) d)]
2. Power Series [1^x, 2^x, 3^x, …., n^x]
3. Fibonacci Series [0, 1, 1, 2, 3, 5, 6….]
4. Reversal Series [1, -2, 3, -4, …, (+ or -) n]

Therefore, the valid choices for selecting series range in between 1 to 4.

Sample Input #1: Which Series to Build? (1= Common Difference, 2= Power, 3= Fibonacci, 4= Reversal): 1 Value of n: 5 Value of a: 1 Value of d: 3 **Output:** Series: 1, 4, 7, 10, 13 Sum of the Series: 35	**Sample Input 2:** Which Series to Build? (1= Common Difference, 2= Power, 3= Fibonacci, 4= Reversal): 2 Value of n: 5 Value of x: 2 **Output:** Series: 1^2, 2^2, 3^2, 4^2, 5^2 Sum of the Series: 55	**Sample Input 3:** Which Series to Build? (1= Common Difference, 2= Power, 3= Fibonacci, 4= Reversal): 2 Value of n: 4 Value of x: 3 **Output:** Series: 1^3, 2^3, 3^3, 4^3 Sum of the Series: 100
Sample Input 4: Which Series to Build? (1= Common Difference, 2= Power, 3= Fibonacci, 4= Reversal): 3 Value of n: 7 **Output:** Series: 0, 1, 1, 2, 3, 5, 8 Sum of the Series: 20	**Sample Input 5:** Which Series to Build? (1= Common Difference, 2= Power, 3= Fibonacci, 4= Reversal): 4 Value of n: 6 **Output:** Series: 1, -2, 3, -4, 5, -6 Sum of the Series: -3	**Sample Input 5:** Which Series to Build? (1= Common Difference, 2= Power, 3= Fibonacci, 4= Reversal): 4 Value of n: 7 **Output:** Series: 1, -2, 3, -4, 5, -6, 7 Sum of the Series: 4
Sample Input 7: Which Series to Build? (1= Common Difference, 2= Power, 3= Fibonacci, 4= Reversal): 2 Value of n: -1 **Output:** Invalid Input	**Sample Input 8:** Which Series to Build? (1= Common Difference, 2= Power, 3= Fibonacci, 4= Reversal): 5 **Output:** Invalid Choice	 I SEE YOU'RE DOING YOUR MATH HOMEWORK IN PEN I TOO LIKE TO LIVE DANGEROUSLY

App #15: Dark Bootstrap Paradox [Difficulty Level: Finding the Origin of the Paradox]

Bootstrap Paradox

Bootstrap Paradox occurs when an object, information or a person travels back in time and gets trapped in an infinite cause and effect loop and no longer has a point of origin.

Imagine you have been recruited by Future Adam and Claudia of the famous series *Dark*. Now, you have been tasked to list and investigate all the paradoxes. You have to design a program that should be able to detect the number of paradoxes present in the events and print each of the paradox. Copy the events string directly into your code.

events= "Michael gets lost in the woods. Ulrich goes to find Michael. Both travel to the past but not in the same timeline. Ulrich gets put into the prison. Michael is Jonas's father. Michael committed suicide by hanging himself. Jonas travels back in time to save his father. He finds his dad and tells him about his death. His dad now understood what he had to do. Michael committed suicide by hanging himself. The entire family of Tannahaus was killed in a car accident. This drove him crazy. Tannahaus built a time machine. He also wrote a book on time travel. Claudia finds the book on time travel. Future Claudia gave the present Claudia and time machine. The present Claudia travels back in time and gave the time travelling book to Tannahaus. Tannahaus took ideas from the book he was supposed to write in future. Tannahaus built a time machine. A cult like faction keeps on abducting kids from Winden and experiments on them. None of the abducted kids are ever found again. Charlotte gave birth to Elizabeth. The apocalypse takes place and Charlotte dies. However, Elizabeth somehow survives and gets sent to the past. Elizabeth meets a guy and they have a daughter named Charlotte. After growing up Charlotte gets married. Charlotte gave birth to Elizabeth. Thus the cycle continues. After the apocalypse, only a handful of people survived. Elizabeth was one of them. She became the leader of a savage group. Jonas and Martha love each other. Martha is killed by Adam. Adam is from the future. Jonas travels through time to stop Adam, only to find out that he is the one who eventually becomes Adam. Adam is the only one capable of breaking the endless cycle. For Jonas to become Adam, it is necessary for Martha to die. Without her dying, Jonas would not become Adam and the cycle would never be broken. So, Adam goes back in time. Martha is killed by Adam. After Martha's death Jonas travels into another world."

Output:

```
4 Paradoxes

Michael committed suicide by hanging himself. Jonas travels back in time to save his father. He finds his dad and tells him about his death. His dad now understood what he had to do. Michael committed suicide by hanging himself.

Tannahaus built a time machine. He also wrote a book on time travel. Claudia finds the book on time travel. Future Claudia gave the present Claudia and time machine. The present Claudia travels back in time and gave the time travelling book to Tannahaus. Tannahaus took ideas from the book he was supposed to write in future. Tannahaus built a time machine.

Charlotte gave birth to Elizabeth. The apocalypse takes place and Charlotte dies. However, Elizabeth somehow survives and gets sent to the past. Elizabeth meets a guy and they have a daughter named Charlotte. After growing up Charlotte gets married. Charlotte gave birth to Elizabeth.

Martha is killed by Adam. Adam is from the future. Jonas travels through time to stop Adam, only to find out that he is the one who eventually becomes Adam. Adam is the only one capable of breaking the endless cycle. For Jonas to become Adam, it is necessary for Martha to die. Without her dying, Jonas would not become Adam and the cycle would never be broken. So, Adam goes back in time. Martha is killed by Adam.
```

App #16: Attack on Titan [Difficulty Level: Fighting Levi Ackerman]

Throughout history 9 legendary titans have existed. These titans are: Founding, Attack, Armored, Colossal, Female, Cart, Jaw, Beast and Warhammer. A titan shifter gets access to another titan shifter's power through devouring them.

Within the last 200 years, there have been many battles among the titans and the titan powers have switched hands many times. Your job is to find out the latest wielders of the titans.

You have two dictionaries below which you must copy directly onto your code. The first dictionary shows who eats whom and the second dictionary shows the original wielders of each of the titans.

***Do not worry about being spoiled :P This problem contains fake spoilers. ***

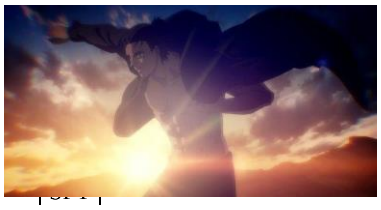
Given Dictionaries:

titan_eaters= {"Eren Yeager": ["Grisha Yeager", "Lara Tybur"], "Grisha Yeager": ["Eren Kruger", "Frieda Reiss"], "Porco": ["Marcel"], "Marcel": ["Ymir"], "Falco": ["Porco"], "Armin": ["Pieck"], "Levi Ackerman": ["Zeke Yeager"], "Mikasa Ackerman": ["Levi", "Annie"], "Commander Erwin": ["Berethold"]}

original_titans= {"Founder": "Frieda Reiss", "Attack": "Eren Kruger", "Armored": "Reiner", "Colossal": "Berethold", "Female": "Annie", "Jaw": " Ymir", "Cart": "Pieck", "Warhammer": "Lara Tybur", "Beast": "Zeke Yeager"}

Output:

{"Eren Yeager": ["Founder", "Attack", "Warhammer"], "Falco": ["Jaw"], "Mikasa Ackerman": ["Beast", "Female"], "Commander Erwin": ["Colossal"], "Reiner": ["Armored"], "Armin": ["Cart"]}



App #17: Minecraft Crafting Table [Difficulty Level: As Easy as Mining Bedrock]

Imagine due to multiple realities collapsing in on each other, we ended up in an entirely different dimension, where everything is made of Minecraft blocks. Now in order to survive the hordes of zombies and skeletons, you have to quickly design a program that provides you with the recipe of crafting a few necessary items. The recipe includes all the ingredients needed to craft an object of your choosing, the sub-ingredients needed to craft the ingredients and so on. The process will continue until there are no more sub-ingredient.

Copy the following dictionary directly in your code. If a specific item is not found within this dictionary, simply print "Recipe not found". Otherwise, print a dictionary containing all the ingredients (including quantity) necessary to craft that particular item.

craft= { "redstone torch": {"torch": 1, "redstone": 1}, "iron sword": {"iron ingot": 2, "stick": 1}, "torch": {"stick": 1, "coal": 1}, "stick": {"planks": 1}, "boat": {"planks": 5}, "planks": {"oak log": 1}, "bed": {"wool": 3, "planks": 3}, "beacon": {"nether star": 1, "glass": 5, "obsidian": 3}, "furnace": {"cobblestone": 8}, "blast furnace": {"furnace": 1, "iron ingot": 5, "smooth stone": 3} }

Sample Input #1: redstone torch **Output:** {"torch": 1, "redstone": 1, "stick": 1, "coal": 1, "planks": 1, "oak log": 1}	**Sample Input #2:** iron sword **Output:** {"iron ingot": 2, "stick": 1, "planks": 1, "oak log": 1}
Sample Input #3: grindstone **Output:** Recipe not found	**Sample Input #4:** blast furnace **Output:** {"furnace": 1, "iron ingot": 5, "smooth stone": 3, "cobblestone": 8}

Hero of the Village
Villagers, Thanking me for saving them of the raid
Me, who provoked the raid because I wanted lower prices

App #18: Spotify Playlist Modifier [Difficulty Level: In the end it doesn't even matter]

Imagine your Spotify subscription just ended, and you are thinking about renewing your subscription. However, you are not satisfied with some of the features of Spotify. So, you took it upon yourself to design a playlist modifier for Spotify which will add, delete, search, or sort songs in a playlist; create new playlists, blend multiple playlists together or delete playlists.

You should have a global dictionary that holds all your playlists. Each playlist will have a list of its own that holds its corresponding songs. Your program should have the following functions:

1. create_playlist (name)
2. add_songs (playlist, song)
3. delete_songs (playlist, song)
4. search_songs (playlist, song)
5. sort_songs (playlist)
6. blend_playlists (playlist_1, playlist_2, new_name)
7. delete_playlist (playlist)
8. show_playlist (playlist)
9. show_all_playlists ()

Function Calls #1

```
create_playlist ("Chill Hour")
add_songs ("Chill Hour", "Hello")
add_songs ("Chill Hour", "Beautiful Mistakes")
add_songs ("Chill Hour", "Echoes in Your Attic")
show_playlist ("Chill Hour")
search_songs ("Chill Hour", "Beautiful Mistakes")
delete_songs ("Chill Hour", "Beautiful Mistakes")
search_songs ("Chill Hour", "Beautiful Mistakes")
delete_songs ("Chill Hour", "In The End")
sort_songs ("Chill Hour")
create_playlist ("Come vibe with me")
add_songs ("Come vibe with me", "Apocalypse")
show_all_playlists ()
blend_playlists ("Chill Hour", "Come vibe with me", "Colab")
show_playlist ("Colab")
delete_playlist ("Chill Hour")
show_playlist ("Chill Hour")
```

Output

```
New Playlist Created: Chill Hour
Song Hello added to Chill Hour
Song Beautiful Mistakes added to Chill Hour
Song Echoes in Your Attic added to Chill Hour
Chill Hour: ["Hello", "Beautiful Mistakes", "Echoes in Your Attic"]
Beautiful Mistakes found at position 2 in Chill Hour
Beautiful Mistakes removed from Chill Hour
Beautiful Mistakes not found in Chill Hour
No song named In The End in Chill Hour
Chill Hour playlist sorted alphabetically
New Playlist Created: Come vibe with me
Song Apocalypse added to Come vibe with me
{"Chill Hour": ["Echoes in Your Attic", "Hello"], "Come vibe with me":
["Apocalypse"]}
Chill Hour and Come vibe with me blended into Colab
Colab: ["Echoes in Your Attic", "Hello", "Apocalypse"]
Playlist: Chill Hour deleted
No playlist named Chill Hour found
```

Function Calls #2

```
create_playlist ("Twisted Bliss")
add_songs ("Twisted Bliss", "Middle of the Night")
add_songs ("Twisted Bliss", "After Hours")
add_songs ("Twisted Bliss", "Levitating")
show_playlist ("Twisted Bliss")
search_songs ("Twisted Bliss", " Levitating")
delete_songs ("Twisted Bliss", "Awake")
search_songs ("Twisted Bliss", "Awake")
delete_songs ("Twisted Bliss", "Letters from the Sky")
sort_songs ("Twisted Bliss")
create_playlist ("JoJo")
add_songs ("JoJo", "The Pillar Men Theme")
show_all_playlists ()
blend_playlists ("JoJo", " Twisted Bliss", "Colab2")
show_playlist ("Colab2")
delete_playlist ("Love or Hate")
show_playlist ("JoJo")
```

Output

```
New Playlist Created: Twisted Bliss
Song Middle of the Night added to Twisted Bliss
Song After Hours added to Twisted Bliss
Song Levitating added to Twisted Bliss
Twisted Bliss: ["Middle of the Night", "After Hours", "Levitating"]
Levitating found at position 3 in Twisted Bliss
No song named Awake in Twisted Bliss
Awake not found in Twisted Bliss
No song named Letters from the Sky in Twisted Bliss
Twisted Bliss playlist sorted alphabetically
New Playlist Created: JoJo
Song The Pillar Men Theme added to JoJo
{" Twisted Bliss": ["After Hours", "Levitating", "Middle of the Night"], "JoJo": ["The
Pillar Men Theme"]}
Twisted Bliss and JoJo blended into Colab2
Colab: ["The Pillar Men Theme", "After Hours", "Levitating", "Middle of the Night"]
No playlist named Love or Hate found
"JoJo": ["The Pillar Men Theme"]
```