1.Ramesh and Ashwini are college besties. They used to share all the memories that happened in each day. Aditya who is also friend of both, used to tease them always like a lover. He is more interested in playing many fun games and also Pythonista, so he decided to create a python code for FLAMES games between his friends.

**Algorithm:**

Step 1: Define a function remove\_common\_letters(name1, name2):

Step 2: Takes two names as input.

Step 3: Converts each name to lowercase and creates sets of unique characters for each name.

Step 4: Finds the common letters between the two names using set intersection.

Step 5: Removes common letters from both names.

Step 6: Returns the modified names with common letters removed.

Step 7: Define a function flames\_game(name1, name2):

Step 8: Takes two names as input.

Step 9: Initializes a list relationship\_status containing the flames relationship statuses: 'Friendship', 'Love', 'Affection', 'Marriage', 'Enmity', 'Sibling'.

Step 10: Initializes a list letters containing the letters 'FLAMES'.

Step 11: Calls remove\_common\_letters function to remove common letters from both names.

Step 12: Calculates the total number of remaining letters after removing common letters.

Step 13: While the length of the letters list is greater than 1

Step 14: Calculates the index of the letter to be removed using the formula (remaining\_letters - 1) % len(letters).

Step 15: Removes the letter at the calculated index from the letters list.

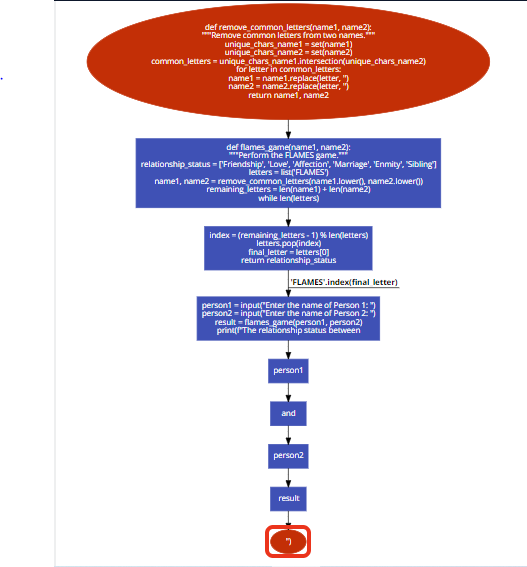
Step 16: Finds the final remaining letter after applying the FLAMES game algorithm.

Step 17: Returns the relationship status corresponding to the final letter from the relationship\_status list.

Step 18: Calls the flames\_game function with the input names.

Step 19: Prints the resulting relationship status between the two people.

**Flowchart:**



**Code**:

def remove\_common\_letters(name1, name2):

"""Remove common letters from two names."""

unique\_chars\_name1 = set(name1)

unique\_chars\_name2 = set(name2)

common\_letters = unique\_chars\_name1.intersection(unique\_chars\_name2)

for letter in common\_letters:

name1 = name1.replace(letter, '')

name2 = name2.replace(letter, '')

return name1, name2

def flames\_game(name1, name2):

"""Perform the FLAMES game."""

relationship\_status = ['Friendship', 'Love', 'Affection', 'Marriage', 'Enmity', 'Sibling']

letters = list('FLAMES')

name1, name2 = remove\_common\_letters(name1.lower(), name2.lower())

remaining\_letters = len(name1) + len(name2)

while len(letters) > 1:

index = (remaining\_letters - 1) % len(letters)

letters.pop(index)

final\_letter = letters[0]

return relationship\_status['FLAMES'.index(final\_letter)]

person1 = input("Enter the name of Person 1: ")

person2 = input("Enter the name of Person 2: ")

result = flames\_game(person1, person2)

print(f"The relationship status between {person1} and {person2} is: {result}")

2. Aibhav and Manisha are bored because of Pongal holidays. They are learning Python programming language. So, they thought to test their programming skills by doing coding, they tried different scenario but still boredom follows, so they thought to play HANGMAN game. They created 5 complicated words for the game, in the starting of the game we need to give our name, then we receive jumbled words of the input, we need to find out correctly, if we found out we get one point and if we lose, we get zero. So parallelly we need to ask the player at the end of each round, we need to continue or quit. If we quit the game then it will show the points or else it will continue for the next round.

**Algorithm**:

Step 1: Define a function choose\_word():

Step 2: Returns a random word chosen from a predefined list of words.

Step 3: Define a function jumble\_word(word):

Step 4: Takes a word as input.

Step 5: Converts the word to a list of characters.

Step 6: Shuffles the characters randomly using the random.shuffle() function.

Step 7: Joins the shuffled characters back into a string and returns the jumbled word.

Step 8: Define a function play\_hangman():

Step 9: Prompts the player to enter their name.

Step 10: Greets the player and informs them that they are about to play the Hangman game.

Step 11: Initializes a variable point to keep track of the player's score.

Step 12: Chooses a random word using the choose\_word() function.

Step 13: Jumbles the chosen word using the jumble\_word() function.

Step 14: Prints the jumbled word for the player to guess.

Step 15: Prompts the player to enter their guess.

Step 16: Checks if the player's guess matches the original word:

Step 17: If correct, prints "Correct!" and increments the player's points.

Step 18: If incorrect, prints "Incorrect!".

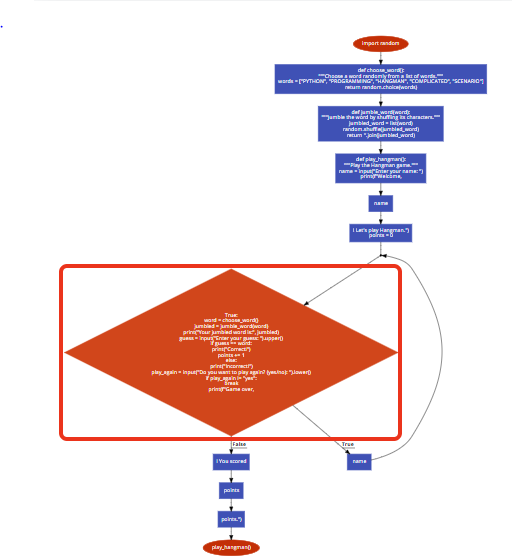
Step 19: Asks the player if they want to play again.

Step 20: If the player chooses not to play again, exits the loop.

Step 21: Prints the player's final score.

Step 22: Calls the play\_hangman() function to start the game.

**Flowchart**:



**Code**:

import random

def choose\_word():

"""Choose a word randomly from a list of words."""

words = ["PYTHON", "PROGRAMMING", "HANGMAN", "COMPLICATED", "SCENARIO"]

return random.choice(words)

def jumble\_word(word):

"""Jumble the word by shuffling its characters."""

jumbled\_word = list(word)

random.shuffle(jumbled\_word)

return ''.join(jumbled\_word)

def play\_hangman():

"""Play the Hangman game."""

name = input("Enter your name: ")

print(f"Welcome, {name}! Let's play Hangman.")

points = 0

while True:

word = choose\_word()

jumbled = jumble\_word(word)

print("Your jumbled word is:", jumbled)

guess = input("Enter your guess: ").upper()

if guess == word:

print("Correct!")

points += 1

else:

print("Incorrect!")

play\_again = input("Do you want to play again? (yes/no): ").lower()

if play\_again != "yes":

break

print(f"Game over, {name}! You scored {points} points.")

# Play the game

play\_hangman()

3. Kithan is mathematical genius, and his friend Krishna is a Pythonista. They do share knowledge by sharing some questions, so that they both can improve in their domain. So Kithan asked Krishna to do Python code for magic square. Krishan didn’t not understand the concept. So Kithan explained step by step,

1. We need to get the input from the user like natural numbers (5, 6, 7, 9…)

2. If the number is 5, then we need to create 5 rows and 5 columns

3. Its not limited with above steps, the sum of rows, columns and even diagonals should be same.

4. Only condition is the number should be ODD, even it wont work!

Example:

Magic Squre for n = 3

Sum of each row or column 15.0

2 7 6

9 5 1

4 3 8

**Algorithm**:

Step 1: Define a function generate\_magic\_square(n):

Step 2: Takes an odd number n as input.

Step 3: Checks if n is even. If it is, prints a message stating that a magic square is only possible for odd numbers and returns.

Step 4: Initializes an n×n matrix magic\_square with all elements initialized to 0.

Step 5: Initializes the starting position i and j at the top middle cell.

Step 6: Initializes a counter num starting from 1 to n×n representing the numbers to be filled in the magic square.

Step 7: Enters a loop that continues until all cells are filled with numbers from 1 to n×n:

Step 8: Assigns the current number num to the cell at position(i,j) in the magic square.

Increments num by 1.

Step 9: Calculates the next position (newi,newj) by moving diagonally up and to the right from the current position (i,j), considering the boundaries of the matrix using modular arithmetic.

Step 10: If the next position is already filled, moves one cell down from the current position.

Step 11: Updates the current position (i,j) to the next position (newi,newj)

Step 12: Returns the generated magic square.

Step 13: Define a function print\_magic\_square(square):

Step 14: Takes a magic square square as input.

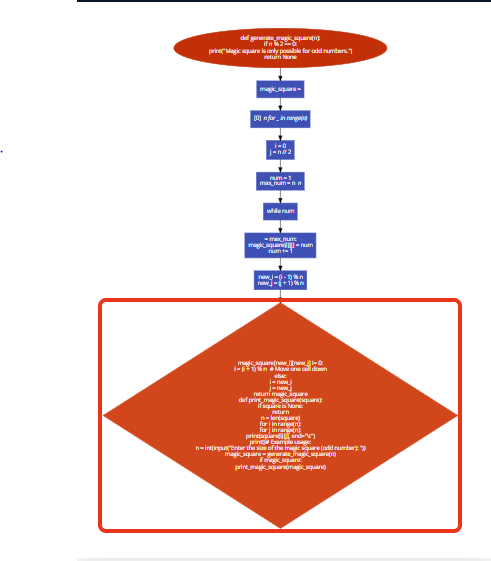
Step 15: Prints the magic square in a tabular format.

Step 16: Prompt the user to input the size of the magic square n

Step 17: Generate the magic square using the generate\_magic\_square(n) function.

Step 18: Print the generated magic square using the print\_magic\_square(magic\_square) function.

**Flowchart:**



**Code:**

def generate\_magic\_square(n):

if n % 2 == 0:

print("Magic square is only possible for odd numbers.")

return None

magic\_square = [[0] \* n for \_ in range(n)]

# Initial position

i = 0

j = n // 2

num = 1

max\_num = n \* n

while num <= max\_num:

magic\_square[i][j] = num

num += 1

new\_i = (i - 1) % n

new\_j = (j + 1) % n

if magic\_square[new\_i][new\_j] != 0:

i = (i + 1) % n # Move one cell down

else:

i = new\_i

j = new\_j

return magic\_square

def print\_magic\_square(square):

if square is None:

return

n = len(square)

for i in range(n):

for j in range(n):

print(square[i][j], end="\t")

print()

# Example usage:

n = int(input("Enter the size of the magic square (odd number): "))

magic\_square = generate\_magic\_square(n)

if magic\_square:

print\_magic\_square(magic\_square)

4. Write a python code for SNAKE and LADDER game by using random module.

Condition:

1. Need to two players, so whoever reach the number 30, wins

2. Check mate numbers are, if the player reach 17 then move to 4, 19 then move 7, 21 then move to 9, if 27 then move to 1.

17 19 21 27

4 7 9 1

Ladder numbers are, which means if the player reaches the numbers like 11, 3, 5 and 20, then the player should reach the number 22, 8, 26, 29 as a bonus.

11 3 5 20

22 8 26 29

**Algorithm:**

Step 1: Define a function snake\_ladder(player, current\_position):

Step 2: Takes the current player and their current position as input.

Step 3: Defines dictionaries snake\_positions and ladder\_positions to store the positions where snakes and ladders are present, along with their corresponding target positions.

Step 4: Checks if the current position is in snake\_positions or ladder\_positions.

Step 5: If the current position is in snake\_positions, the player is bitten by a snake and moves to the target position.

Step 6: If the current position is in ladder\_positions, the player finds a ladder and moves to the target position.

Step 7: Returns the updated position after considering snakes and ladders.

Step 8: Define a function snake\_ladder\_game():

Step 9: Initializes the positions of Player 1 and Player 2 to 0.

Step 10: Enters a loop until either Player 1 or Player 2 reaches position 30:

Step 11: Player 1 rolls the dice by pressing Enter.

Step 12: Generates a random number between 1 and 6 as the dice roll.

Step 13: Updates Player 1's position by adding the dice roll and checks for snakes and ladders using the snake\_ladder() function.

Step 14: Prints Player 1's current position.

Step 15: Checks if Player 1 has reached position 30 and declares Player 1 as the winner if so.

Step 16: Player 2 rolls the dice by pressing Enter.

Step 17: Generates a random number between 1 and 6 as the dice roll.

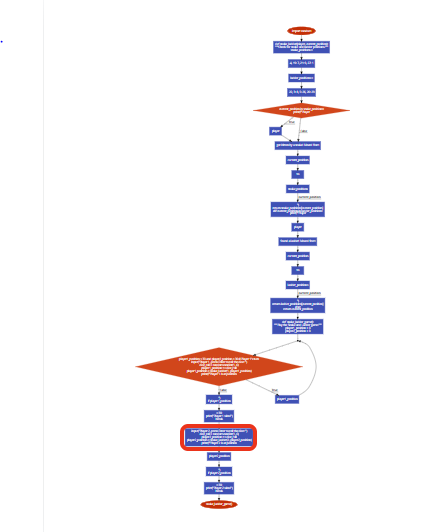
Step 18: Updates Player 2's position by adding the dice roll and checks for snakes and ladders using the snake\_ladder() function.

Step 19: Prints Player 2's current position.

Step 20: Checks if Player 2 has reached position 30 and declares Player 2 as the winner if so.

Step 21: Play the game by calling the snake\_ladder\_game() function.

**Flowchart:**



**Code:**

import random

def snake\_ladder(player, current\_position):

"""Check for snake and ladder positions."""

snake\_positions = {17: 4, 19: 7, 21: 9, 27: 1}

ladder\_positions = {11: 22, 3: 8, 5: 26, 20: 29}

if current\_position in snake\_positions:

print(f"Player {player} got bitten by a snake! Moved from {current\_position} to {snake\_positions[current\_position]}")

return snake\_positions[current\_position]

elif current\_position in ladder\_positions:

print(f"Player {player} found a ladder! Moved from {current\_position} to {ladder\_positions[current\_position]}")

return ladder\_positions[current\_position]

else:

return current\_position

def snake\_ladder\_game():

"""Play the Snake and Ladder game."""

player1\_position = 0

player2\_position = 0

while player1\_position < 30 and player2\_position < 30:

# Player 1's turn

input("Player 1, press Enter to roll the dice: ")

dice\_roll = random.randint(1, 6)

player1\_position += dice\_roll

player1\_position = snake\_ladder(1, player1\_position)

print(f"Player 1 is at position {player1\_position}")

if player1\_position >= 30:

print("Player 1 wins!")

break

# Player 2's turn

input("Player 2, press Enter to roll the dice: ")

dice\_roll = random.randint(1, 6)

player2\_position += dice\_roll

player2\_position = snake\_ladder(2, player2\_position)

print(f"Player 2 is at position {player2\_position}")

if player2\_position >= 30:

print("Player 2 wins!")

break

# Play the game

snake\_ladder\_game()