```
#Q1
import random

# Define the colors of the spaces on the roulette wheel
red_numbers = [1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34, 36]
black_numbers = [i for i in range(1, 37) if i not in red_numbers]

# Spin the roulette wheel
spin = random.randint(0, 38)

# Check if the spin landed on 0 or 00
if spin == 0:
    print("The spin resulted in 0... Pay 0")
elif spin == 38:
    print("The spin resulted in 00... Pay 00")
else:
    # Check the color of the space
    if spin in red_numbers:
        color = "Red"
    else:
        color = "Black"

    # Check if the number is odd or even
    if spin % 2 == 0:
        odd_even = "Even"
    else:
        odd_even = "Odd"

    # Check if the number is in the first half or second half of the range
    if spin <= 18:
        range_ = "1 to 18"
    else:
        range_ = "19 to 36"

    # Print the results
    print(f"The spin resulted in {spin}... Pay {spin}")
    print(f"Pay {color}")
    print(f"Pay {odd_even}")
    print(f"Pay {range_}")

#Q2
# Get the year from the user
year = int(input("Enter a year: "))
```

```
# Check if the year is divisible by 400 or (divisible by 4 and not divisible by 100)
if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
    # If the year is a leap year, print a message indicating so
    print(year, "is a leap year.")
else:
    # If the year is not a leap year, print a message indicating so
    print(year, "is not a leap year.")

#Q3
# Define the animals in the Chinese zodiac and their corresponding years
zodiac = {
    0: "Monkey",
    1: "Rooster",
    2: "Dog",
    3: "Pig",
    4: "Rat",
    5: "Ox",
    6: "Tiger",
    7: "Hare",
    8: "Dragon",
    9: "Snake",
    10: "Horse",
    11: "Sheep"
}

# Read the year from the user
year = int(input("Enter a year: "))

# Calculate the zodiac index for the given year
zodiac_index = (year - 2000) % 12

# Get the animal associated with the zodiac index
animal = zodiac[zodiac_index]

# Print the animal associated with the given year
print(year, "is the Year of the", animal)


#Q4
# Read input from user
num_sides = int(input("Enter the number of sides (3-10): "))

# Define dictionary mapping number of sides to shape names
shape_names = {
    3: "Triangle",
    4: "Quadrilateral",
    5: "Pentagon",
    6: "Hexagon",
    7: "Heptagon",
    8: "Octagon",
    9: "Nonagon",
    10: "Decagon"
}

# Check if the input is within the valid range
if num_sides < 3 or num_sides > 10:
    print("Error: Invalid number of sides. Please enter a number between 3 and 10.")
else:
    # Retrieve the shape name from the dictionary based on the number of sides
    shape_name = shape_names[num_sides]
    print("The shape with", num_sides, "sides is a", shape_name)
```

QUESTION 5

```
# Initialize variables to store age and admission cost
total_admission_cost = 0.0

# Loop to read ages of guests from user
while True:
    age_str = input("Enter age (or leave blank to finish): ")

    # Check if input is blank, if so, exit the loop
    if age_str == "":
        break

    age = int(age_str)

    # Determine admission cost based on age
    if age <= 2:
        admission_cost = 0.0
    elif age >= 3 and age <= 12:
        admission_cost = 14.0
    elif age >= 65:
        admission_cost = 18.0
    else:
        admission_cost = 23.0

    total_admission_cost += admission_cost
```

```
# Display total admission cost with appropriate message
print("Total admission cost for the group: £{:.2f}".format(total_admission_cost))

#Q6

import math

# Read the first x-coordinate from the user
x1 = float(input("Enter the first x-coordinate: "))

# Read the first y-coordinate from the user
y1 = float(input("Enter the first y-coordinate: "))

# Initialize variables to store previous x and y coordinates
prev_x = x1
prev_y = y1

# Initialize variable to store the perimeter
perimeter = 0.0

# Loop to read additional coordinates from the user
while True:
    # Read the next x-coordinate from the user (or blank line to quit)
    x_str = input("Enter the next x-coordinate (blank to quit): ")
    if x_str == "":
        break

    x = float(x_str)

    # Read the next y-coordinate from the user
    y = float(input("Enter the next y-coordinate: "))

    # Compute the distance between current and previous points
    distance = math.sqrt((x - prev_x) ** 2 + (y - prev_y) ** 2)

    # Add the distance to the perimeter
    perimeter += distance

    # Update the previous x and y coordinates
    prev_x = x
    prev_y = y

# Compute the distance from the last point back to the first point
```

```
distance = math.sqrt((x1 - prev_x) ** 2 + (y1 - prev_y) ** 2)

# Add the distance to the perimeter
perimeter += distance

# Display the computed perimeter
print("The perimeter of that polygon is {:.15f}".format(perimeter))

#Q7
while True:
    # Read 8 bits from the user
    bits = input("Enter 8 bits: ")

    # Check if the user entered a blank line
    if not bits:
        break

    # Check if the user entered exactly 8 bits
    if len(bits) != 8:
        print("Error: Please enter exactly 8 bits.")
        continue

    # Count the number of ones in the input string
    num_ones = bits.count('1')

    # Determine if the parity bit should be 0 or 1
    parity_bit = '0' if num_ones % 2 == 0 else '1'

    # Display the result to the user
    print("The parity bit should be:", parity_bit)

#Q8
    # Loop through numbers from 1 to 100
for num in range(1, 101):
    output = ""
    # Check if the number is divisible by 3
    if num % 3 == 0:
        output += "Fizz"
    # Check if the number is divisible by 5
    if num % 5 == 0:
        output += "Buzz"
    # If not divisible by 3 or 5, use the number itself
    if output == "":
        output = str(num)
    # Display the output for the current number
    print(output)

#Q9
def caesar_cipher(message, shift):
    shifted_message = ""
    for char in message:
        if char.isalpha():
            if char.isupper():
                shifted_char = chr((ord(char) - 65 + shift) % 26 + 65)
            else:
                shifted_char = chr((ord(char) - 97 + shift) % 26 + 97)
        else:
            shifted_char = char
        shifted_message += shifted_char
    return shifted_message

message = input("Enter the message to be encrypted/decrypted: ")
shift = int(input("Enter the shift value: "))
encrypted_message = caesar_cipher(message, shift)
print("The encrypted/decrypted message is:", encrypted_message)

#Q10
import random

def sing_verse(num):
    print(f"There are {num} green bottles hanging on the wall, {num} green bottles hanging on the
wall,")
    print("And if 1 green bottle should accidentally fall,")

def play_game():
    num_bottles = 10
    while num_bottles > 0:
        sing_verse(num_bottles)
        while True:
            answer = input("How many green bottles will be hanging on the wall? ")
            if answer.isdigit() and int(answer) == num_bottles - 1:
                num_bottles -= 1
                print(f"There will be {num_bottles} green bottles hanging on the wall\n")
                break
            else:
                print("No, try again\n")
    print("There are no more green bottles hanging on the wall.")
```

```
play_game()

#Q11
# Function to calculate the ordinal day within the year for a given date
def ordinal_date(day, month, year):
    # List of days in each month, considering leap years
    days_in_month = [31, 28 if not year % 4 == 0 else 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    # Calculate the sum of days for the months preceding the given month
    # up to the given month (not inclusive)
    ordinal_day = sum(days_in_month[:month-1]) + day
    # If the given year is a leap year and the month is after February,
    # add 1 to the ordinal day to account for the extra day in February
    if month > 2 and year % 4 == 0:
        ordinal_day += 1
    return ordinal_day

# Main program
if __name__ == '__main__':
    # Read input from user for day, month, and year
    day = int(input('Enter the day: '))
    month = int(input('Enter the month: '))
    year = int(input('Enter the year: '))
    # Call the ordinal_date function to calculate the ordinal day
    ordinal_day = ordinal_date(day, month, year)
    # Print the result
    print('The day within the year is:', ordinal_day)




#Q12
# Q12
def is_triangle(a, b, c):
    if a <= 0 or b <= 0 or c <= 0:
        return False
    if a >= b + c or b >= a + c or c >= a + b:
        return False
    return True

if __name__ == '__main__':
    a = float(input('Enter the length of the first side: '))
    b = float(input('Enter the length of the second side: '))
    c = float(input('Enter the length of the third side: '))
    if is_triangle(a, b, c):
        print('These lengths can form a triangle.')
    else:
        print('These lengths cannot form a triangle.')


#Q13
def capitalize_string(string):
    # Split the string into a list of words
    words = string.split()
    # Iterate over the words and capitalize the appropriate characters
    for i in range(len(words)):
        # Remove leading/trailing spaces and capitalize the first non-space character in the word
        words[i] = words[i].strip().capitalize()
        # Capitalize the first non-space character after a period, exclamation mark, or question
mark
        if i > 0 and (words[i-1].endswith('.') or words[i-1].endswith('!') or words[i-1].endswith('?')):
            words[i] = words[i].capitalize()
        # Capitalize "i" if it's surrounded by spaces or punctuation
        if words[i] == 'i' and (i == 0 or words[i-1] in [' ', '.', '!', '?', '"']) and (i == len(words)-1 or
words[i+1] in [' ', '.', '!', '?', '"']):
            words[i] = 'I'
    # Join the list of words back into a string and return it
    return ' '.join(words)


if __name__ == '__main__':
    string = input('Enter a string to be capitalized: ')
    capitalized_string = capitalize_string(string)
    print(capitalized_string)


#Q14
def is_good_password(password):
    """
    Returns True if the password is good, False otherwise.
    A good password is at least 8 characters long and contains
    at least one uppercase letter, one lowercase letter, and one number.
    """
    if len(password) < 8:
        return False
    has_uppercase = False
    has_lowercase = False
    has_number = False
    for char in password:
        if char.isupper():
            has_uppercase = True
        elif char.islower():
            has_lowercase = True
        elif char.isdigit():
            has_number = True
    return has_uppercase and has_lowercase and has_number

if _name_ == '_main_':
    password = input("Enter a password: ")
    if is_good_password(password):
        print("Good password!")
    else:
        print("Not a good password.")



def convert_volume(num_units, unit):
    teaspoons = {"teaspoon": 1, "tablespoon": 3, "cup": 48}
    tablespoons = {"tablespoon": 1, "cup": 16}
    cups = {"cup": 1}

    if unit == "teaspoon":
        total_tsp = num_units
    elif unit == "tablespoon":
        total_tsp = num_units * teaspoons["tablespoon"]
    else:
        total_tsp = num_units * teaspoons["cup"]

    total_tbsp = total_tsp / teaspoons["tablespoon"]
    total_cups = total_tbsp / tablespoons["cup"]

    # calculate the remaining tablespoons and teaspoons
    remaining_tbsp = int(total_tbsp % tablespoons["cup"])
    remaining_tsp = int(total_tsp % teaspoons["tablespoon"])

    # create the result string
    result = ""
    if total_cups > 0:
        result += str(int(total_cups)) + " cup"
        if total_cups > 1:
            result += "s"
        if remaining_tbsp > 0 or remaining_tsp > 0:
            result += ", "
    if remaining_tbsp > 0:
        result += str(remaining_tbsp) + " tablespoon"
        if remaining_tbsp > 1:
            result += "s"
        if remaining_tsp > 0:
            result += ", "
    if remaining_tsp > 0:
        result += str(remaining_tsp) + " teaspoon"
        if remaining_tsp > 1:
            result += "s"

    return result


num_units = int(input("Enter the number of units: "))
unit = input("Enter the unit of measure (teaspoon, tablespoon, or cup): ")
result = convert_volume(num_units, unit)
print(result)




#Q16
def isSublist(larger, smaller):
    if len(smaller) == 0:
        return True
    for i in range(len(larger)):
        if larger[i:i+len(smaller)] == smaller:
            return True
    return False

larger = [1, 2, 3, 4, 5, 6]
smaller1 = [2, 3, 4]
smaller2 = [3, 5, 6]
smaller3 = [7]
smaller4 = []

print(isSublist(larger, smaller1))  # True
print(isSublist(larger, smaller2))  # True
print(isSublist(larger, smaller3))  # False
print(isSublist(larger, smaller4))  # True
print(isSublist(larger, larger))    # True

#Q17
```

```
def get_all_sublists(lst):
    """
    Get all possible sublists of a list.

    Args:
        lst (list): The input list.

    Returns:
        list: A list containing all possible sublists of the input list.
    """
    if not lst:
        # return empty list if input list is empty
        return [[]]

    # get all sublists excluding the first element
    sublists = get_all_sublists(lst[1:])

    # add the first element to each sublist
    sublists_with_first = [[lst[0]] + sublist for sublist in sublists]

    # combine both sets of sublists
    return sublists + sublists_with_first


# main program to demonstrate the get_all_sublists function
lst1 = [1, 2, 3]
lst2 = ['a', 'b', 'c']
lst3 = [10, 20]
lst4 = []

print("All sublists of", lst1)
print(get_all_sublists(lst1))

print("\nAll sublists of", lst2)
print(get_all_sublists(lst2))

print("\nAll sublists of", lst3)
print(get_all_sublists(lst3))

print("\nAll sublists of", lst4)
print(get_all_sublists(lst4))
```

#Q18

```
import random

def create_bingo_card():
    """
    Create a random Bingo card and store it in a dictionary.

    Returns:
        dict: A dictionary representing a Bingo card with keys as B, I, N, G, O and values as lists of five numbers
        that appear under each letter.
    """
    # Create a dictionary to store the Bingo card
    bingo_card = {'B': [], 'I': [], 'N': [], 'G': [], 'O': []}

    # Generate and store random numbers for each column
    for key in bingo_card:
        # Generate 5 unique random numbers within the appropriate range for each column
        numbers = random.sample(range((key == 'B') * 1 + (key == 'I') * 16 + (key == 'N') * 31 + (key == 'G') * 46 + (key == 'O') * 61,
                                       (key == 'B') * 16 + (key == 'I') * 31 + (key == 'N') * 46 + (key == 'G') * 61 + (key == 'O') * 76),
                               5)
        # Sort the numbers in ascending order and store them in the dictionary
        bingo_card[key] = sorted(numbers)

    return bingo_card


def display_bingo_card(bingo_card):
    """
    Display a Bingo card with the columns labelled appropriately.

    Args:
        bingo_card (dict): A dictionary representing a Bingo card with keys as B, I, N, G, O and values as lists of
        five numbers that appear under each letter.
    """
    print("B\tI\tN\tG\tO")
    for i in range(5):
        print("\t".join(str(bingo_card[key][i]) for key in bingo_card))


# Main program
if __name__ == "__main__":
    # Create a random Bingo card
    card = create_bingo_card()

    # Display the Bingo card
    print("Random Bingo Card:")
    display_bingo_card(card)


#Q19
import random

def create_bingo_card():
    card = {
        'B': [],
        'I': [],
        'N': [],
        'G': [],
        'O': []
    }
    for letter in card:
        lower_bound = 1 + 15 * ('BINGO'.index(letter))
        upper_bound = lower_bound + 15
        card[letter] = random.sample(range(lower_bound, upper_bound), 5)
    return card

def display_bingo_card(card):
    print("{:<3} {:<3} {:<3} {:<3} {:<3}".format("B", "I", "N", "G", "O"))
    for row in range(5):
        print("{:<3} {:<3} {:<3} {:<3} {:<3}".format(card['B'][row], card['I'][row], card['N'][row],
card['G'][row], card['O'][row]))
    print()

def has_winning_line(card):
    # Check for horizontal lines
    for letter in card:
        if sum(card[letter]) == 0:
            return True

    # Check for vertical lines
    for col in range(5):
        if sum([card[letter][col] for letter in card]) == 0:
            return True

    # Check for diagonal lines
    if (card['B'][0] == 0 and card['I'][1] == 0 and card['N'][2] == 0 and card['G'][3] == 0 and
card['O'][4] == 0) or \
       (card['O'][0] == 0 and card['G'][1] == 0 and card['N'][2] == 0 and card['I'][3] == 0 and
card['B'][4] == 0):
        return True

    # No winning line found
    return False

# Main program
if __name__ == '__main__':
    card1 = create_bingo_card()
    display_bingo_card(card1)
    print("Winning card:", has_winning_line(card1))

    card2 = create_bingo_card()
    card2['B'][2] = card2['I'][2] = card2['N'][2] = card2['G'][2] = card2['O'][2] = 0  # mark center as
called
    display_bingo_card(card2)
    print("Winning card:", has_winning_line(card2))

    card3 = create_bingo_card()
    card3['B'][0] = card3['I'][0] = card3['N'][0] = card3['G'][0] = card3['O'][0] = 0  # mark top row as
called
    display_bingo_card(card3)
    print("Winning card:", has_winning_line(card3))

    card4 = create_bingo_card()
    card4['B'][0] = card4['I'][1] = card4['N'][2] = card4['G'][3] = card4['O'][4] = 0  # mark diagonal
line as called
    display_bingo_card(card4)
    print("Winning card:", has_winning_line(card4))
```

```
#Q20
import random

# Function to generate a random Bingo card
def generate_card():
    card = {"B": [], "I": [], "N": [], "G": [], "O": []}
    used = set() # To ensure no duplicate numbers are added to the card

    # Generate the numbers for each column
    for key in card:
        while len(card[key]) < 5:
            # Generate a random number in the valid range for the current column
            num = random.randint((ord(key)-65)*15+1, (ord(key)-64)*15)
            if num not in used:
                card[key].append(num)
                used.add(num)
    return card

# Function to check if a Bingo card contains a winning line
def has_winning_line(card):
    # Check horizontal lines
    for row in card.values():
        if row == [0, 0, 0, 0, 0]:
            return True
    # Check vertical lines
    for i in range(5):
        if [card[key][i] for key in card] == [0, 0, 0, 0, 0]:
            return True
    # Check diagonal lines
    if [card["B"][0], card["I"][1], card["N"][2], card["G"][3], card["O"][4]] == [0, 0, 0, 0, 0]:
        return True
    if [card["O"][0], card["G"][1], card["N"][2], card["I"][3], card["B"][4]] == [0, 0, 0, 0, 0]:
        return True
    return False

# Simulate 1000 games and record the number of calls made for each game
num_calls_list = []
for _ in range(1000):
    # Generate a new Bingo card for each game
    card = generate_card()

    # Generate a list of all valid Bingo calls and shuffle it
    calls = ["{}{}".format(key, num) for key in card for num in card[key]]
    random.shuffle(calls)

    # Cross out numbers on the card until a winning line is found
    num_calls = 0
    while not has_winning_line(card):
        num_calls += 1
        call = calls.pop()
        key, num = call[0], int(call[1:])
        card[key][card[key].index(num)] = 0
    num_calls_list.append(num_calls)

# Report the minimum, maximum and average number of calls needed to win
print("Minimum number of calls:", min(num_calls_list))
print("Maximum number of calls:", max(num_calls_list))
print("Average number of calls:", sum(num_calls_list)/len(num_calls_list))
```