```python
# PIC 16A HW1
# Name: William Martinez
# Collaborators: None
# Date: 4/18/24

import random # This is only needed in Problem 5
# random.seed(1)

# Problem 1

def print_s(s):
    """
    Prints a given string.
    ---
    Args:
        s: A string.
    Returns:
        None
    """
    print(s)

# you do not have to add docstrings for the rest of these print_s_*
  functions.

def print_s_lines(s):
    # replace colon with new line
    print(s.replace(': ', '\n'))

def print_s_parts(s):
    # remove all spaces then replace colon with new line.
    sos = s.replace(' ', '').replace(':', '\n')
    # Keep everyother element
    sos = sos.split(sep = '\n')[::2]
    # join list, sos, by new lines then print
    print('\n'.join(sos))

def print_s_some(s):
    # split string into list by new line
    # perform a decending sort, then drop fisrt element.
    sos = sorted(s.split('\n'), key = len, reverse = True)[1:]
    # join list into a string.
    sos = '\n'.join(sos)
    print(sos)

def print_s_change(s):
```

```
    # replace math with data science
    sc = s.replace('math', 'data science')
    # replace long division with machine learning
    sc = sc.replace('long division', 'machine learning')
    print(sc)

# Problem 2

def make_count_dictionary(L):
    """
    Return a dictionary of the frequency of elements in a list.
    ---
    Args:
        L: A list
    Returns:
        D: A dictionary of element frequencies
    """
    # Initialize d_c and d_v
    d_k = []
    d_v = []
    # Distinct elements in list, L, keeping the same order of list, L
    for i in L:
        if i not in d_k:
            d_k.append(i)
    # Frequency of distinct elements
    for i in d_k:
        d_v.append(L.count(i))
    # Dictionary of the frequencies of distinct elements
    D = dict(zip(d_k, d_v))
    return D

# Problem 3

def gimme_an_odd_number():
    """
    A loop that terminates when a user inputs an odd number.
    Returns a list of user inputted numbers.
    ---
    Args:
        None
    Returns:
        usr_list: A list of user responses
    """
    # initialize user responses
    usr = 0
```

```
    usr_list = []
    # Keep requesting integers till until input is odd. Append all responses
    # to usr_list then print
    while (usr % 2 == 0):
        usr = int(input("Please enter an integer."))
        usr_list.append(usr)
    # print(usr_list)
    return print(usr_list)

# Problem 4

def get_triangular_numbers(k):
    """
    Returns a list of the number of objects needed to make
    a k-sided, equalateral triangle from 1 to k.
    ---
    Args:
        k: An Integer
    Returns:
        num_list: A list of the number of objects needed to make
        a k-sided, equalateral triangle from 1 to k
    """
    # initialize num_list
    num_list = []
    # Append the integer corresponding to the number of objects needed to
      make
    # a k-sided, equalateral triangle from 1 to k
    for i in range(1, k + 1):
        num_list.append(int(i * (i + 1) / 2)) # Formula for triangle numbers
    return num_list

def get_consonants(s):
    """
    Returns a list of characters that are not a vowel, space, comma, or
period.
    ---
    Args:
        s: A string
    Returns:
        cp_list: A list
    """
    # list of characters that are a vowel, space, comma, or period.
    rm_list = ["a", "e", "i", "o", "u", " ", ",", ".",]
    # initialize cp_list
    cp_list = []
```

```
    # for each character, drop
    for i in s:
        if i in rm_list:
            pass
        else:
            cp_list.append(i)
    return cp_list


def get_list_of_powers(X, k):
    """
    Returns a 2 dimentional list of integers. Each element is a list
    of the powers of an element of X from 0 to k.
    ---
    Args:
        X: List of integers
        k: An Integer
    Returns:
        L: A 2-dimentional list
    """
    # Initialize the 2 dimentional list, L
    L = []
    # Initial the sub list, L_sub, then append the
    # the powers of each element X from 0 to k
    for i in X:
        L_sub = []
        for j in range(0, k + 1):
            L_sub.append(i**j)
        # Append each sub list, L_sub, to the 2 dimentional list, L
        L.append(L_sub)
    return L


def get_list_of_even_powers(X, k):
    """
    Returns a 2 dimentional list of integers. Each element is a list
    of the even powers of an element of X from 0 to k.
    ---
    Args:
        X: List of integers
        k: An Integer
    Returns:
        L: A 2-dimentional list
    """
    # Initialize the 2 dimentional list, L
```

```
    L = []
    # Initial the sub list, L_sub, then append the
    # the even powers of each element X from 0 to k
    for i in X:
        L_sub = []
        for j in range(0, k + 1, 2):
            L_sub.append(i**j)
        # Append each sub list, L_sub, to the 2 dimentional list, L
        L.append(L_sub)
    return L



# Problem 5

def random_walk(ub, lb):
    """
    Returns the last postion, position history, and step history for a fair
    coin toss where head moves forwards (+1) and tails moves backwards (-1).
    ---
    Args:
        ub: An integer that represents the upperbound position. When pos
reaches
        ub, the loop stops.
        lb: An integer that represents the lowererbound position. When pos
        reaches sub, the loop stops.
    Returns:
        pos: An integer that represents the last position.
        positions: A List that is a log of the position history.
        steps: A list that is a log of the step history.
    """
    # Initialize pos, positions, and steps. Start pos and positions at 0
    pos = 0
    positions = [0]
    steps = []
    # Loop random coin flips
    while True:
        # Break loop if upper or lower bounds reached. Drop last element.
        if (pos == ub):
            print("Upper bound at {} reached".format(ub))
            positions = positions[:-1]
            break
        elif (pos == lb):
            print("Lower bound at {} reached".format(lb))
            positions = positions[:-1]
```

```
            break
        # Perform coin flip, assign +1 to heads and -1 to tails. Append
          current
        # position, pos, to positions and append step result to steps.
        else:
            x = random.choice(["heads", "tails"])
            if x == "heads":
                pos += 1
                positions.append(pos)
                steps.append(1)
            elif x == "tails":
                pos -= 1
                positions.append(pos)
                steps.append(-1)
    return pos, positions, steps


# If you uncomment these two lines, you can run
# the gimme_an_odd_number() function by
# running this script on your IDE or terminal.
# Of course you can run the function in notebook as well.
# Make sure this stays commented when you submit
# your code.
#
# if __name__ == "__main__":
#     gimme_an_odd_number()
```