```
##************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 06 Problem 1
##************************************************************

import check
import math


def obey_benford(L):
  '''
  Returns the results from the chi squared test
  based on the list of positive numbers in L and what is a
  tolerable distance from what is predicted by Benford's Law

  obey_benford: (listof Nat) -> Float
  requires: Each element of L is positive.

  Example:
  L = [1]*301 + [2] * 176 + [3] * 125 + [4] * 97 + [5] * 79 \
              + [6] * 67  + [7] * 58  + [8] * 51 + [9] * 46
  obey_benford(L) => 0.0023622171891271 43
  L = L + [9] * 1000
  obey_benford(L) => 20854.345326782823
  obey_benford([]) => 0.0
  '''
  if L == []:
    return 0.0
  digits = []
  for num in L:
    digits.append(int(str(num)[0]))
  counts = []
  exs = []
  for d in range(1, 10):
    counts.append(digits.count(d))
    exs.append(math.log((d+1)/d, 10))
  n = len(L)
  chi_intermediate = []
  for d in range(9):
    chi_intermediate.append((counts[d] - n*exs[d])**2/(n*exs[d]))
  chi_squared = sum(chi_intermediate)
  return chi_squared

##Examples:

L = [1,2]
check.within("Example 1", obey_benford(L), 2.500400841077468, 0.00001)

L = [1,1337,2]
```

```
check.within("Example 2", obey_benford(L), 3.32219532272341, 0.00001)

L = [1]*301 + [2] * 176 + [3] * 125 + [4] * 97 + [5] * 79 \
            + [6] * 67  + [7] * 58  + [8] * 51 + [9] * 46
check.within("Example 3", obey_benford(L), 0.00236221718912714 3,
0.00001)

##Tests

check.within("Test 0 Empty", obey_benford([]), 0.0, 0.00001)

L = [1]*301 + [2] * 176 + [3] * 125 + [4] * 97 + [5] * 79 \
            + [6] * 67  + [7] * 58  + [8] * 51 + [9] * 1046
check.within("Test 1", obey_benford(L), 10432.473729532016, 0.00001)

L = [1]*1000 + [2] * 1000 + [3] * 1000 + [4] * 1000 + [5] * 1000 \
            + [6] * 1000  + [7] * 1000  + [8] * 1000 + [9] * 1000
check.within("Test 2: uniform", obey_benford(L), 3615.284636209621,
0.00001)
L = [1]*300 + [2] * 180 + [3] * 130 + [4] * 100 + [5] * 80 \
            + [6] * 65  + [7] * 60  + [8] * 50 + [9] * 46
check.within("Test 3: Perfect", obey_benford(L), 0.429976554716235 8,
0.00001)
L = [9] * 1000
check.within("Test 4: lots one value", obey_benford(L),
             20854.345326782823, 0.00001)
L = [1] * 1
check.within("Test 5: minimal", obey_benford(L), 2.321928094887362 6,
0.00001)
L = [6, 9, 7, 9, 8, 5, 8, 2, 5, 1, 7, 9, 9, 8, 1, 1, 6, 2, 9, 8, 1, 3,
2, 1,
     9, 8, 5, 8, 4, 2, 3, 1, 5, 4, 4, 4, 9, 6, 2, 9, 8, 8, 9, 8, 5, 8,
2, 3,
     1, 1, 1, 5, 6, 7, 3, 1, 7, 2, 9, 9, 5, 6, 2, 4, 4, 1, 8, 7, 4, 5,
2, 5,
     4, 6, 8, 5, 6, 8, 6, 9, 5, 7, 9, 3, 8, 4, 8, 3, 3, 7, 6, 1, 9, 9,
5, 5,
     1, 3, 2, 7, 4, 8, 7, 4, 6, 3, 7, 9, 4, 7, 7, 1, 8, 5, 5, 1, 6, 1,
8, 3,
     2, 7, 9, 9, 1, 7, 9, 9, 4, 9, 6, 1, 6, 3, 6, 4, 5, 4, 4, 8, 3, 6,
4, 2,
     7, 3, 8, 4, 8, 4, 4, 3, 9, 2, 5, 1, 6, 3, 4, 6, 4, 3, 8, 4, 4, 5,
9, 1,
     6, 6, 8, 4, 4, 9, 7, 6, 8, 5, 5, 3, 6, 9, 3, 7, 4, 3, 1, 5, 9, 9,
2, 7,
     1, 6, 9, 7, 9, 3, 8, 9, 9, 6, 8, 6, 7, 1, 2, 1, 7, 2, 6, 1, 2, 1,
5, 7,
     4, 9, 7, 3, 7, 5, 3, 7, 7, 9, 4, 2, 4, 4, 7, 9, 9, 1, 1, 2, 8, 4,
5, 5,
```

2, 8, 2, 8, 4, 8, 5, 2, 3, 9, 1, 9, 4, 3, 3, 4, 5, 7, 4, 5, 4, 5,
8, 8,
2, 9, 3, 4, 1, 6, 3, 1, 2, 7, 2, 8, 2, 9, 7, 9, 1, 2, 5, 2, 2, 6,
8, 6,
5, 4, 9, 6, 6, 6, 6, 8, 9, 9, 7, 4, 1, 9, 2, 5, 3, 4, 8, 5, 4, 2,
4, 9,
1, 1, 5, 6, 4, 6, 8, 5, 6, 4, 5, 6, 1, 5, 1, 6, 2, 1, 3, 3, 8, 2,
1, 3,
7, 5, 7, 5, 5, 9, 7, 9, 7, 6, 1, 4, 5, 1, 9, 2, 4, 4, 7, 8, 7, 9,
7, 8,
2, 9, 9, 9, 9, 1, 6, 1, 6, 2, 2, 3, 7, 4, 9, 7, 3, 3, 2, 8, 5, 6,
2, 7,
8, 4, 2, 3, 5, 6, 1, 9, 8, 7, 1, 8, 6, 1, 7, 1, 3, 3, 6, 1, 4, 3,
6, 4,
1, 8, 7, 8, 2, 2, 1, 9, 7, 6, 1, 1, 5, 5, 8, 7, 9, 5, 8, 4, 8, 3,
8, 1,
9, 5, 2, 9, 4, 1, 7, 2, 5, 7, 6, 8, 6, 3, 6, 5, 8, 3, 5, 6, 6, 3,
1, 3,
3, 1, 9, 1, 3, 8, 1, 1, 9, 6, 5, 9, 2, 8, 6, 5, 9, 9, 5, 7, 2, 2,
3, 4,
8, 5, 5, 9, 9, 7, 7, 9, 5, 5, 7, 2, 5, 6, 6, 8, 7, 7, 8, 7, 9, 5,
7, 4,
5, 6, 3, 4, 4, 4, 4, 5, 1, 9, 5, 3, 6, 4, 2, 6, 1, 9, 4, 1, 7, 2,
3, 8,
3, 9, 4, 4, 3, 4, 1, 7, 6, 7, 6, 2, 2, 2, 8, 9, 7, 6, 3, 8, 9, 4,
3, 4,
2, 3, 1, 4, 6, 2, 6, 3, 8, 6, 3, 1, 3, 4, 6, 7, 4, 8, 6, 4, 9, 3,
3, 4,
6, 7, 1, 4, 8, 9, 4, 8, 1, 2, 3, 9, 4, 6, 2, 6, 5, 1, 3, 1, 8, 8,
2, 6,
1, 1, 6, 4, 1, 8, 6, 9, 3, 9, 4, 1, 2, 4, 8, 1, 9, 1, 8, 2, 4, 1,
1, 7,
4, 8, 7, 8, 8, 2, 1, 5, 4, 1, 5, 2, 6, 3, 8, 5, 7, 7, 3, 2, 4, 5,
8, 8,
4, 9, 4, 7, 3, 8, 4, 1, 2, 6, 8, 9, 8, 5, 5, 2, 1, 7, 7, 2, 4, 7,
6, 3,
2, 2, 8, 4, 4, 3, 2, 9, 9, 5, 8, 2, 2, 4, 1, 2, 4, 3, 6, 7, 7, 5,
4, 8,
3, 9, 1, 7, 2, 7, 9, 5, 1, 1, 8, 5, 3, 1, 8, 9, 5, 1, 2, 6, 4, 7,
9, 9,
4, 3, 9, 1, 2, 3, 1, 5, 8, 7, 5, 2, 2, 5, 2, 4, 2, 9, 9, 3, 5, 4,
1, 4,
1, 7, 1, 6, 6, 1, 7, 7, 5, 2, 2, 4, 1, 4, 8, 8, 2, 9, 7, 5, 5, 3,
4, 2,
2, 6, 7, 7, 5, 2, 8, 9, 9, 9, 2, 6, 3, 6, 8, 7, 5, 2, 2, 4, 3, 8,
3, 4,
5, 1, 8, 4, 6, 6, 1, 2, 4, 6, 7, 8, 6, 4, 4, 9, 9, 4, 4, 9, 3, 4,
7, 8,
4, 7, 4, 9, 3, 7, 6, 4, 6, 9, 5, 8, 9, 8, 7, 4, 4, 7, 5, 3, 9, 8,
4, 8,

```
     2, 9, 3, 9, 7, 9, 2, 3, 4, 8, 6, 3, 7, 2, 7, 2, 5, 6, 4, 9, 6, 6,
7, 8,
     1, 9, 6, 5, 9, 2, 4, 2, 8, 3, 3, 9, 8, 9, 9, 4, 7, 6, 1, 6, 7, 8,
2, 2,
     7, 4, 4, 2, 8, 9, 8, 3, 1, 5, 1, 8, 3, 2, 2, 2, 3, 3, 3, 6, 9, 9,
5, 1,
     2, 1, 3, 1, 2, 8, 9, 4, 1, 6, 5, 9, 2, 3, 8, 6, 2, 7, 8, 9, 6, 9,
4, 9,
     7, 4, 6, 9, 1, 5, 7, 9, 8, 8, 4, 9, 5, 1, 1, 2, 9, 8, 2, 9, 8, 8,
6, 4,
     1, 4, 9, 2, 1, 7, 8, 8, 4, 8, 1, 2, 9, 3, 3, 4, 7, 4, 3, 1, 4, 8,
4, 6,
     8, 4, 6, 1, 9, 4, 6, 6, 1, 1, 2, 7, 8, 3, 9, 1]
check.within("Test 6: Random single digit", obey_benford(L),
             452.5259715426208, 0.00001)
L = [5803, 8827, 6712, 4049, 5435, 6449, 9051, 8972, 8622, 4555, 9391,
4022,
     1296, 6123, 9793, 3987, 7833, 5625, 3013, 1035, 1605, 5149, 873,
956,
     4104, 9372, 9661, 1013, 2849, 1934, 732, 6735, 7894, 9883, 2751,
6603,
     9197, 7346, 6007, 5034, 2713, 883, 3116, 6119, 4791, 2988, 1650,
1130,
     2505, 1737, 8514, 4274, 1655, 5689, 7368, 6697, 9022, 4431, 9874,
2767,
     4726, 2813, 6240, 4530, 2291, 2515, 9839, 1489, 5710, 5087, 3436,
1656,
     6213, 7111, 3528, 3248, 8948, 3265, 9709, 747, 1510, 9171, 2050,
6465,
     5839, 7989, 8654, 5998, 5302, 6680, 82, 2068, 9892, 1019, 3935,
5583,
     2704, 2175, 9674, 2895]
check.within("Test 7: Random up to 4 digits", obey_benford(L),
             45.96446343766565, 0.00001)
enron = [1] * 4000 + [2] * 1100 + [3] * 950 + [4] * 700 + [5] * 500 \
                + [6] * 650  + [7] * 450 + [8] * 950 + [9] * 950
check.within("Test 8: Simplified Enron courtesy WSJ",
             obey_benford(enron), 1713.8130681179023, 0.00001)



##**********************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 06 Problem 2
##**********************************************************************

import check
```

```
def le_rabot(n):
  '''
  Returns an application of le rabot to n
  (decreasing every digit run by 1)

  le_rabot: Nat -> Nat
  '''
  ans = ''
  s = str(n)
  for i in range(len(s)-1):
    if s[i] == s[i+1]:
      ans += s[i]
  if ans == "":
    return 0
  return int(ans)
```

```
def rabotez(L):
  '''
  Applies le rabot to each element in L
  (decreasing every digit run by 1)
  and mutates L after the application.

  Effects: Mutates L

  rabotez: (listof Nat) -> None

  Examples:
     L = []
     rabotez(L) => None
     and L is not mutated

     L = [1255511, 11111011111]
     rabotez(L) =>  None
     and L is mutated to [551, 11111111]
  '''
  for pos in range(len(L)):
    L[pos] = le_rabot(L[pos])


##Examples:
L = []
check.expect("Example 1",rabotez(L), None)
check.expect("Example 1 Mutation",L, [])


L = [1255511, 11111011111]
check.expect("Example 2",rabotez(L), None)
check.expect("Example 2 Mutation",L, [551, 11111111])
```

```
##Tests:
L = [1255511, 1111111111]
check.expect("Test all same digit",rabotez(L), None)
check.expect("Test all same digit Mutation",L, [551, 111111111])

L = [1234567890, 12121212121212]
check.expect("Test all different digits and alternating",rabotez(L),
None)
check.expect("Test all same digit Mutation",L, [0, 0])

L = [1]*100
check.expect("Test large single",rabotez(L), None)
check.expect("Test all same digit Mutation",L, [0]*100)

M = [111]*103
check.expect("Test large rep digit",rabotez(M), None)
check.expect("Test large rep digit Mutation",M, [11]*103)

M = [11111092311]
check.expect("Test single number",rabotez(M), None)
check.expect("Test single number Mutation",M, [11111])

M = [111110923117]
check.expect("Test single number last diff",rabotez(M), None)
check.expect("Test single number last diffMutation",M, [11111])

M = [11111092311, int(str(123)*100)]
check.expect("Test number large",rabotez(M), None)
check.expect("Test number large Mutation",M, [11111, 0])

##************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 06 Problem 3
##************************************************************

import check


SIZE = 3
player_prompt = "Player {0} enter a valid square: "
win_msg = "Player {0} wins!"
tie_msg = "Tie game"
invalid = "Invalid Input"
PLAYER1 = 'X'
PLAYER2 = 'O'

def print_board(board):
  '''
  Prints the tic-tac-toe board
```

```
Effects: Prints to screen

print_board: (listof (listof Str)) -> None
Requires:
   board is non-empty
   Each inner list of board is the same size as the outer list
   len(board[i]) <= 1 for all valid indices i.
'''
size = len(board)
for i in range(1, size * size + size):
  to_print = ""
  if i % (size + 1) == 0:
    to_print = ("---|"*size)
  elif i % (size + 1) == 2:
    row = i //(size+1)
    for entry in board[row]:
      to_print += " {0} |".format(entry)
  else:
    for entry in range(size):
      to_print += "   |"
  print(to_print[:-1])

def selected(square, board):
  '''
  Returns False if the square in the board is occupied
  and True otherwise.

  selected: Nat (listof (listof Str)) -> Bool
  Requires:
     Each inner list is the same size as the outer list
  '''
  row = (square-1) // SIZE
  col = (square-1) % SIZE
  return board[row][col] != " "

def fill_square(square, board, cur_player):
  '''
  Mutates the board so that the square in the board
  has player cur_player. Uses convention that 1 is top left
  and increases to the right and then downward.

  Effects: Mutates board

  fill_square: Nat (listof (listof Str)) Str -> Bool
  Requires:
     Each inner list is the same size as the outer list
  '''
  row = (square-1) // SIZE
  col = (square-1) % SIZE
```

```
  board[row][col] = cur_player


def swap_player(cur_player):
  '''
  Swaps players from 'X' to 'O' or sets player to be 'X'
  if non 'X' is passed

  swap_player: Str -> Str
  '''
  if cur_player == PLAYER1:
    cur_player = PLAYER2
  else:
    cur_player = PLAYER1
  return cur_player

def winner(board):
  '''
  Returns True if the tic-tac-toe board has a winner
  and False otherwise

  tied: (listof (listof Str)) -> Bool
  Requires:
     Each inner list is the same size as the outer list
     Each inner square is 'X', 'O' or ''.
  '''
  possibilities = [""]*(2*SIZE + 2)
  for i in range(SIZE):
    possibilities[-1] += board[i][-(i+1)].strip()
    possibilities[-2] += board[i][i].strip()
    for j in range(SIZE):
      possibilities[i] += board[i][j].strip()
      possibilities[i+SIZE] += board[j][i].strip()
  return "X"*SIZE in possibilities or "O"*SIZE in possibilities

def tied(board):
  '''
  Returns True if the tic-tac-toe board is completely filled
  and False otherwise

  tied: (listof (listof Str)) -> Bool
  Requires:
     Each inner list is the same size as the outer list
     Each inner square is 'X', 'O' or ''.
  '''
  all_squares = ""
  for i in range(SIZE):
    all_squares += "".join(board[i])
  return len(all_squares.replace(" ", "")) == SIZE * SIZE
```

```
def tic_tac_toe():
  '''
  Plays a game of tic-tac-toe

  Effects:
     Prints to screen
     Reads input form keyboard

  tic_tac_toe: None -> None
  '''
  board = []
  for i in range(SIZE):
    board.append([" "]*SIZE)
  cur_player = PLAYER2 #Start with P2 since swap first in loop.
  while not winner(board) and not tied(board):
    cur_player = swap_player(cur_player)
    print_board(board)
    square = input(player_prompt.format(cur_player))
    while (not square.isdecimal() or not (1 <= int(square) <=
SIZE*SIZE) or
           selected(int(square), board)):
      print(invalid)
      print_board(board)
      square = input(player_prompt.format(cur_player))
    fill_square(int(square), board, cur_player)

  print_board(board)  #Extra print board either at beginning or end.
  if winner(board):
    print(win_msg.format(cur_player))
  else:
    print(tie_msg)



##Examples:
check.set_screen("Test")
check.set_input("1", "4", "2", "5", "3")
check.expect("Example 1", tic_tac_toe(), None)

check.set_screen("Invalid then normal")
check.set_input("failed", "1", "4", "2", "5", "3")
check.expect("Example 1", tic_tac_toe(), None)


##Tests:
check.set_screen("Many Failed (9 total)")
check.set_input("failed", "0", "10", "3298573", "dsfdkon382598",
                "!#$*@#*%&", "1", "ASDFLJ", "4", "-12", "-1", "2",
"5", "3")
```

```
check.expect("Test Many Failed Win horizontal", tic_tac_toe(), None)

check.set_screen("5 boards")
check.set_input("1", "2", "4", "3", "7")
check.expect("Win Vertical", tic_tac_toe(), None)

check.set_screen("5 boards")
check.set_input("1", "2", "5", "3", "9")
check.expect("Win down left", tic_tac_toe(), None)

check.set_screen("5 boards")
check.set_input("3", "2", "5", "1", "7")
check.expect("Win down right", tic_tac_toe(), None)

check.set_screen("6 boards 0 wins")
check.set_input("6", "3", "2", "5", "1", "7")
check.expect("Win down right", tic_tac_toe(), None)

check.set_screen("Tie!")
check.set_input("1", "2", "3", "5", "4", "7", "6", "9", "8")
check.expect("Win down right", tic_tac_toe(), None)

check.set_screen("6 boards")
check.set_input("1", "2", "4", "5", "9", "8")
check.expect("Win Vertical 1", tic_tac_toe(), None)

check.set_screen("6 boards")
check.set_input("1", "3", "4", "6", "8", "9")
check.expect("Win Vertical 2", tic_tac_toe(), None)

check.set_screen("5 boards")
check.set_input("7", "3", "8", "6", "9")
check.expect("Win Horizontal 1", tic_tac_toe(), None)

check.set_screen("5 boards, one invalid same square")
check.set_input("4", "4", "3", "5", "9", "6")
check.expect("Win Horizontal 2", tic_tac_toe(), None)

check.set_screen("9 boards")
check.set_input("1", "2", "3", "4", "7", "6", "9", "8", "5")
check.expect("Win last piece", tic_tac_toe(), None)

check.set_screen("8 boards, invalid repeated square")
check.set_input("1", "2", "1", "3", "4", "7", "6", "9", "5")
check.expect("Win last piece", tic_tac_toe(), None)

##************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 06 Problem 4
```

```
##**************************************************************

import check


def diag_is_all_ones(M):
  '''
  Returns True if and only if all diagonal entries are 1.0 and False
otherwise

  diag_is_all_ones: (listof (listof Float)) -> Bool
  Requires: len(M) = len(M[0]) = ... = len(M[len(M)-1])
  '''
  #Note to self - needed for 0.0 case on diagonal.
  epsilon = 0.00001
  for i in range(len(M)):
    if abs(M[i][i] - 1.0) > epsilon:
      return False
  return True

def find_arbitrage(M):
  '''
  Returns True if and only if an arbitrage opportunity exists in
  a currency exchange matrix M.

  find_arbitrage: Currency_Exchange_Matrix -> Bool

  Examples:
  M = [[1.0, 0.76, 0.675675676],
       [1.31578947, 1.0, 0.89],
       [1.48, 1.1235955, 1.0]]
  find_arbitrage(M) => True

  M = [[1.0, 0.76, 0.675675676],
       [1.31578947, 1.0, 0.8890469],
       [1.48, 1.1248, 1.0]]
  find_arbitrage(M) => False
  '''
  epsilon = 0.00001
  if not diag_is_all_ones(M):
    return True
  n = len(M)
  for a in range(n):
    for b in range(n):
      for c in range(n):
        if abs(M[a][b]-M[a][c]*M[c][b]) >= epsilon:
          return True
  return False

##Examples:
```

```
M = [[1.0, 0.76, 0.675675676], [1.31578947, 1.0, 0.89],
     [1.48, 1.1235955, 1.0]]
check.expect("Example", find_arbitrage(M),  True)
M2 = [[1.0, 0.76, 0.675675676], [1.31578947, 1.0, 0.8890469],
      [1.48, 1.1248, 1.0]]
check.expect("Example", find_arbitrage(M2),  False)

##Tests:

M = [[1.01, 0.76, 0.675675676], [1.31578947, 1.0, 0.89], [1.48,
1.1235955, 1.0]]
check.expect("Test Diagonal not 1.0", find_arbitrage(M),  True)
M = [[1.0, 0.76, 0.675675676], [1.31578947, 1.01, 0.89], [1.48,
1.1235955, 1.0]]
check.expect("Test Diagonal not 1.0", find_arbitrage(M),  True)
M = [[1.0, 0.76, 0.675675676], [1.31578947, 1.0, 0.89], [1.48,
1.1235955, 1.01]]
check.expect("Test Diagonal not 1.0", find_arbitrage(M),  True)
M = [[1.0, 0.76, 0.675675676], [1.31578947, 1.0, 2.0], [1.48, 0.5,
1.0]]
check.expect("Test exists arbitrage", find_arbitrage(M),  True)
M = [[1.0, 0.1111111, 0.25, 0.3333333],
     [9.0, 1.0, 2.25, 3.0],
     [4.0, 0.4444444, 1.0, 99.0],
     [3.0, 0.3333333, 99.0, 1.0]]
check.expect("Test exists arbitrage", find_arbitrage(M),  True)
M = [[1.0, 0.1111111, 0.25, 0.3333333],
     [9.0, 1.0, 2.25, 3.0],
     [4.0, 0.4444444, 1.0, 1.3333333],
     [3.0, 0.3333333, 0.75, 1.0]]
check.expect("Test no arbitrage", find_arbitrage(M), False)

M = [[1.01]]
check.expect("Test Small Diagonal not 1.0", find_arbitrage(M),  True)

M = [[1.00]]
check.expect("Test Small Diagonal 1.0", find_arbitrage(M),  False)

M = [[0.0]]
check.expect("Test Small Diagonal not 1.0", find_arbitrage(M),  True)
```