```
##*****************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 04 Problem 1
##*****************************************************************

import check


def v_num(L, mean, pos):
  '''
  Returns the variance numerator

  v_num: (listof Float) Float Nat -> Float
  '''
  if pos >= len(L):
    return 0
  return (L[pos] - mean)**2 + v_num(L, mean, pos+1)


def variance(L):
  '''
  Returns the variance of a list of points using recursion

  variance_rec: (listof Float) -> Float
  Requires: len(L) > 0

  Examples:
     variance([2.0]) => 0.0
     variance([1.0, 2.0, 3.0, 4.0, 5.0]) => 2.0
  '''
  n = len(L)
  mean = sum(L)/n
  return v_num(L, mean, 0)/n

EPSILON = 0.00001

##Examples:
check.within("Ex1", variance([2.0]), 0.0, EPSILON)
check.within("Ex2", variance([1.0, 2.0, 3.0, 4.0, 5.0]), 2.0, EPSILON)

##Tests:
check.within("Test 1: Identical", variance([1.5, 1.5, 1.5, 1.5]), 0.0, EPSILON)
check.within("Test 2: Large", variance([0.0, 10000.0]), 25000000.0, EPSILON)
check.within("Test 3: Negative",
             variance([-10.25, 6, 4.25]), 53.04166666, EPSILON)




##*****************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 04 Problem 2
##*****************************************************************

import check


def variance_func(L):
  n = len(L)
  mean = sum(L)/n
  var = sum(list(map(lambda i: (L[i] - mean)**2, range(0,n))))/n
  return var

EPSILON = 0.00001
##Examples:
check.within("Ex1", variance([2.0]), 0.0, EPSILON)
check.within("Ex2", variance([1.0, 2.0, 3.0, 4.0, 5.0]), 2.0, EPSILON)
```

```
##Tests:
check.within("Test 1: Identical", variance([1.5, 1.5, 1.5, 1.5]), 0.0, EPSILON)
check.within("Test 2: Large", variance([0.0, 10000.0]), 25000000.0, EPSILON)
check.within("Test 3: Negative",
             variance([-10.25, 6, 4.25]), 53.04166666, EPSILON)
```

```
##################################################################
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 04 Problem 3
##################################################################

import check


def simulate_game(L, dirs):
  '''
  Mutates the cells in L in dirs so that zeroes and ones
  are flipped

  Effects: Mutates L

  simulate_game: (listof (listof Nat)) (listof (list Nat Nat)) -> None
  Requires:
     0 < len(L) == len(L[0]) == len(L[1]) == ... == len(L[len(L)-1])
     L[i][j] in [0, 1] is True for all integers i and j.

  '''
  if dirs != []:
    if 0 <= dirs[0][0] <= len(L)-1 and 0 <= dirs[0][1] <= len(L)-1:
      L[dirs[0][0]][dirs[0][1]] = (L[dirs[0][0]][dirs[0][1]] + 1) % 2
    simulate_game(L, dirs[1:])

def lights_out(L, row, col):
  '''
  Mutates L to simulate a button press of Lights Out.
  Pushes the button at (row, col) where rows cols
  start at 0 and increse downwards and to the right

  Effects: Mutates L

  lights_out: (listof (listof Nat)) Nat Nat -> None
  Requires:
     0 < len(L) == len(L[0]) == len(L[1]) == ... == len(L[len(L)-1])
     L[i][j] in [0, 1] is True  for all integers i and j.

  Examples:
     L = []
     lights_out(L, 0, 0) => None
     and L is not mutated

     L = [[1]]
     lights_out(L, 0, 1234) => None
     and L is not mutated

     L = [[1, 0, 0, 1, 0],
          [1, 0, 1, 1, 0],
          [1, 0, 0, 1, 1],
          [0, 0, 0, 0, 0],
          [1, 0, 0, 1, 0]]
```

```
    lights_out(L, 4, 1) => None
    and L is mutated to
    [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 1, 0, 0, 0],
     [0, 1, 1, 1, 0]]
  '''
  dirs = [[row, col], [row-1, col], [row+1,col], [row,col-1], [row,col+1]]
  simulate_game(L, dirs)

##Examples:

L = []
ans = []
check.expect("Example 1", lights_out(L, 0, 0), None)
check.expect("Example 1 Mutation", L, ans)

L = [[1]]
ans = [[1]]
check.expect("Example 2", lights_out(L, 0, 1234), None)
check.expect("Example 2 Mutation", L, ans)

L = [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 1, 0]]
ans =  [[1, 0, 0, 1, 0],
        [1, 0, 1, 1, 0],
        [1, 0, 0, 1, 1],
        [0, 1, 0, 0, 0],
        [0, 1, 1, 1, 0]]
check.expect("Example 3", lights_out(L, 4, 1), None)
check.expect("Example 3 Mutation", L, ans)

#Tests

L = [[1, 0, 1, 0],
     [1, 0, 1, 0],
     [0, 0, 0, 0],
     [1, 1, 1, 1]]
ans =  [[0, 1, 1, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 0],
        [1, 1, 1, 1]]
check.expect("Test corner 1", lights_out(L, 0, 0), None)
check.expect("Test corner 1 Mutation", L, ans)

L = [[1, 0, 1, 0],
     [1, 0, 1, 0],
     [0, 0, 0, 0],
     [1, 1, 1, 1]]
ans =  [[1, 0, 0, 1],
        [1, 0, 1, 1],
        [0, 0, 0, 0],
        [1, 1, 1, 1]]
check.expect("Test corner 2", lights_out(L, 0, 3), None)
check.expect("Test corner 2 Mutation", L, ans)

L = [[1, 0, 1, 0],
     [1, 0, 1, 0],
     [0, 0, 0, 0],
     [1, 1, 1, 1]]
ans =  [[1, 0, 1, 0],
        [1, 0, 1, 0],
        [1, 0, 0, 0],
        [0, 0, 1, 1]]
check.expect("Test corner 3", lights_out(L, 3, 0), None)
check.expect("Test corner 3 Mutation", L, ans)
```

```
L = [[1, 0, 1, 0],
     [1, 0, 1, 0],
     [0, 0, 0, 0],
     [1, 1, 1, 1]]
ans =  [[1, 0, 1, 0],
        [1, 0, 1, 0],
        [0, 0, 0, 1],
        [1, 1, 0, 0]]
check.expect("Test corner 4", lights_out(L, 3, 3), None)
check.expect("Test corner 4 Mutation", L, ans)


L = [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 1, 0]]
ans =  [[1, 0, 0, 1, 0],
        [0, 0, 1, 1, 0],
        [0, 1, 0, 1, 1],
        [1, 0, 0, 0, 0],
        [1, 0, 0, 1, 0]]
check.expect("Test Left side", lights_out(L, 2, 0), None)
check.expect("Test Left side Mutation", L, ans)

L = [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 1, 0]]
ans =  [[1, 1, 1, 0, 0],
        [1, 0, 0, 1, 0],
        [1, 0, 0, 1, 1],
        [0, 0, 0, 0, 0],
        [1, 0, 0, 1, 0]]
check.expect("Test Top side", lights_out(L, 0, 2), None)
check.expect("Test Top side Mutation", L, ans)


L = [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 1, 0]]
ans =  [[1, 0, 0, 1, 0],
        [1, 0, 1, 1, 0],
        [1, 0, 0, 1, 1],
        [0, 0, 1, 0, 0],
        [1, 1, 1, 0, 0]]
check.expect("Test Bottom side", lights_out(L, 4, 2), None)
check.expect("Test Bottom side Mutation", L, ans)

L = [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 1, 0]]
ans =  [[1, 0, 0, 1, 0],
        [1, 0, 1, 1, 1],
        [1, 0, 0, 0, 0],
        [0, 0, 0, 0, 1],
        [1, 0, 0, 1, 0]]
check.expect("Test Right side", lights_out(L, 2, 4), None)
check.expect("Test Right side Mutation", L, ans)

L = [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 1, 0]]
ans =  [[1, 0, 0, 1, 0],
```

```
        [1, 0, 0, 1, 0],
        [1, 1, 1, 0, 1],
        [0, 0, 1, 0, 0],
        [1, 0, 0, 1, 0]]
check.expect("Test Middle", lights_out(L, 2, 2), None)
check.expect("Test Middle Mutation", L, ans)

L = [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 1, 0]]
ans =  [[1, 0, 0, 1, 0],
        [1, 0, 1, 1, 0],
        [1, 0, 0, 1, 1],
        [0, 0, 0, 0, 0],
        [1, 0, 0, 1, 0]]
check.expect("Test Out of range", lights_out(L, 22134, 21342), None)
check.expect("Test Out of Range Mutation", L, ans)

L = [[1, 0, 0, 1, 0],
     [1, 0, 1, 1, 0],
     [1, 0, 0, 1, 1],
     [0, 0, 0, 0, 0],
     [1, 0, 0, 1, 0]]
ans =  [[1, 0, 0, 1, 0],
        [1, 0, 1, 1, 0],
        [1, 0, 0, 1, 1],
        [0, 0, 0, 0, 0],
        [1, 0, 0, 1, 0]]
check.expect("Test Just Out of range", lights_out(L, 5, 5), None)
check.expect("Test Just Out of Range Mutation", L, ans)

L = list(map(list, [[0]*100]*100))
ans = list(map(list, [[0]*100]*100))
ans[56][23] = 1
ans[56][22] = 1
ans[56][24] = 1
ans[55][23] = 1
ans[57][23] = 1
check.expect("Test Large", lights_out(L, 56, 23), None)
check.expect("Test Large Mutation", L, ans)
```

```
##*****************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 04 Problem 4
##*****************************************************************

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.

Examples:
   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 = list(map(lambda s: int(str(s)[0]), L))
   counts = list(map(lambda d: digits.count(d), range(1, 10)))
   exs = list(map(lambda d: math.log((d+1)/d, 10), range(1, 10)))
   n = len(L)
   chi_squared = sum(list(map(
     lambda d: (counts[d] - n*exs[d])**2/(n*exs[d]), range(9))))

   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), 3.322195322272341, 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), 0.0023622171891271 43, 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), 0.4299765547162358, 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), 2.3219280948873626, 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)
```