```
##
##=============================================================
##  J.P. Pretti (instructor)
##  CS 116 Winter 2021
##  Assignment 04, Problem 1
##=============================================================
##


import check
import math


## A Polynomial is a Str
## Requires:
##  It represents an integer polynomial and is
##  formatted as described in the question.

## A Term is a Str
## Requires:
##  It represents a term and is formatted as
##  described in the question.


def eval_term(term, val):
  '''
  Returns the value of term at val.

  eval_term: Term Int -> Int
  '''
  if "x" not in term:         # constant term
    return int(term)
  elif term == "x":           # term is x
    return val
  elif not "^" in term:       # general linear term
    return int(term[:-1])*val
  elif term[:2] == "x^":      # coefficient of one
    return val ** int(term[2:])
  else:                       # general case
    coeff = term.split("x^")[0]
    exp = term.split("x^")[1]
    return int(coeff) * (val ** int(exp))


def eval_list(L,val):
  '''
  Returns the value of an integer polynomial at val. The terms of the polynomial
  are precisely the entries in L.

  eval_list: (listof Term) Int -> Int
  '''
  if L == []:
    return 0
  else:
    return eval_term(L[0],val) + eval_list(L[1:],val)


def eval_poly(poly, val):
  '''
  Returns the value of poly at val.

  eval_poly: Polynomial Int -> Int

  Examples:
```

```
    eval_poly("7",116) => 7
    eval_poly("8x",5) => 40
    eval_poly("3x^4+x^3+6x+2", 2) => 70
  '''
  terms = poly.split("+")
  return eval_list(terms,val)

## Examples as Tests:
check.expect("Example 1", eval_poly("7",116),7)
check.expect("Example 2", eval_poly("8x",5),40)
check.expect("Example 3",eval_poly("3x^4+x^3+6x+2", 2),70)

## Other Tests:
check.expect("Just a Constant", eval_poly("116",116),116)
check.expect("Just x", eval_poly("x",116),116)
check.expect("Just Linear Term and Negative Value", eval_poly("116x",-2),-232)
check.expect("Just a High Order Term", eval_poly("4x^3",7), 1372)
check.expect("Quadratic. No Special Cases", eval_poly("2x^2+2x+2",0),2)
check.expect("Big Coefficients", eval_poly("10000x^7+100x^5+8",1), 10108)
check.expect("Big Exponents", eval_poly("9x^116116+9x^116",1),18)
check.expect("Big Coefficients and Exponents", eval_poly("9x^116+116x",1),125)
big_poly = "7x^8+3x^7+2x^5+x^4+x^3+100x+75"
check.expect("Many Terms",eval_poly(big_poly,3),53457)##



##=============================================================
##  J.P. Pretti (instructor)
##  CS 116 Winter 2021
##  Assignment 04, Problem 2
##=============================================================
##


import check


## Global Constant
max_ends = 99

## Constants for Examples and Tests
sample_rock_positions = ["041Y1","209R1","389Y1","596R1",\
                  "045Y2","198R2","376R2","380R2","415R2","606R2",\
                  "043Y3","071Y3","108R3","400Y3","402Y3",\
                  "042Y4","075Y4","076Y4","422R4"]
sample_rock_positions_scrambled = \
 ['198R2', '402Y3', '376R2', '071Y3', '596R1', '209R1', '606R2', '389Y1', \
  '042Y4', '045Y2', '108R3', '076Y4', '380R2', '043Y3', '415R2', '400Y3', \
  '041Y1', '422R4', '075Y4']

def end_score(colour,rock_positions,end):
  '''
  Returns the score of team colour in the end numbered end and given by the
  positions of rocks represented by rock_positions.

  score: Str (listof Str) Int -> Nat

  Requires:
    1<= end <= 99
    colour is "Y" or "R"
    Entries in rock_positions are as specified on the assignment.

  '''
  end_rock_posns = list(filter(lambda s : s.endswith(str(end)), rock_positions))
```

```
  end_rock_posns.sort()
  colours_in_order = list(map(lambda s : s[3], end_rock_posns))
  if colour == 'R':
    if 'Y' not in colours_in_order:
      num = len(colours_in_order)
    else:
      num = colours_in_order.index('Y')
  else:
    if 'R' not in colours_in_order:
      num = len(colours_in_order)
    else:
      num = colours_in_order.index('R')
  return num


def score(colour, rock_positions):
  '''
  Returns the score of team colour given by the positions of rocks
  represented by rock_positions.

  score: Str (listof Str) -> Nat

  Requires:
    colour is "Y" or "R"
    Entries in rock_positions are as specified on the assignment.

  Examples:
    score("R",[]) => 0
    score("Y",sample_rock_positions) => 7
  '''
  L = list(map(lambda i : end_score(colour,rock_positions,i),
               range(1,max_ends+1)))
  return sum(L)

## Examples as Tests:

check.expect("Example 1",score("R",[]),0)
check.expect("Example 2",score("Y",sample_rock_positions),7)

## Other Tests:

check.expect("List of Length 1",score("R",["100R2"]),1)
check.expect("Example 2 but R Team",score("R",sample_rock_positions),0)
check.expect("All One Colour", \
             score("Y",["001Y1","002Y1","001Y2","001Y3"]),4)
check.expect("Example 2 but Scrambled", \
             score("Y",sample_rock_positions_scrambled),7)
check.expect("Scattered Blank Ends", \
             score("Y",["001Y1","002R1","001R4","001Y7","002R7"]),2)
check.expect("One end and one rock.", score("R",["001R1"]), 1)
check.expect("One end and one rock.", score("R",["001Y1"]), 0)
check.expect("One end and one rock.", score("Y",["001R1"]), 0)
check.expect("One end and one rock.", score("Y",["001Y1"]), 1)
check.expect("All 99 Ends", score("Y",\
             list(map(lambda i : "001R"+str(i), range(1,51))) + \
             list(map(lambda i : "001Y"+str(i), range(51,100))), \
             49)




##
##=============================================================
##  J.P. Pretti (instructor)
```

```
## CS 116 Winter 2021
## Assignment 04, Problem 3
##============================================================
##


import check


## A Board is a (listof (listof Str))
## Requires:
##   The length of the outer lists is 10.
##   The length of each inner list is 10.
##   Each string is '.', 'L', or 'S'.


## Constants for Examples and Tests
sample_snakes = [[1,20,21,40],[78,64],[50]]
sample_ladders = [[99,82,79,62,59],[47,53,69]]
sample_board = \
[['.', 'L', '.', '.', '.', '.', '.', '.', '.', '.'], \
 ['.', 'L', '.', '.', '.', '.', '.', '.', '.', '.'], \
 ['.', 'L', 'S', '.', '.', '.', '.', '.', '.', '.'], \
 ['.', 'L', '.', 'S', '.', '.', '.', '.', 'L', '.'], \
 ['.', 'L', '.', '.', '.', '.', '.', 'L', '.', '.'], \
 ['.', '.', '.', '.', '.', '.', 'L', '.', '.', 'S'], \
 ['S', '.', '.', '.', '.', '.', '.', '.', '.', '.'], \
 ['S', '.', '.', '.', '.', '.', '.', '.', '.', '.'], \
 ['S', '.', '.', '.', '.', '.', '.', '.', '.', '.'], \
 ['S', '.', '.', '.', '.', '.', '.', '.', '.', '.']]
all_cells = list(map(lambda i : list(range(10*i+1,10*i+11)),range(10)))


def empty_board():
    '''
    Returns a list of lists representing a Snakes and Ladders 10 by 10 board where
    each entry is a period.

    empty_board: None -> Board
    '''
    return list(map(lambda ignore : ["."]*10, range(10)))


def row(n):
    '''
    Returns the row number (between 0 and 9 inclusive) containing n
    in the game of Snakes and Ladders.

    row: Int -> Nat

    Requires: 1 <= n <= 100
    '''
    return 9 - (((n-1) // 10) % 10)


def col(n):
    '''
    Returns the column number (between 0 and 9 inclusive) containing n
    in the game of Snakes and Ladders.

    row: Int -> Nat

    Requires: 1 <= n <= 100
    '''
```

```
257   if ((n-1) // 10) % 2 == 0:
258     return ((n-1) % 10)
259   else:
260     return 9 - ((n-1) % 10)
261
262
263 def add_chain(B,char,chain):
264   '''
265   Mutates B by changing each cell corresponding to a number in chain to char.
266
267   Effects:
268     Mutates B
269
270   add_chain: Board Str (listof (listof Int)) -> None
271
272   Requires:
273     1 <= x <= 100 for all entries in chain
274       and
275     the number of occurrences of x in chain is at most one
276   '''
277   if chain != []:
278     B[row(chain[0])][col(chain[0])] = char
279     add_chain(B,char,chain[1:])
280
281
282 def add_chains(B,char,chains):
283   '''
284   Mutates B by changing each cell corresponding to a number in a sublist of
285   chains to char.
286
287   Effects:
288     Mutates B
289
290   add_chains: Board Str (listof (listof Int)) -> None
291
292   Requires:
293     1 <= x <= 100 for all entries in sublists of chains
294       and
295     the number of occurrences of x across all sublists is at most one
296   '''
297   if chains != []:
298     add_chain(B, char, chains[0])
299     add_chains(B, char, chains[1:])
300
301
302 def make_board(snakes,ladders,board):
303   '''
304   Mutates board so that "S" (a snake) is in cells determined by the numbers in
305   snakes and "L" (a ladder) is in cells determined by ladders. Other cells
306   remain unchanged.
307
308   Effects:
309     Mutates board.
310
311   make_board: (listof (listof Int)) (listof (listof Int)) Board -> None
312
313   Requires:
314     1 <= x <= 100 for all entries in sublists of snakes and ladders
315       and
316    the number of occurrences of x across both lists is at most one
317
318   Examples:
319     if board == empty_board(), then
320     make_board([],[],board) => None
```

```
    and board is unchanged

    if board == empty_board(), then
    make_board(sample_snakes,sample_ladders,board) => None
    and board is changed to be sample_board
  '''
  add_chains(board,"S",snakes)
  add_chains(board,"L",ladders)

## Examples as Tests:

B = empty_board()
copy_of_B = B[:]
check.expect("No snakes or ladders", make_board([],[],B), None)
check.expect("No snakes or ladders (checking B)",B,copy_of_B)

B = empty_board()
check.expect("Given example", make_board(sample_snakes,sample_ladders,B), None)
check.expect("Given example (checking B)",B,sample_board)

## Other Tests:

B = empty_board()
sample_snakes = [[]] + sample_snakes + [[]]
sample_ladders = [[]] + sample_ladders + [[]]
check.expect("Empty nested lists", \
          make_board(sample_snakes,sample_ladders,B), None)
check.expect("Empty nested lists (checking B)",B,sample_board)

B = empty_board()
copy_of_B = B[:]
copy_of_B[9][1] = "S"
check.expect("One snake", make_board([[2]],[],B), None)
check.expect("One snake (checking B)",B,copy_of_B)

B = empty_board()
copy_of_B = B[:]
copy_of_B[5][5] = "L"
check.expect("One ladder", make_board([],[[46]],B), None)
check.expect("One ladder (checking B)",B,copy_of_B)

B = empty_board()
check.expect("All snakes", make_board(all_cells,[],B), None)
check.expect("All snakes (checking B)",B,[["S"]*10]*10)

B = empty_board()
check.expect("All ladders", make_board([],all_cells,B), None)
check.expect("All ladders (checking B)",B,[["L"]*10]*10)
```