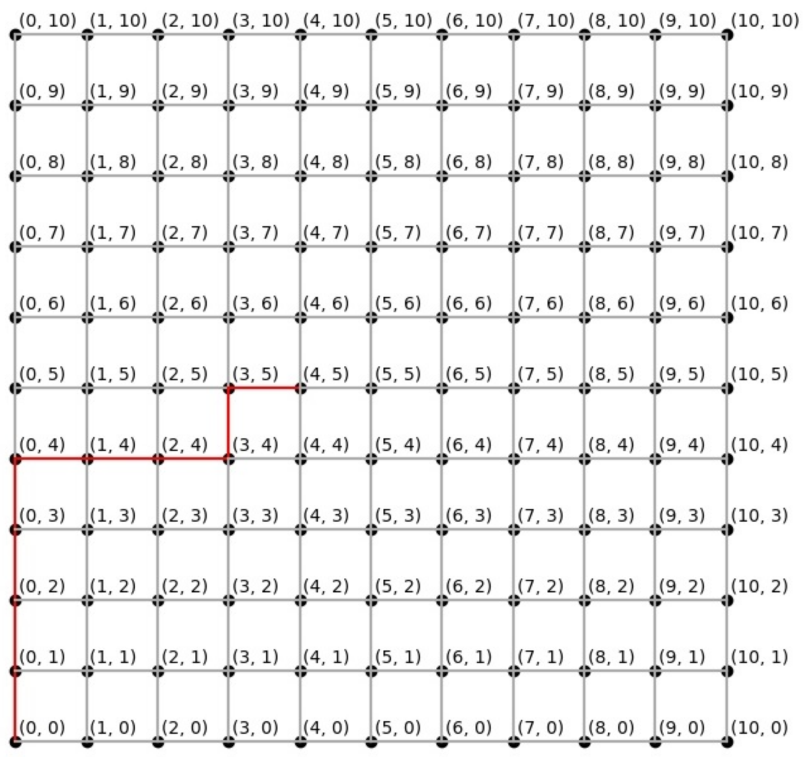
Week-8 Practice Programming

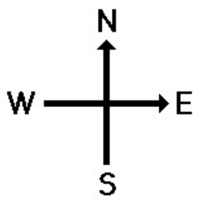
Problem 1

The below image represents a grid having 11 x 11 nodes numbered from 0 to 10.

- Distance between one node to next connected node is 1 unit.
- One can go in any direction, each letter counts as 1 unit in respective direction.
    - `N` North
    - `S` South
    - `E` East
    - `W` West

The below graph shows the path for `NNNNEEENE` starting from (0, 0).





Question

Write a python program to take a string as input from user and print the total distance traveled.

Answer

```python
p = input()
print(p.count('N') + p.count('E') + p.count('S') + p.count('W') ) # count every step and print it
```

Testcases

Public

Input	Output
`NNNNEEENE`	`9`
`NEWS`	`4`

Private

Input	Output
`NEEWWWSSEWEWEWS`	`15`
`EEEEEEEEEEEEE`	`13`
`N`	`1`

Tags

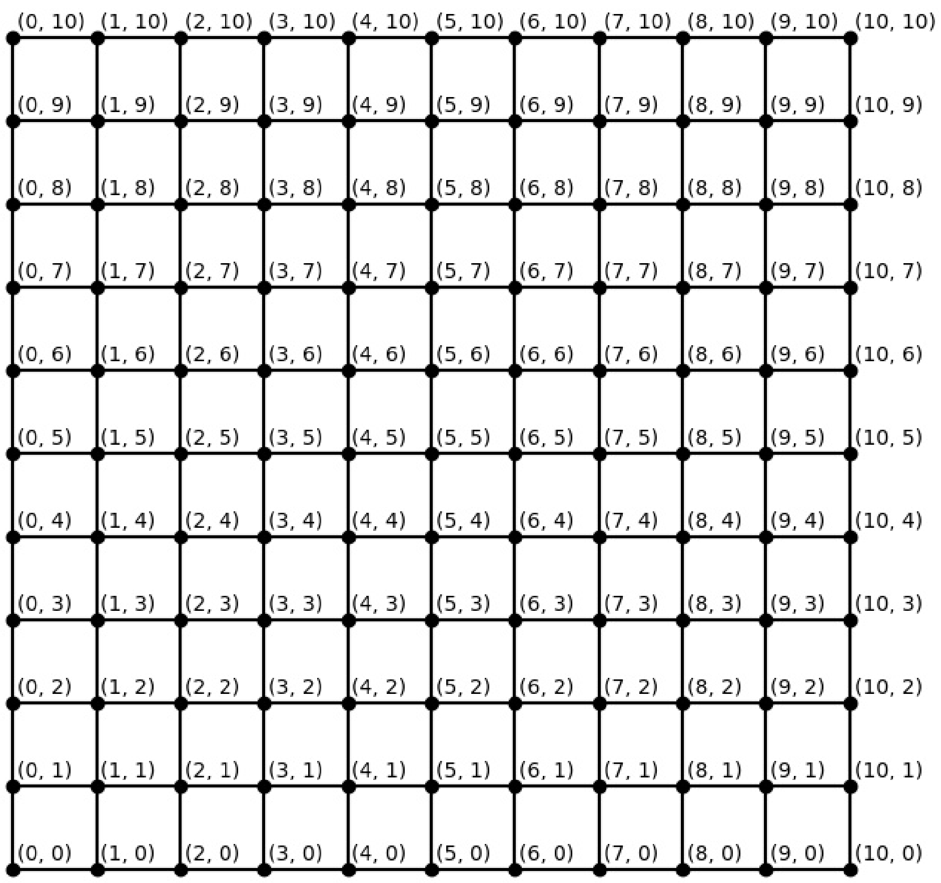
TAGS SEPERATED BY COMMAS

Comments

COMMENTS IN TEXT/`CODE`

Problem 2

The below image represents a grid having 11 x 11 nodes numbered from 0 to 10.



Question

A block is a closed loop of distinct adjacent connected nodes.

> (0, 0), (1, 0), (1, 1) and (0, 1) will form a block.

> (0, 0), (1, 0), (1, 1) and (1, 2) will not form a block.

Write a function `isBlock` to take a list of four nodes as tuples and check whether they form block or not. Return True if they form a block and False otherwise.

- The function is only required.
- No need to take any input or print any output.

Answer

```python
def isBlock(l):
    l.sort() # sorted in order of Bottom-Left, Top-Left Bottom-Right, and Top-Right node
    b = 0
    if (l[0][0],   l[0][1]+1) == l[1]: # Checking of Bottom-Left and Top-Left
        b += 1
    if (l[0][0]+1, l[0][1]   ) == l[2]: # Checking of Bottom-Left and Botton-Right
        b += 1
    if (l[0][0]+1, l[0][1]+1) == l[3]: # Checking of Bottom-Left and Top-Right
        b += 1
    return b == 3
```

Suffix (Hidden)

```python
# suffix
import ast
def parse(inp):
  inp = ast.literal_eval(inp)
  return inp

fncall = input()
lparen = fncall.find("(")
rparen = fncall.rfind(")")
fname = fncall[:lparen]
farg = fncall[lparen+1:rparen]

if fname == "isBlock":
   arg = parse(farg)
   print(isBlock(arg))
else:
   print("Function", fname, "unknown")
```

Testcases

Public

Input	Output
`isBlock([(0, 0), (1, 0), (1, 1), (0, 1)])`	`True`
`isBlock([(0, 0), (1, 0), (1, 1), (1, 4)])`	`False`

Private

Input	Output
`isBlock([(0, 0), (1, 0), (1, 1), (1, 0)])`	`False`
`isBlock([(2, 1), (3, 1), (3, 2), (2, 2)])`	`True`
`isBlock([(5, 5), (6, 6), (5, 6), (6, 5)])`	`True`
`isBlock([(1, 1), (1, 1), (1, 1), (1, 1)])`	`False`

Tags

TAGS SEPERATED BY COMMAS

Comments

COMMENTS IN TEXT/`CODE`

Problem 3

Question

Write a Python program for a ticket reservation system. The operational details are given below.

- Available tickets are 100
- Booking time starts at 10:00 and closes at 17:00 ends inclusive.
- Any booking that falls outside the booking time should be rejected.
- One person can book tickets for multiple persons, hence that reservation should be completely reserved or completely rejected.
- Assume that multiple bookings do not happen at the same time.
- `HH:MM S 10 C 25 W 20 O 5` is an example line from the input.
    - `'S'`, `'C'`, `'W'` and `'O'` are the identifiers that denote Senior citizen, Child, Woman and Others respectively.
    - The line always starts with `HH:MM` which are the time in 24 hours.
    - `S 10` denotes the 10 tickets of Senior citizens, similarly for other identifiers.
    - The ticket type and number can be in any order, where the number is always followed by the identifier. The same example is valid and is equivalent to `HH:MM C 25 S 10 O 5 W 20`.
- Update the dictionary `log` accordingly. Refer the suffix part of the code.
- You don't need to print anything.
- The last line of input will be an empty line.

Answer

```python
availableTickets = 100 # variable to store the available tickets
log = {} # initilization of log
for i in ['S', 'C', 'W', 'O']: # creating a key of all type of tickets
    log[i] = 0
line = ' '
while line:
    line = input().strip() # read input
    if availableTickets > 0: # continue if there is atleast one ticket
        t = line[:5] # time
        if '10:00' <= t <= '17:00': # checking for the time limit
            tk = line[5:].strip().split() #  ticket type and numbers in list
            d = {} # dictionary for storing ticket type and numbers of a line
            for i in range(0, len(tk), 2): # storing the ticket type and number
                d[tk[i].strip()] = int(tk[i+1])
            tkCount = 0 # variable to store total tickets in the line
            for k in d:
                tkCount += d[k] # total tickets in the line
            if tkCount <= availableTickets: # is enough tickets available
                availableTickets -= tkCount # reduce the number of tickets from available tickets
                for k in d: # insert in to log
                    log[k] += d[k]
```

Suffix Visible

```python
sold = 0
for i in log:
    sold += log[i]
print(f'Tickets sold: {sold}')
print(f'Tickets remaining: {100-sold}')
print(f'Senior citizens: {log["S"]}')
print(f'Children: {log["C"]}')
print(f'Women: {log["W"]}')
print(f'Other: {log["O"]}')
```

Testcases

Public

Input	Output
`09:15 S 1 C 1 O 1`	`Tickets sold: 92`
`10:00 S 12 O 21`	`Tickets remaining: 8`
`12:00 W 19`	`Senior citizens: 12`
`13:01 O 12`	`Children: 1`
`14:15 O 15 W 12 C 1`	`Women: 31`
	`Other: 48`
`09:15 S 1 C 1 O 1`	
`10:00 S 12 O 21`	`Tickets sold: 97`
`12:00 W 19`	`Tickets remaining: 3`
`13:01 O 12`	`Senior citizens: 12`
`14:15 O 15 W 12 C 1`	`Children: 1`
`14:16 O 25`	`Women: 31`
`16:17 O 5`	`Other: 53`

Private

Input	Output

`12:00 w 19`
`13:01 o 12`
`14:15 o 15 w 12 c 1`
`14:16 o 25`
`16:17 o 5`
	`Tickets sold: 89`
	`Tickets remaining: 11`
	`Senior citizens: 0`
	`Children: 1`
	`Women: 31`
	`Other: 57`

`09:15 s 1 c 1 o 1`
`10:00 s 12 o 21`
`12:00 w 19`
`13:01 o 12`
`14:15 o 15 w 12 c 1`
`14:16 o 25`
`16:17 o 5`
`17:00 w 3`
	`Tickets sold: 100`
	`Tickets remaining: 0`
	`Senior citizens: 12`
	`Children: 1`
	`Women: 34`
	`Other: 53`

`09:15 s 1 c 1 o 1`
`10:00 s 12 o 21`
`12:00 w 19`
`13:01 o 12`
`14:15 o 15 w 12 c 1`
`14:16 o 25`
`16:17 o 5`
`17:00 w 2`
`17:01 o 1`
	`Tickets sold: 99`
	`Tickets remaining: 1`
	`Senior citizens: 12`
	`Children: 1`
	`Women: 33`
	`Other: 53`

Tags

TAGS SEPERATED BY COMMAS

Comments

COMMENTS IN TEXT/`CODE`

Problem 4

Question

Write a function `findMe`, it returns `True` if a * is present anywhere in the nested list and `False` otherwise.

Answer

```python
def findMe(l):
    found = False # variable to store wheter the '*' is found
    for i in l: # iterating through each list elements of passed list into the function
        if i == '*': # if '*' is found return True
            return True
        if type(i) == type([]): # if the element type is list then search for '*' in that list before moving on to the next element
            found = found or findMe(i) # found will become True if atleast one '*' is preseent inside the list
    return found
```

Suffix Hidden

```python
# Suffix
import ast

def parse(inp):
  inp = ast.literal_eval(inp)
  return inp

fncall = input()
lparen = fncall.find("(")
rparen = fncall.rfind(")")
fname = fncall[:lparen]
farg = fncall[lparen+1:rparen]

if fname == "findMe":
   arg = parse(farg)
   print(findMe(arg))
else:
   print("Function", fname, "unknown")
```

Testcases

Public

Input	Output
`findMe([['a', 'z'], ['b'], ['c'], [['d'], [[5], ['*'], []], [], ['j']]])`	`True`
`findMe([['a', 'z'], ['b'], ['c'], [['d'], [[5], [], []], [], ['j']]])`	`False`
`findMe([['a', 'z'], ['b'], ['c'], [['d'], [[5], [], []], [], ['j']], '*'])`	`True`

Private

Input	Output
`findMe([[1,2,3,],[1,2,'*']])`	`True`
`findMe([*])`	`True`
`findMe([[],[],[],[[],[],[],[],[],[],[[[[[[[]]]]]]]]])`	`False`
`findMe([[],[],[],[[],[],[],[],[],[],[[[[[[['*']]]]]]]]])`	`True`
`findMe([[1,2,3,],[1,2]])`	`False`
`findMe([[1,2,3,],,[[[[[]]]]],[1,2]])`	`False`

Tags

TAGS SEPERATED BY COMMAS

Comments

COMMENTS IN TEXT/`CODE`

Problem 5

Question

Write a function `listToDict` to convert a nested list (two level) into dictionary where the keys of the dictionary be the index of the nested list.

- Let `l` be the passed nested list and `d` be the returned dictionary, where `l[i][j] == d[i][j]` should return True for all valid index / key `i` and `j`.

Write a `dictToList` to convert a dictionary (two level) into nested list where the index of the list be the keys of the dictionary.

- Let `d` be the passed dictionary and `l` be the returned nested list, where `d[i][j] == l[i][j]` should return True for all valid index / key `i` and `j`.
- The keys of the dictionaries are always an integer from 0 to 100.
- 0 is placed in the position where the there is no key.
- The size of the list should be in a ways that accommodate all the keys in the dictionary as list index.

Answer

```python
def listToDict(l):
    d = {} # intializing the dictionary to be returned
    for i in range(len(l)): # iterating through the rows
        d[i] = {} # inner dictionary initialization
        for j in range(len(l[i])): # iterating through the column of ith row
            d[i][j] = l[i][j] # storing the list element of index [i][j] to dictionary of key [i][j]
    return d

def dictToList(d):
    l = [] # intializing the list to be returned
    maxKeys1 = max(d.keys()) # length of outer list/row is the maximum of the key in outer dictionary
    maxKeys2 = 0 # length of inner list/column is the maximum of the key in inner dictionary
    for i in d:
        for j in d[i]:
            if j > maxKeys2: #finding the maximum of the key in inner dictionary
                maxKeys2 = j
    for i in range(maxKeys1+1): # iteraing for number of rows required
        l.append([]) # appending and empty list for each column
        for j in range(maxKeys2+1): # iteraing for number of rows required
            if i in d.keys() and j in d[i].keys():  # append to list element [i][j] if i and j are in the keys of outer and inner dictionary respectively
                l[-1].append(d[i][j])
            else: # otherwise zero is appended
                l[-1].append(0)
    return l
```

Suffix Hidden

```python
import ast

def parse(inp):
  inp = ast.literal_eval(inp)
  return inp

fncall = input()
lparen = fncall.find("(")
rparen = fncall.rfind(")")
fname = fncall[:lparen]
farg = fncall[lparen+1:rparen]

if fname == "listToDict":
   arg = parse(farg)
   print(listToDict(arg))
elif fname == "dictToList":
   arg = parse(farg)
   print(dictToList(arg))
else:
   print("Function", fname, "unknown")
```

Testcases

Public

Input	Output
`listToDict([[1], [1,2]])`	`{0: {0: 1}, 1: {0: 1, 1: 2}}`
`dictToList({1:{5:1, 2:9}, 3:{0:1}})`	`[[0, 0, 0, 0, 0, 0], [0, 0, 9, 0, 0, 1], [0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0]]`

Private

Input	Output
`listToDict([[1,2,2,2,2,2,2], [1,2,2]])`	`{0: {0: 1, 1: 2, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2}, 1: {0: 1, 1: 2, 2: 2}}`
`listToDict([[1,2,3]])`	`{0: {0: 1, 1: 2, 2: 3}}`
`listToDict([[1,2],[3,4]])`	`{0: {0: 1, 1: 2}, 1: {0: 3, 1: 4}}`
`dictToList({10:{1:0},1:{3:12}})`	`[[0, 0, 0, 0], [0, 0, 0, 12], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]`
`dictToList({10:{0:0}})`	`[[0], [0], [0], [0], [0], [0], [0], [0], [0], [0], [0]]`
`dictToList({0:{0:0}})`	`[[0]]`

Tags

Comments