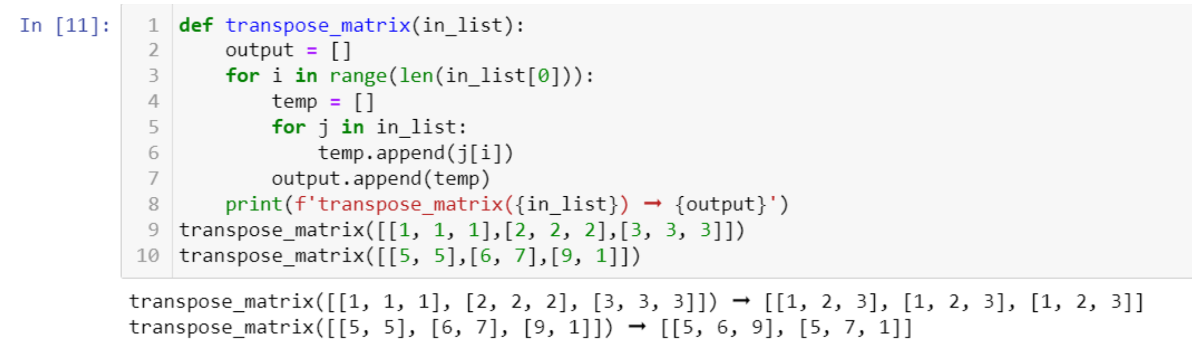
1. Create a function that transposes a 2D matrix.

Examples:

transpose_matrix([
  [1, 1, 1],
  [2, 2, 2],
  [3, 3, 3]
]) ➞ [
  [1, 2, 3],
  [1, 2, 3],
  [1, 2, 3]
]

transpose_matrix([
  [5, 5],
  [6, 7],
  [9, 1]
]) ➞ [
  [5, 6, 9],
  [5, 7, 1]
]

Ans:



```
In [11]:
def transpose_matrix(in_list):
    output = []
    for i in range(len(in_list[0])):
        temp = []
        for j in in_list:
            temp.append(j[i])
        output.append(temp)
    print(f'transpose_matrix({in_list}) ➞ {output}')
transpose_matrix([[1, 1, 1],[2, 2, 2],[3, 3, 3]])
transpose_matrix([[5, 5],[6, 7],[9, 1]])
```

```
transpose_matrix([[1, 1, 1], [2, 2, 2], [3, 3, 3]]) ➞ [[1, 2, 3], [1, 2, 3], [1, 2, 3]]
transpose_matrix([[5, 5], [6, 7], [9, 1]]) ➞ [[5, 6, 9], [5, 7, 1]]
```

2. Create a function that determines whether a string is a valid hex code.

A hex code must begin with a pound key # and is exactly 6 characters in length. Each character must be a digit from 0-9 or an alphabetic character from A-F. All alphabetic characters may be uppercase or lowercase.

Examples:

is_valid_hex_code("#CD5C5C") ➞ True

is_valid_hex_code("#EAECEE") ➞ True

is_valid_hex_code("#eaecee") ➞ True

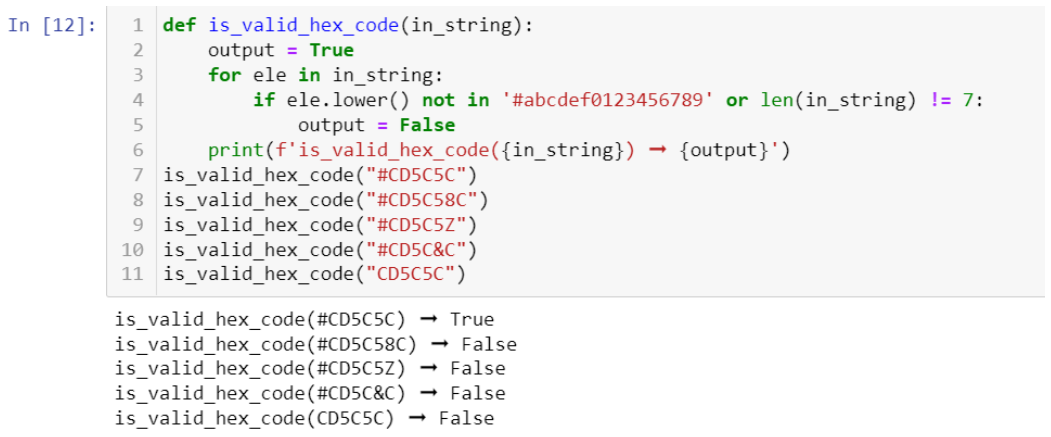
is_valid_hex_code("#CD5C58C") ➞ False
# Length exceeds 6

is_valid_hex_code("#CD5C5Z") ➞ False
# Not all alphabetic characters in A-F

is_valid_hex_code("#CD5C&C") ➞ False
# Contains unacceptable character

is_valid_hex_code("CD5C5C") ➞ False
# Missing #

Ans:

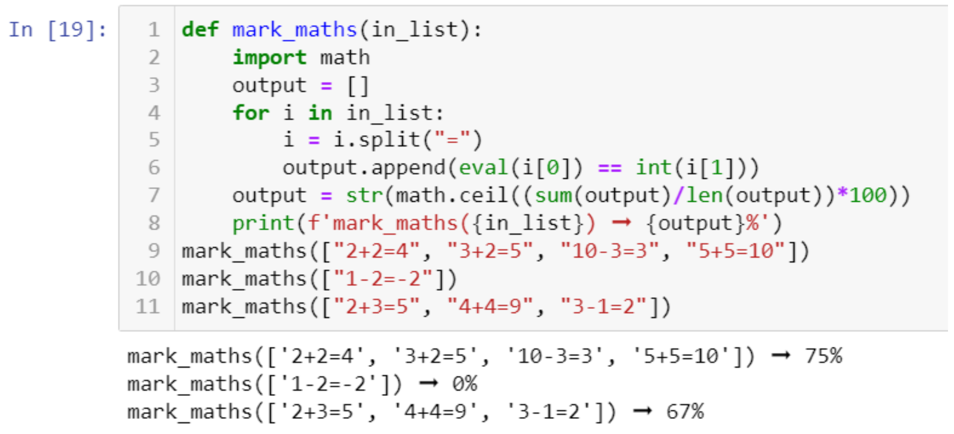


```
In [12]:
def is_valid_hex_code(in_string):
    output = True
    for ele in in_string:
        if ele.lower() not in '#abcdef0123456789' or len(in_string) != 7:
            output = False
    print(f'is_valid_hex_code({in_string}) ➞ {output}')
is_valid_hex_code("#CD5C5C")
is_valid_hex_code("#CD5C58C")
is_valid_hex_code("#CD5C5Z")
is_valid_hex_code("#CD5C&C")
is_valid_hex_code("CD5C5C")
```

```
is_valid_hex_code(#CD5C5C) ➞ True
is_valid_hex_code(#CD5C58C) ➞ False
is_valid_hex_code(#CD5C5Z) ➞ False
is_valid_hex_code(#CD5C&C) ➞ False
is_valid_hex_code(CD5C5C) ➞ False
```

3. Given a list of math equations (given as strings), return the percentage of correct answers as a string. Round to the nearest whole number.

Examples:
mark_maths(["2+2=4", "3+2=5", "10-3=3", "5+5=10"]) ➞ "75%"
mark_maths(["1-2=-2"]), "0%"
mark_maths(["2+3=5", "4+4=9", "3-1=2"]) ➞ "67%"
Ans:



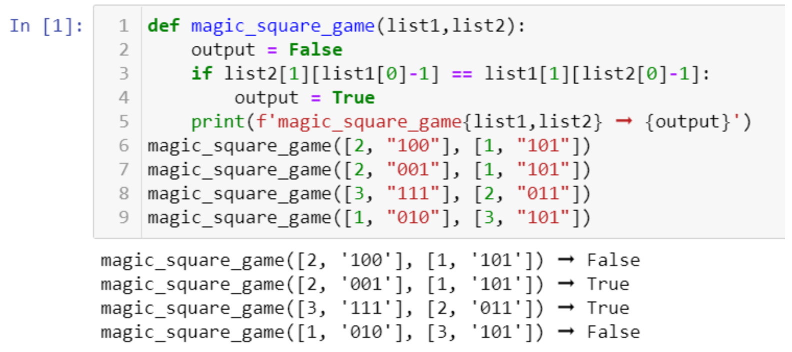
```
In [19]:
def mark_maths(in_list):
    import math
    output = []
    for i in in_list:
        i = i.split("=")
        output.append(eval(i[0]) == int(i[1]))
    output = str(math.ceil((sum(output)/len(output))*100))
    print(f'mark_maths({in_list}) ➞ {output}%')
mark_maths(["2+2=4", "3+2=5", "10-3=3", "5+5=10"])
mark_maths(["1-2=-2"])
mark_maths(["2+3=5", "4+4=9", "3-1=2"])
```

```
mark_maths(['2+2=4', '3+2=5', '10-3=3', '5+5=10']) ➞ 75%
mark_maths(['1-2=-2']) ➞ 0%
mark_maths(['2+3=5', '4+4=9', '3-1=2']) ➞ 67%
```

4. There are two players, Alice and Bob, each with a 3-by-3 grid. A referee tells Alice to fill out one particular row in the grid (say the second row) by putting either a 1 or a 0 in each box, such that the sum of the numbers in that row is odd. The referee tells Bob to fill out one column in the grid (say the first column) by putting either a 1 or a 0 in each box, such that the sum of the numbers in that column is even.

Alice and Bob win the game if Alice's numbers give an odd sum, Bob's give an even sum, and (most important) they've each written down the same number in the one square where their row and column intersect.

Examples:
magic_square_game([2, "100"], [1, "101"]) ➞ False
magic_square_game([2, "001"], [1, "101"]) ➞ True
magic_square_game([3, "111"], [2, "011"]) ➞ True
magic_square_game([1, "010"], [3, "101"]) ➞ False
# Two lists, Alice [row, "her choice"], Bob [column, "his choice"]
Ans:



```
In [1]:
def magic_square_game(list1,list2):
    output = False
    if list2[1][list1[0]-1] == list1[1][list2[0]-1]:
        output = True
    print(f'magic_square_game{list1,list2} ➞ {output}')
magic_square_game([2, "100"], [1, "101"])
magic_square_game([2, "001"], [1, "101"])
magic_square_game([3, "111"], [2, "011"])
magic_square_game([1, "010"], [3, "101"])
```

```
magic_square_game([2, '100'], [1, '101']) ➞ False
magic_square_game([2, '001'], [1, '101']) ➞ True
magic_square_game([3, '111'], [2, '011']) ➞ True
magic_square_game([1, '010'], [3, '101']) ➞ False
```

5. From point A, an object is moving towards point B at constant velocity va (in km/hr). From point B, another object is moving towards point A at constant velocity vb (in km/hr). Knowing this and the distance between point A and B (in km), write a function that returns how much time passes until both objects meet.

Format the output like this: "2h 23min 34s"

Examples:
lets_meet(100, 10, 30) ➞ "2h 30min 0s"
lets_meet(280, 70, 80) ➞ "1h 52min 0s"
lets_meet(90, 75, 65) ➞ "0h 38min 34s"

Ans:

In [2]:

```python
def lets_meet(distance,va,vb):
    import math
    time = distance/(va+vb)
    hrs = math.floor(time)
    mins = math.floor((time-hrs)*60)
    secs = math.floor(((((time)-hrs)*60)-mins)*60)
    print(f'lets_meet{distance,va,vb} ➞ {hrs}h {mins}min {secs}s')
lets_meet(100, 10, 30)
lets_meet(280, 70, 80)
lets_meet(90, 75, 65)
```

```
lets_meet(100, 10, 30) ➞ 2h 30min 0s
lets_meet(280, 70, 80) ➞ 1h 52min 0s
lets_meet(90, 75, 65) ➞ 0h 38min 34s
```