Problem 1
Write a function `sum_diag_UL_LR(grid, offset)` that takes as arguments a grid of numbers and an offset and returns the result of summing the numbers on a specified diagonal of grid. This function considers diagonals running from the upper-left of the grid to the lower-right (hence the 'UL_LR' in the function name). The offset is used to select which diagonal to sum, as shown on the right; the grid shown in this figure is represented in the program as a list of lists:

```python
[[11, 22, 33, 44, 55],
 [66, 77, 88, 99, 11],
 [22, 33, 44, 55, 66],
 [77, 88, 99, 11, 22],
 [33, 44, 55, 66, 77]]
```

Your program can assume that the argument `grid` is in fact a grid (i.e., a list of equal-length lists of numbers), but should be able to handle grids of any size.
Problem 2
Write a function `sum_diag UR_LL(grid, offset)` that takes as arguments a grid of numbers and an offset and returns the result of summing the numbers on a specified diagonal of grid. This function considers diagonals running from the upper-right of the grid to the lower-left (hence the 'UR_LL' in the function name). The offset is used to select which diagonal to sum, as shown on the right. The grid is represented as a list of lists, as in Problem 1. Your code can assume that the input is in fact a grid, but should be able to handle grids of any size.
SOLUTIONS

Problem 1

There are two things that have to be worked out for this problem: (1) the coordinates for the beginning of the diagonal for a given offset; and (2) the change in x- and y-coordinate values between successive elements of any diagonal.

We can figure out the first quantity by just examining some different offset values; the key is to notice that negative offsets have to be treated differently than positive offsets.

For any given diagonal, we can figure out the change in x- and y-coordinates by stepping through successive elements of a diagonal a few times.

Once these two things are done, programming up the solution is straightforward.

```python
def sum_diag_UL_LR(grid, offset):
    sum = 0
    if offset > 0:
        i, j = 0, offset
    else:
        i, j = -offset, 0

    while i < len(grid) and j < len(grid):  # both indexes within bounds
        sum += grid[i][j]
        i, j = i+1, j+1

    return sum
```
SOLUTIONS
Problem 2
While this problem is fundamentally similar to the previous one, it is slightly more complex because the iteration moves through increasing values along one of the coordinates but decreasing values along the other coordinate. So the change in the x- and y-coordinate values has to be worked out carefully. But other than this, the problem is essentially very similar to Problem 1.

By the way, notice the chained comparison operation in the while loop:

\[ 0 \leq i < \text{len}(x) \]

In Python, this is equivalent to the and of the two comparison operations, but the middle expression (in this case, the variable i) is evaluated only once. Not many other languages offer this kind of syntax.

```python
def sum_diag_UR_LL(grid, offset):
    sum = 0
    if offset > 0:
        i, j = 0, (len(grid)-1)-offset
    else:
        i, j = -offset, (len(grid)-1)
    while 0 <= i < len(grid) and 0 <= j < len(grid):
        sum += grid[i][j]
        i, j = i+1, j-1
    return sum
```