Daily Coding Problem #63

Problem

This problem was asked by Microsoft.

Given a 2D matrix of characters and a target word, write a function that returns whether the word can be found in the matrix by going left-to-right, or up-to-down.

For example, given the following matrix:

```
[['F', 'A', 'C', 'I'],
 ['O', 'B', 'Q', 'P'],
 ['A', 'N', 'O', 'B'],
 ['M', 'A', 'S', 'S']]
```

and the target word 'FOAM', you should return true, since it's the leftmost column. Similarly, given the target word 'MASS',

you should return true, since it's the last row.

Solution

This problem should be quite straightforward: we can go through each entry in the array, try to create the word going right and down, and check if the word matches our word. To make bounds checking simple, we'll just try to grab everything from where we're looking at till the end, and then use the slice operator to just get what we want.

```
def build_word_right(matrix, r, c, length):
    return ''.join([matrix[r][i] for i in range(c, len(matrix[0]))])[:length]

def build_word_down(matrix, r, c, length):
    return ''.join([matrix[i][c] for i in range(r, len(matrix))])[:length]

def word_search(matrix, word):
    for r in range(len(matrix)):
        for c in range(len(matrix[0])):
            word_right = build_word_right(matrix, r, c, len(word))
            word_down = build_word_down(matrix, r, c, len(word))
            if word in (word_right, word_down):
                return True
    return False
```

However, if the matrix was really big, then we would be grabbing the whole row or column just to shorten it. We can improve our `build_word_right` and `build_word_down` functions a bit by just taking what we need, which is whichever is shorter between the length of the word and the end of the row or column:

```
def build_word_right(matrix, r, c, length):
    row_len = len(matrix[0])
    return ''.join([matrix[r][i] for i in range(c, min(row_len, length))])
```

```
def build_word_down(matrix, r, c, length):
    col_len = len(matrix)
    return ''.join([matrix[i][c] for i in range(r, min(col_len, length))])
```

However, let's say our word were both really big. If we notice, when we're checking the current row or column, that the first few letters are off, then we can quit the search early.

The Python built-in function `zip` is very handy:

```
def check_word_right(matrix, r, c, word):
    word_len = len(word)
    row_len = len(matrix[0])
    if word_len != row_len - c:
        return False
    for c1, c2 in zip(word, (matrix[r][i] for i in range(c, row_len))):
        if c1 != c2:
            return False
    return True

def check_word_down(matrix, r, c, word):
    word_len = len(word)
    col_len = len(matrix)
    if word_len != col_len - r:
        return False
    for c1, c2 in zip(word, (matrix[i][c] for i in range(r, col_len))):
        if c1 != c2:
            return False
    return True

    return ''.join([matrix[i][c] for i in range(r, min(col_len, length))])
```

```
def word_search(matrix, word):
    for r, row in enumerate(matrix):
        for c, val in enumerate(row):
            if check_word_right(matrix, r, c, word):
                return True
            if check_word_down(matrix, r, c, word):
                return True
    return False
```