Diffy

General information

Iterate both positions and values of sequence types

The built-in function `enumerate` can be used to request an iterator for a given sequence type (strings, lists, tuples, files, ...) that both returns the position and the value at that position for the next element of the sequence type. The example below illustrates how this can be used to simultaneously iterate the positions of a string and the characters on those position.

```
>>> for index, character in enumerate('abc'):
...     print("index: {}".format(index))
...     print("character: {}".format(character))
...
index: 0
character: a
index: 1
character: b
index: 2
character: c
```

You can also use this to simultaneously iterate over the characters of two strings.

```
>>> first = 'abc'
>>> second = 'def'
>>> for index, character in enumerate(first):
...    print("{}-{}".format(character, second[index]))
...
a-d
b-e
c-f
```

However, in this case it is better to use the built-in function `zip`, which is especially equipped to iterate over multiple iterable objects at once.

```
>>> first = 'abc'
>>> second = 'def'
>>> for character1, character2 in zip(first, second):
...    print("{}-{}".format(character1, character2))
...
a-d
b-e
c-f
```

Remarks

One element tuples

One-element tuples look like:

```
1,
```

The essential part of the notation here is the trailing comma. As for any expression, parentheses are optional, so you may write one-element tuples like:

```
(1, )
```

But it is the comma, not the parentheses, that define the tuple. Take a look at the following example:

```
>>> sequence = (3)              # an integer
>>> reeks
3
>>> isinstance(reeks, tuple)
False
>>> isinstance(reeks, int)
True
>>> reeks = (3, )               # a tuple
>>> isinstance(reeks, tuple)
True
```

The comma is essential, because Python otherwise interprets the notation `(object)` as the object itself.

Repeating the elements of a tuple

Just as strings, tuples can also be multiplied with a positive integer. This creates a new tuple that contains a repetition of the elements in the original tuple.

```
>>> tuple = (2, 3)
>>> tuple * 4
(2, 3, 2, 3, 2, 3, 2, 3)
>>> element = (0, )
>>> 6 * element
(0, 0, 0, 0, 0, 0)
```

A square triangle

Specific information

In the Python code, Pascal's triangle is represented as a list of lists. However, the hexagonal visualisation in the description of the assignment does not properly align all elements that are in the same column. It is easier to think about this assignment, if you do consider a representation of Pascal's triangle where this is the case. Take a close look at the following figure, where the visual representation from the problem description of the assignment is converted to a representation as a list of lists.

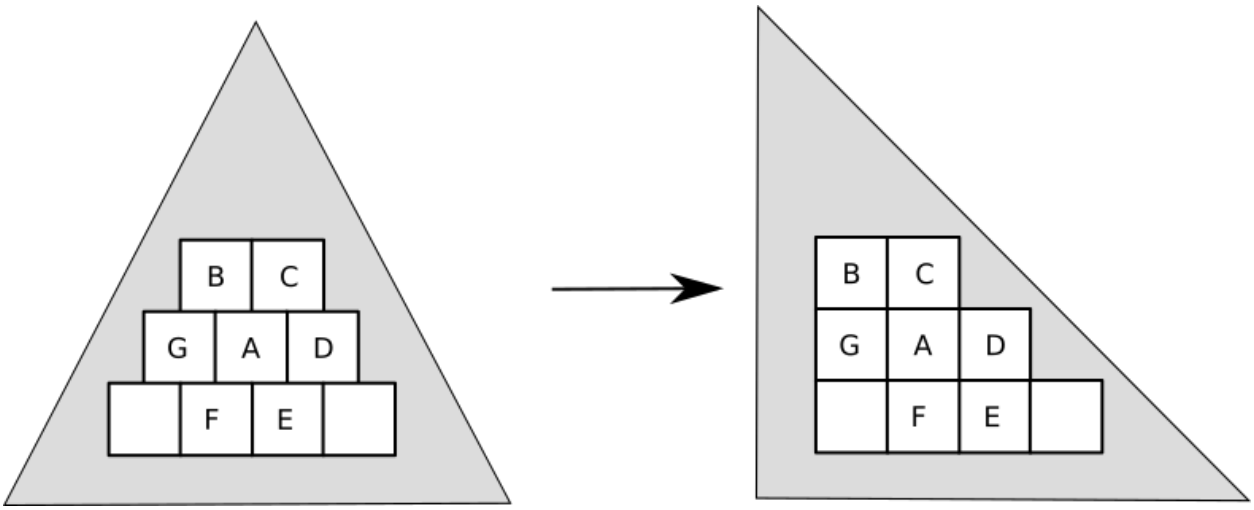

Figure 1: Pascal's triangle

Pozo Azul

General information

Initialize nested list

A rectangular grid with m rows and n columns can be represented by a list `l` containing m lists, with each list from `l` having n elements. Say, for example, that you want to create a list with three rows and two columns, with each cell of the grid containing the empty string. This can be done using *list comprehensions*.

```
>>> m = 3
>>> n = 2
>>> grid = [['' for _ in range(n)] for _ in range(m)]
>>> grid
[['', ''], ['', ''], ['', '']]
>>> grid[0][0] = 'A'
>>> grid
[['A', ''], ['', ''], ['', '']]
```

In most cases you can also use the multiplication operator `*` to create lists of a given size, where each cell is initialized with the same object. But if this object is mutable, you might get surprising results.

```
>>> m = 3
>>> n = 2
>>> [''] * m
['', '', '']
>>> grid = [[''] * n] * m
>>> grid
[['', ''], ['', ''], ['', '']]
>>> grid[0][0] = 'A'
>>> grid
[['A', ''], ['A', ''], ['A', '']]
```

As you can see, putting the string `A` in cell `grid[0][0]`, causes the string `A` to be placed as well in cells `grid[1][0]` and `grid[2][0]`. The reason for this strange behaviour is that the operator `*` has created aliases

to the same list, such that `grid[0]`, `grid[1]` and `grid[2]` all reference exactly the same list object. As a result, adjusting one of these lists, changes all of them (well, actually, there is only one of them). This can be seen clearly when you use the Python Tutor.

Specific information

In this assignment it will definitely help if you define two helper functions that each take a cardinal direction. The first function returns the opposite direction of the given cardinal direction. The second function returns two integers that indicate the change in row and column index of you move to the neighbouring square of the cave as indicated by the given direction.

In addition, there are two possible ways a corridor of the cave may be blocked:

- the corridor in the next square is not connected with the cardinal direction where you enter the square; if you enter a square from the west, the corridor in that square must be connected with the east side of the square
- the corridor in the current square hits the edge of the cave, such that you would leave the cave if you would take the next step