Star Drawings

In this exercise, we’ll create two-dimensional ”star drawings” to practice using functions, files, and lists.

Part (a): Consider the following function definition:

```python
def draw_line(line_spec):
    result = ''
    star_turn = False
    for num in line_spec:
        if star_turn:
            result = result + num * '*'
        else:
            result = result + num * ' '  
        star_turn = not star_turn
    print(result)
```

Trace as much of the execution for the following calls of `draw_line` as needed to understand what the function does. Show what is printed in the grid below each call (assume each square represents a space):

```
draw_line([0, 3, 2, 3])
```

```
draw_line([1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
draw_line([0, 1, 1, 2, 1, 3, 1, 2, 1, 1])
```

Next, download and run the program `starDrawings_0.py`.

Q: Why does the shell just return without printing anything when you run the program?

The `test_draw_line` function was written to test the `draw_line` function. Call `test_draw_line` in the shell (console) to check that your answers in part (a) were correct.
Part (b): In this part of the exercise, you will create star drawings by reading specifications from a file for the lines that are to be drawn.

Download `starDrawingsFromFile.py`, `drawCup.txt`, and `drawSat.txt` from the Artifacts section of our website; put them in the same working directory.

View the two text files in a browser or an editor (use something that shows the new lines). Each line of each file is either empty or contains a sequence of space-separated integers. You will finish the stubs for the functions in `starDrawingsFromFile.py` to create star drawings from files of this form.

Read the documentation in `starDrawingsFromFile.py` and then replace the stub (`pass` instruction) for each function, in order, testing each before moving on to code the next.

After implementing `main`, test it using the two text files provided.
**Part(c):** The **draw_line** function can also be used in defining functions for drawing a variety of common shapes. Consider the following example:

```python
def draw_vee(height):
    
    """draws a vee of the given height (a pos int)""
    for i in range(0, height-1):
        lspec = [i, 1, 2*(height-i)-3, 1]
        draw_line(lspec)
    draw_line([height-1, 1])
```

To understand how it works, answer the following questions about the call `draw_vee(4)`.

1. What are the values of `i` and of `lspec` for each iteration of the loop?
   - on the 1\textsuperscript{st} iteration: \( i \) is _____ and \( lspec \) is _________________
   - on the 2\textsuperscript{nd} iteration: \( i \) is _____ and \( lspec \) is _________________
   - on the 3\textsuperscript{rd} iteration: \( i \) is _____ and \( lspec \) is _________________

2. Given the values for `lspec` that you computed above and knowing what `draw_line` does, show what `draw_vee(4)` should print on the grid below (assume each square represents a space).

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Download **starDrawings.py**, which contains definitions of all the functions discussed so far. (It also contains “stubs” for functions that you will implement in part (d) of this exercise. You can ignore these for now.)

   In the shell, run a test to see if your drawing is correct. Run a few other tests to be certain you understand how `draw_vee` works.
**Part (d):** Now try your hand at writing the following functions. They should exhibit the behavior described by the docstrings in the file `starDrawings.py`. You will replace the `pass` statement with code. Do them in the order indicated below. **Do not cut and paste code from `draw_line` into your function definitions; instead, call `draw_line` with a line specification.**

`draw_right_parallelogram( w, h )` should print a parallelogram that slants to the right, where the values of the parameters indicate the width (`w`) and height (`h`). For example:

```
In [2]: draw_right_parallelogram(4, 5)
       ****
       ****
       ****
       ****
       ****

In [3]: draw_right_parallelogram(1, 3)
    *
    *
    *

In [4]: draw_right_parallelogram(8, 3)
       ********
       ********
       ********
```

`draw_left_parallelogram( w, h )` should print a parallelogram of a given width (`w`) and height (`h`) that slants to the left. For example:

```
In [5]: draw_left_parallelogram(8, 3)
       ********
       ********
       ********

In [6]: draw_left_parallelogram(5, 4)
       *****
       *****
       *****
       *****
```
draw_parallelogram( w, h, d ) should print a parallelogram of a given width (w) and height (h) that slants or not depending on the value of the third parameter (d): if negative, the parallelogram should slant to the left; if positive, the parallelogram should slant to the right; if 0, the parallelogram should not slant. By default, the value for d should be 0.

Be smart: Do NOT cut and paste the code from your other functions into the body of draw_parallelogram. Call the appropriate functions to print the slanted parallelograms.

```python
In [2]: draw_parallelogram(3, 4, 1)
   ***
   ***
   ***

In [3]: draw_parallelogram(4, 3, -1)
   ****
   ****

In [4]: draw_parallelogram(4, 3)
   ****
   ****
   ****
```

Part (e): Reverse engineer your library of functions so that each has a last parameter, named left_pad, which indicates the number of blank spaces to leave before drawing the corresponding figure. The default value for left_pad should be 0.

```python
In [16]: draw_parallelogram(2, 3, 1)
   **
   **
   **

In [17]: draw_parallelogram(2, 3, 1, 5)
   **
   **

In [18]: draw_parallelogram(2, 3, left_pad=2)
   **
   **
```
**Part (f):** Try your hand at writing functions for printing the shapes below. Design your functions to keep code cloning to a minimum.

<table>
<thead>
<tr>
<th>*</th>
<th>***********</th>
<th>*</th>
<th>***********</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>***</td>
<td>***********</td>
<td>**</td>
<td>***********</td>
<td>***</td>
</tr>
<tr>
<td>****</td>
<td>***********</td>
<td>***</td>
<td>***********</td>
<td>****</td>
</tr>
<tr>
<td>***********</td>
<td>****</td>
<td>****</td>
<td>*</td>
<td>***********</td>
</tr>
<tr>
<td>********</td>
<td>***</td>
<td>***</td>
<td>***</td>
<td>***********</td>
</tr>
<tr>
<td>***********</td>
<td>**</td>
<td>**</td>
<td>*****</td>
<td>***</td>
</tr>
<tr>
<td>**********</td>
<td>*</td>
<td>*</td>
<td>***********</td>
<td>*</td>
</tr>
</tbody>
</table>