Timestamp 2/6/2017

Website: https://www.cs.hmc.edu/twiki/bin/view/CS5/ASCIIArtGold

CS5 Web > Homework8Gold > ASCIIArtGold
Next HW: **Homework 2: *Functioning recursively*** will be due on: **Mon., Feb. 6, 11:59pm**
Next Lab: **Lab 2: Turtle!!** will be held on: Tue./Wed. evening, Jan. 31-Feb. 1
Submissions: CS submission site

Gold Hw8 Problem 4: ASCII Art

up to +8 e.c. points; individual or pair (even more for the crazy diamond...)

Place all of the code for this problem in a file called `hw8pr4.py`.

In this assignment you will revisit a classic art form: ASCII art!

Important limitation! For these problems, you should **not** use Python's string-multiplication or string-addition operators.

Because our goal is to use loop constructs, use loops to achieve the repetition that those operators might otherwise provide. There is one exception, however — you ***may*** use string-multiplication with the space character `' '`. That is, you can create any number of consecutive spaces with constructs like

```
' '*n
```

ASCII art problems

The goal of this problem is to solidify further your reasoning skills with loops, and nested loops. For many of the problems (especially the striped diamond), you will have to think carefully about the value of your loop control variable as your loop or loops execute. "Debugging by random permutation" — that is, arbitrarily changing the values of your loop conditions or variables — will lead to much frustration. The path to success on this assignment is to reason carefully about your loops.

There are five problems here, the first two are worth 1 point and the other three are worth 2 points each.

The `printCrazyStripedDiamond` is worth a possible +5 points beyond the others. It is, however, *crazy*...

printRect

Write a function named `printRect` that takes three arguments, `width`, `height`, and `symbol`, and prints a `width` by `height` rectangle of `symbols` on the screen.

```
In [1]: printRect( 4, 6, '%' )
% % % %
% % % %
% % % %
% % % %
% % % %
% % % %
```

Hint: If you look back at the slides from the first day of nested loops (10/27/15), you'll see that we did precisely this problem! The only differences are that

- the width is a variable, instead of a constant
- the height is a variable, instead of a constant
- the character printed is a variable, instead of a constant

printTriangle

Create a function `printTriangle` that takes three arguments: `width`, `symbol`, and `rightSideUp` and prints a triangle of symbols on the screen. `width` is a number that determines the width of the base of the triangle and `rightSideUp` is a boolean that determines whether the triangle is printed right side up (`True`) or upside down (`False`).

```
In [1]: printTriangle( 3, '@', True )
@
@ @
@ @ @

In [2]: printTriangle( 3, '@', False )
@ @ @
@ @
@
```

printBumps

Now, use your printTriangle function to write a function called `printBumps( num, symbol1, symbol2 )` that will print the specified number of two-symbol "bumps", where each bump is larger than the last, as in the following example:

```
In [1]: printBumps( 4, '%', '#' )
%
#
%
% %
# #
#
%
% %
% % %
# # #
# #
#
%
% %
% % %
% % % %
# # # #
# # #
# #
#
```

printDiamond

For these "diamond" functions, you **may** use string multiplication, but only for strings of blank spaces, such as `' '*n` or the like. Each visible character should be printed separately, just as in the functions earlier in this problem. Also, you don't *have* to use the string `*` operator for strings of spaces, either.

Write a function called `printDiamond( width, symbol )` that prints a diamond of `symbol` whose maximum width is determined by `width`.

```
In [1]: printDiamond( 3, '&' )
   &
  & &
 & & &
  & &
   &
```

printStripedDiamond

Next, write a function called `printStripedDiamond( width, sym1, sym2)` that prints a "striped diamond" of sym1 and sym2.

For example:

```
In [1]: printStripedDiamond( 7, '.', '%' )
       .
      . %
     . % .
    . % . %
   . % . % .
  . % . % . %
. % . % . % .
 % . % . % .
  . % . % .
   % . % .
    . % .
     % .
      .
```

printCrazyStripedDiamond

Finally, (and this is worth +5 points *beyond* the +8 for the previous figures...) write a function called `printCrazyStripedDiamond( width, sym1, sym2, sym1Width, sym2Width)` that prints a "striped diamond" of sym1 and sym2 where the stripes can have varied widths. `sym1Width` determines the width of the stripe made of symbol 1 and `sym2Width` determines the width of the stripe made of symbol 2.

For example:

```
In [1]: printCrazyStripedDiamond( 7, '.', '%', 2, 1 )
       .
      . .
     . . %
    . . % .
   . . % . .
  . . % . . %
. . % . . % .
 . % . . % .
  % . . % .
   . . % .
    . % .
```

```
% .
.
```

Be sure to submit your problem to the usual place as `hw8pr4.py`.