CSc 110, Autumn 2017
Programming Assignment #2: ASCII Art (16 points)
Due Tuesday, September 5, 2017, 7:00 PM

Thanks to Stuart Reges of the University of Washington, Marty Stepp of Stanford and Susan Rodger of Duke

This assignment focuses on `for` loops, expressions, variables, constants and parameters. Turn in two Python files as described below.

Part A: ASCII Art (2 points):

The first part of your assignment is to write a program that produces any text art (sometimes called "ASCII art") picture you like. Write a Python program in a file named `ascii_art.py`. We will share everyone's art on the class website. Your program can produce any text picture you like, with the following restrictions and details:

- The picture should be your own creation, not an ASCII image you found on the Internet or elsewhere.
- The number of lines drawn should be at least 3 but no more than 200, and no more than 200 characters / line.
- The picture should not include hateful, offensive, or otherwise inappropriate images.
- The code should use at least one `for` loop or function, but should not contain infinite loops.
- The picture must not be identical to your solution for Part B or consist entirely of reused Part B code.
- Your code should not use material beyond lecture 8.
- If your Part A program runs successfully and meets the above constraints, it will receive the full 2 points. Part A will not be graded on style ("internal correctness").

```
       \\\\////
        |.)(.|
        | || |
        \(__)/
        678876
        >>><<<
        \\\\\\
        (0)(0)
        | /\ |
        ( () )
        567765
        |-..-|
        |o\/o|
   .----\    /----.
  / / / |~~~~| \ \ \
 / / / /|::::|\ \ \ \
'-'-'-'-|::::|-'-'-'-'
       (((^^)))
        456654
        >>><<<
        //////
        |.)(.|
        | || |
        \(__)/
        345543
        /\--/\
       ( .  . )
        234432
        |-..-|
        |o\/o|
       (\    /)
      ( |vvvv| )
      ( |vvvv| )
         T  T
        123321
        >>><<<
        ||||||
        (o)(o)
        | /\ |
        (====)
        |_/\_|
        (_/\_)
       _|_,__|_
      (___\____)
```

Part B: Totem Pole (14 points):

The second part of your assignment is to produce a specific text figure that is supposed to look like a totem pole. Turn in a file named `totem_pole.py`. You should **exactly** reproduce the format of the output at left. This includes having identical characters and spacing.

One way to write a Python program to draw this figure would be to write a single `print` statement that prints each line of the figure. However, this solution would not receive full credit. A major part of this assignment is showing that you understand `for` loops, expressions and parameters.

In lines that have repeated patterns of characters that vary in number from line to line, represent the lines and character patterns using `for` loops and string multiplication. It may help to write pseudocode and tables to understand the patterns, as described in section and lecture.

Another significant component of this assignment is the task of generalizing the program using a single constant that can be changed to adjust the size of the figure. See the next page for a description of this constant and how it should be used in your program.

The course web site will contain files that show you the expected output if your size constant is changed to various other values. You can use our Output Comparison Tool on the course web site to measure numbers of characters and to verify the correctness of your output for various values of the size constant.

Part A will not be graded for style except as specified in Part A. But Part B will be graded both on "external correctness" (whether the program runs and produces exactly the expected output) and "internal correctness" (whether your source code follows the style guidelines in this document).

Style Guidelines (for Part B):

Use of `for` loops and string multiplication

This program is intended to test your knowledge through lecture 6, especially `for` loops, string multiplication and single parameters. If you like, you may also use the Python features from lectures 7 and 8, multiple parameters and nested loops although you are not required to do so and will receive no extra credit for doing so. You may not use any Python constructs beyond lecture 8.

Use of functions with single parameters for structure and elimination of redundancy

Continue to use functions to structure your solution in such a way that the functions match the structure of the output itself. Avoid significant redundancy; use functions so that no substantial groups of identical statements appear in your code. No `print` statements should appear in your `main` function. You must also use functions to capture redundancy in lines that are identical in all ways but one. For instance, a function with a single parameter could be used to eliminate redundancy in the following two lines:

```
|.../\/\....../\/\...|
|.../\/\vvvvvv/\/\...|
```

Source code aesthetics (commenting, indentation, identifier names)

No line of your code should be over 80 characters long.

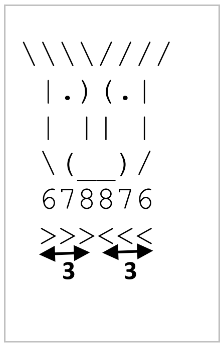
Give meaningful names to functions and variables in your code. Follow Python’s naming standards about the format of `function_and_variable_names`, and `CONSTANT_NAMES`.

Include a comment header at the beginning of your program with basic information and a description of the program. **Also include a comment at the start of each function**, describing its behavior. Your comments should be in your own words.

Constant for figure's width

You should create one (and only one) constant to represent the width of the pieces of the figure. Use **3** as the default value of your constant to produce the figure shown above. Your figure must be based on that exact value to receive full credit.

On any given execution your program will produce just one version of the figure. However, you should refer to the constant throughout your code, so that by simply changing your constant's value and rerunning, your program would produce a figure of a different size. Your program should scale correctly for any constant value of 3 or greater.



```
\\\\////
 |.)(.|
 | || |
 \(__)/
 678876
 >>><<<
```

Development Strategy (How to Get Started):

This program is best completed in stages. We strongly recommend the following development strategy:

1. **Code w/o Constant**: Completely write the Python code to draw the Totem Pole at its default size of 3.
2. **Tables**: Examine the output at different sizes and write tables to discover the patterns of repeated characters on each line.
3. **Code w/ Constant**: Add a constant to your code, using the equations from step 2, so that the pole can scale to different sizes.

To summarize, you should not worry about the constant at first. Write an initial program without a constant, using loop tables or pseudocode to help you deduce the patterns in the output. After your figure looks correct at the default size, begin a second version with the constant.