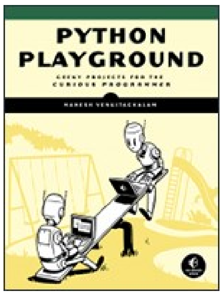

Python Playground: Geeky Projects for the Curious Programmer

by Mahesh Venkitachalam

No Starch Press. (c) 2016. Copying Prohibited.

Chapter 6: ASCII Art

In the 1990s, when email ruled and graphics capabilities were limited, it was common to include a signature in your email that contained a graphic made of text, commonly called *ASCII art*. (ASCII is simply a character- encoding scheme.) Figure 6-1 shows a couple of examples. Although the Internet has made sharing images immeasurably easier, the humble text graphic isn't quite dead yet.

```
           \\|/
          (o o)
____o00o_( - )_o00o_________
________________________________
+                        J.  RAVIPRAKASH +
+-------------------------------------------------------------+
+postdoctoral Researcher           | e-mail-->rxil0flpsu.edu +
+MateriaLs Research Laboratory     | Tel   -->(814) 865-9931 +
+Permsylvania State University     | fax   -->(814) 863-6734 +
+-------------------------------------------------------------+
+        web page --> http://www.personal.psu.edu/-rxil0     +
--------------------------------------------------------------
         ||     ||
        ooo0   0ooo
```

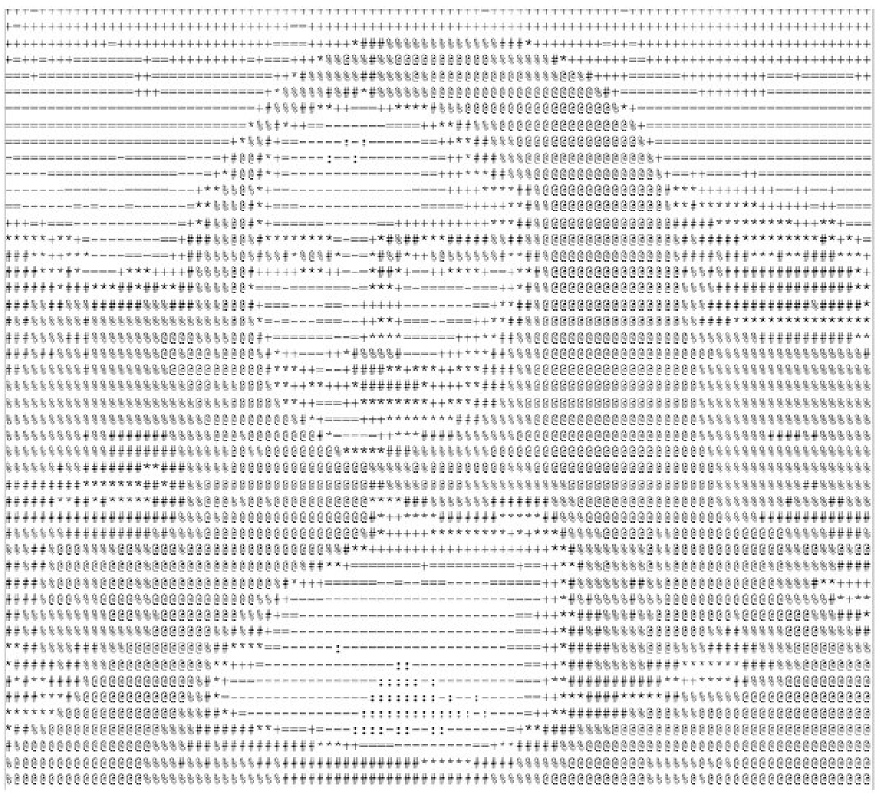

Figure 6-1: Examples of ASCII art

ASCII art has its origins in typewriter art created in the late 1800s. In the 1960s, when computers had minimal graphics-processing hardware, ASCII was used to represent images. These days, ASCII art continues as a form of expression on the Internet, and you can find a variety of creative examples online.

In this project, you'll use Python to create a program that generates ASCII art from graphical images. The program will let you specify the width of the output (the number of columns of text) and set a vertical scale factor. It also supports two mappings of grayscale values to ASCII characters: a sparse 10-level mapping and a more finely calibrated 70-level mapping.

To generate your ASCII art from an image, you'll learn how to do the following:

- Convert images to grayscale using Pillow, a fork of the Python Imaging Library (PIL)
- Compute the average brightness of a grayscale image using numpy

- Use a string as a quick lookup table for grayscale values

How It Works

This project takes advantage of the fact that from a distance, we perceive grayscale images as the average value of their brightness. For example, in Figure 6-2, you can see a grayscale image of a building and, next to it, an image filled with the average brightness value of the building image. If you look at the images from across the room, they will look similar.

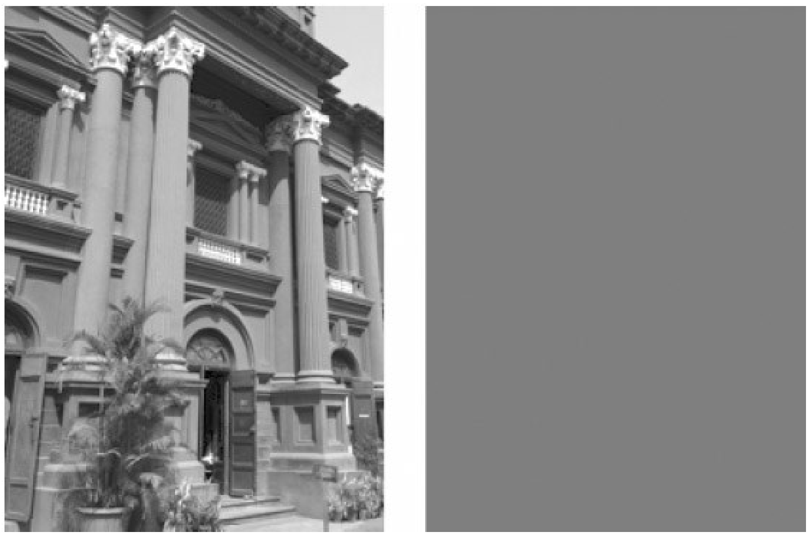
Figure 6-2: Average value of a grayscale image

The ASCII art is generated by splitting an image into tiles and replacing the average RGB value of a tile with an ASCII character. From a distance, since our eyes have limited resolution, we sort of see the "average" values in ASCII art while losing the details that would otherwise make the art look less real.

This program will take a given image and first convert it to 8-bit grayscale so that each pixel has a grayscale value in the range [0, 255] (the range of an 8-bit integer). Think of this 8-bit value as *brightness*, with 0 being black and 255 being white and the values in between being shades of gray.

Next, it will split the image into a grid of $M \times N$ tiles (where M is the number of rows and N the number of columns in the ASCII image). The program will then calculate the average brightness value for each tile in the grid and match it with an appropriate ASCII character by predefining a *ramp* (an increasing set of values) of ASCII characters to represent grayscale values in the range [0, 255]. It will use these values as a lookup table for the brightness values.

The finished ASCII art is just a bunch of lines of text. To display the text, you'll use a constant-width (also called *monospace*) font such as Courier because unless each text character has the same width, the characters in the image won't line up properly along a grid and you'll end up with unevenly spaced and scrambled output.

The *aspect ratio* (the ratio of width to height) of the font used also affects the final image. If the aspect ratio of the space taken up by a character is different from the aspect ratio of the image tile the character is replacing, the final ASCII image will appear distorted. In effect, you're trying to replace an image tile with an ASCII character, so their shapes need to match. For example, if you were to split your image into square tiles and then replace each of the tiles with a font that is stretched in height, the final output would appear stretched vertically.

To address this issue, you'll scale the rows in your grid to match the Courier aspect ratio. (You can send the program command line arguments to modify the scaling to match other fonts.)

In sum, here are the steps the program takes to generate the ASCII image:

1. Convert the input image to grayscale.
2. Split the image into $M \times N$ tiles.
3. Correct M (the number of rows) to match the image and font aspect ratio.

4. Compute the average brightness for each image tile and then look up a suitable ASCII character for each.

5. Assemble rows of ASCII character strings and print them to a file to form the final image.

Requirements

In this project, you'll use Pillow, the friendly fork of the Python Imaging Library, to read in the images, access their underlying data, and create and modify them. You'll also use the numpy library to compute averages.

The Code

You'll begin by defining the grayscale levels used to generate the ASCII art. Then you'll look at how the image is split into tiles and how average brightness is computed for those tiles. Next, you'll work on replacing the tiles with ASCII characters to generate the final output. Finally, you'll set up command line parsing for the program to allow a user to specify the output size, output filename, and so on.

For the full project code, skip to "The Complete Code" on page 95.

Defining the Grayscale Levels and Grid

As the first step in creating your program, define the two grayscale levels used to convert brightness values to ASCII characters as global values.

```
  # 70 levels of gray
1 gscale1 = "$@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\|()1{}[]?-_+~<>i!lI;:,\"^
            `". "
  # 10 levels of gray
2 gscale2 = "@%#*+=-:. "
```

The value gscale1 at 1 is the 70-level grayscale ramp, and gscale2 at 2 is the simpler 10-level grayscale ramp. Both of these values are stored as strings, with a range of characters that progress from darkest to lightest. (To learn more about how characters are represented as grayscale values, see Paul Bourke's "Character Representation of Grey Scale Images" at *http://paulbourke.net/dataformats/asciiart/*.)

Now that you have your grayscale ramps, you can set up the image. The following code opens the image and splits it into a grid:

```
    # open the image and convert to grayscale
1   image = Image.open(fileName).convert("L")
    # store the image dimensions
2   W, H = image.size[0], image.size[1]
    # compute the tile width
3   w = W/cols

    # compute the tile height based on the aspect ratio and scale of the font
4   h = w/scale
    # compute the number of rows to use in the final grid
5   rows = int(H/h)
```

At 1, Image.open() opens the input image file and Image.convert() con-verts the image to grayscale. The "L" stands for *luminance*, a measure of the brightness of an image.

At 2, you store the width and height of the input image. At 3, you compute the width of a tile for the number of columns (cols) specified by the user. (The program uses a default of 80 columns if the user doesn't set another value in the command line.) For the division at , use floatingpoint, not integer division, in order to avoid truncation errors while calculating the dimensions of the tiles.

Once you know the width of a tile, you compute its height at 4 using the vertical scale factor passed in as scale. At 5, use this grid height to compute the number of rows.

The scale factor sizes each tile to match the aspect ratio of the font you are using to display the text so that the final image won't be distorted. The value for scale can be passed in as an argument, or it's set to a default of 0.43, which works well for displaying the result in Courier.

Computing the Average Brightness

Next, you compute the average brightness for a tile in the grayscale image. The function getAverageL() does the job.

```
1   def getAverageL(image):
    # get the image as a numpy array
2   im = np.array(image)
    # get the dimensions
3   w,h = im.shape
    # get the average
4   return np.average(im.reshape(w*h))
```

At 1, the image tile is passed in as a PIL Image object. Convert image into a numpy array at 2, at which point im becomes a two-dimensional array of brightness for each pixel. At 3, you store the dimensions (width and height) of the image. At 4, numpy.average() computes the average of the bright ness values in the image by using numpy.reshape() to first convert the two-dimensional array of the dimensions width and height (w,h) into a flat one-dimensional array whose length is a product of the width times the height (w*h). The numpy.average() call then sums these array values and computes the average.

Generating the ASCII Content from the Image

The main part of the program generates the ASCII content from the image.

```
    # an ASCII image is a list of character strings
1   aimg = []
       # generate the list of tile dimensions
       for j in range(rows):
2          y1 = int(j*h)
           y2 = int((j+1)*h)
           # correct the last tile
           if j == rows-1:
               y2 = H
           # append an empty string
3          aimg.append("")
           for i in range(cols):
               # crop the image to fit the tile
4              x1 = int(i*w)
               x2 = int((i+1)*w)
               # correct the last tile
5         if i == cols-1:
              x2 = W
          # crop the image to extract the tile into another Image object
6         img = image.crop((x1, y1, x2, y2))
          # get the average luminance
7         avg = int(getAverageL(img))
          # look up the ASCII character for grayscale value (avg)
          if moreLevels:
8             gsval = gscale1[int((avg*69)/255)]
          else:
9             gsval = gscale2[int((avg*9)/255)]
          # append the ASCII character to the string
10        aimg[j] += gsval
```

In this section of the program, the ASCII image is first stored as a list of strings, which is initialized at 1. Next, you iterate through the calculated number of row image tiles, and at 2 and the following line, you calculate the starting and ending y-coordinates of each image tile. Although these are floating-point calculations, truncate them to integers before passing them to an image-cropping method.

Next, because dividing the image into tiles creates edge tiles of the same size only when the image width is an integer multiple of the number of columns, correct for the y-coordinate of the tiles in the last row by setting the y-coordinate to the image's actual height. By doing so, you ensure that the top edge of the image isn't truncated.

At 3, you add an empty string into the ASCII as a compact way to represent the current image row. You'll fill in this string next. (You treat the string as a list of characters.)

At 4 and the next line, you compute the left and right x-coordinates of each tile, and at 5, you correct the x-coordinate for the last tile for the same reasons you corrected the y-coordinate. Use image.crop() at 6 to extract the image tile and then pass that tile to the getAverageL() function 7, defined in "Computing the Average Brightness" on page 93, to get the average brightness of the tile. At 9, you scale down the average brightness value from [0, 255] to [0, 9] (the range of values for the default 10-level grayscale ramp). You then use gscale2 (the stored ramp string) as a lookup table for the relevant ASCII value. The line at 8 is similar, except that it's used only when the command line flag is set to use the ramp with 70 levels. Finally, you append the looked-up ASCII value, gsval, to the text row at 10, and the code loops until all rows are processed.

Command Line Options

Next, define some command line options for the program. This code uses the built-in argparse class:

```
     parser = argparse.ArgumentParser(description="descStr")
     # add expected arguments
1    parser.add_argument('--file', dest='imgFile', required=True)
2    parser.add_argument('--scale', dest='scale', required=False)
3    parser.add_argument('--out', dest='outFile', required=False)
4    parser.add_argument('--cols', dest='cols', required=False)
5    parser.add_argument('--morelevels', dest='moreLevels', action='store_true')
```

At 1, you include options to specify the image file to input (the only required argument) and to set the vertical scale factor 2, the output filename 3, and the number of text columns in the ASCII output 4. At 5, you add a --morelevels option so the user can select the grayscale ramp with more levels.

Writing the ASCII Art Strings to a Text File

Finally, take the generated list of ASCII character strings and write those strings to a text file:

```
     # open a new text file
1    f = open(outFile, 'w')
     # write each string in the list to the new file
2    for row in aimg:
         f.write(row + '\n')
     # clean up
3    f.close()
```

At 1, you use the built-in open() method to open a new text file for writing. Then you iterate through each string in the list and write it to the file at 2. At 3, you close the file object to release system resources.

The Complete Code

Here is the complete ASCII art program. You can also download the code for this project from *https://github.com/electronut/pp/blob/master/ascii/ascii. py*.

```
import sys, random, argparse
import numpy as np
import math

from PIL import Image

# grayscale level values from:
# http://paulbourke.net/dataformats/asciiart/

# 70 levels of gray
gscale1 = "$@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\|()1{}[]?-_+~<>i!lI;:,\"^
          `'. "
# 10 levels of gray
gscale2 = '@%#*+=-:. '

def getAverageL(image):
    """
    Given PIL Image, return average value of grayscale value
    """
    # get image as numpy array
    im = np.array(image)
    # get the dimensions
    w,h = im.shape
    # get the average
    return np.average(im.reshape(w*h))
def covertImageToAscii(fileName, cols, scale, moreLevels):
    """
    Given Image and dimensions (rows, cols), returns an m*n list of Images
    """
    # declare globals
    global gscale1, gscale2
    # open image and convert to grayscale
    image = Image.open(fileName).convert('L')
    # store the image dimensions
    W, H = image.size[0], image.size[1]
    print("input image dims: %d x %d" % (W, H))
    # compute tile width
    w = W/cols
    # compute tile height based on the aspect ratio and scale of the font
    h = w/scale
    # compute number of rows to use in the final grid
    rows = int(H/h)

    print("cols: %d, rows: %d" % (cols, rows))
    print("tile dims: %d x %d" % (w, h))

    # check if image size is too small
    if cols > W or rows > H:
        print("Image too small for specified cols!")
        exit(0)

    # an ASCII image is a list of character strings
    aimg = []
    # generate the list of tile dimensions
    for j in range(rows):
        y1 = int(j*h)
        y2 = int((j+1)*h)

        # correct the last tile
        if j == rows-1:
           y2 = H
        # append an empty string
        aimg.append("")
        for i in range(cols):
            # crop the image to fit the tile
            x1 = int(i*w)
            x2 = int((i+1)*w)
            # correct the last tile
            if i == cols-1:
               x2 = W
            # crop the image to extract the tile into another Image object
            img = image.crop((x1, y1, x2, y2))
            # get the average luminance
            avg = int(getAverageL(img))
            # look up the ASCII character for grayscale value (avg)
            if moreLevels:
               gsval = gscale1[int((avg*69)/255)]
            else:
               gsval = gscale2[int((avg*9)/255)]
            # append the ASCII character to the string
            aimg[j] += gsval

    # return text image
    return aimg

# main() function
def main():
    # create parser
    descStr = "This program converts an image into ASCII art."
    parser = argparse.ArgumentParser(description=descStr)
    # add expected arguments
    parser.add_argument('--file', dest='imgFile', required=True)
    parser.add_argument('--scale', dest='scale', required=False)
    parser.add_argument('--out', dest='outFile', required=False)
    parser.add_argument('--cols', dest='cols', required=False)
    parser.add_argument('--morelevels', dest='moreLevels', action='store_true')

    # parse arguments
    args = parser.parse_args()
    imgFile = args.imgFile
    # set output file
    outFile = 'out.txt'
    if args.outFile:
        outFile = args.outFile
    # set scale default as 0.43, which suits a Courier font
    scale = 0.43
    if args.scale:
       scale = float(args.scale)
    # set cols
    cols = 80
    if args.cols:
        cols = int(args.cols)
    print('generating ASCII art...')
    # convert image to ASCII text
    aimg = covertImageToAscii(imgFile, cols, scale, args.moreLevels)

    # open a new text file
    f = open(outFile, 'w')
    # write each string in the list to the new file
    for row in aimg:
        f.write(row + '\n')
    # clean up
    f.close()
    print("ASCII art written to %s" % outFile)

# call main
if __name__ == '__main__':
    main()
```

Running the ASCII Art Generator

To run your finished program, enter a command like the following one, replacing data/robot.jpg with the relative path to the image file you want to use.

```
$ python ascii.py --file data/robot.jpg --cols 100
```

Figure 6-3 shows the ASCII art that results from sending the image *robot.jig* (at the left).

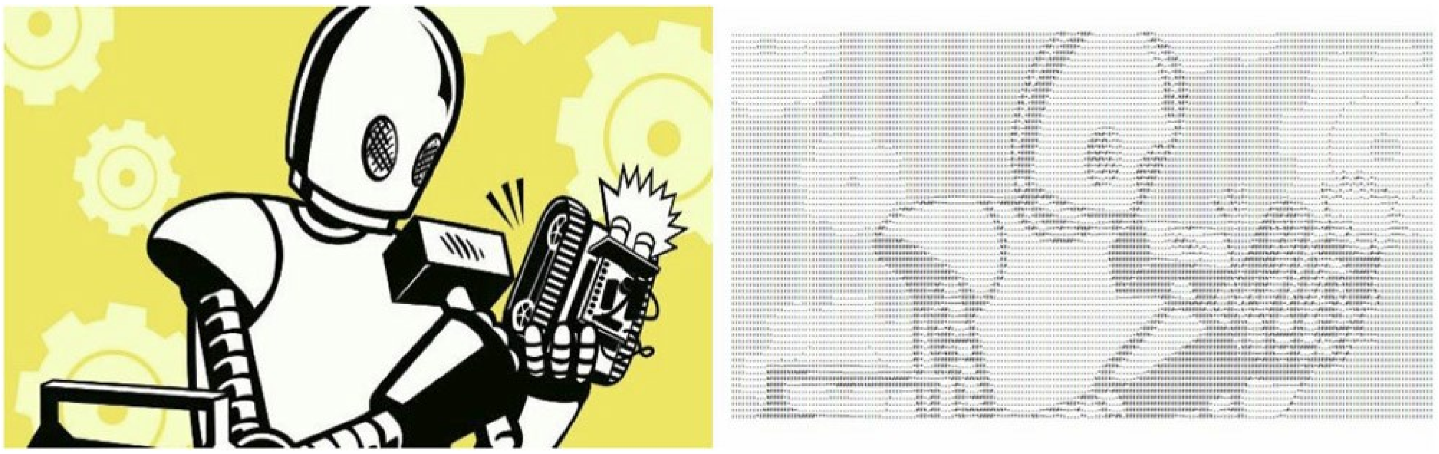

Figure 6-3: Sample run of ascii.py

Now you are all set to create your own ASCII art!

Summary

In this project, you learned how to generate ASCII art from any input image. You also learned how to convert an image to grayscale by computing average brightness values and how to replace part of an image with a character based on the grayscale value. Have fun creating your own ASCII art!

Experiments!

Here are some ideas for exploring ASCII art further.

1. Run the program with the command line option --scale 1.0. How does the resulting image look? Experiment with different values for scale. Copy the output to a text editor and try setting the text to different (fixed-width) fonts to see how doing so affects the appearance of the final image.
2. Add the command line option --invert to the program to invert the ASCII art input values so that black appears white, and vice versa. (Hint: try subtracting the tile brightness value from 255 during lookup.)
3. In this project, you created lookup tables for grayscale values based on two-character hard-coded ramps. Implement a command line option to pass in a different character ramp to create the ASCII art, like so:

```
python3 ascii.py --map "@$%^`."
```

A ramp like the previous one should create the ASCII output using the given six-character ramp, where @ maps to a brightness value of 0 and. maps to a value of 255.