

Activity 1.4.5 Image Algorithms

Introduction

|  |  |
| --- | --- |
| You’ve learned the basics of writing computer programs. But most programming builds on code that has already been written. Knowing how to find and use code from other people will help make you an efficient and successful software developer.  You could create your own algorithms, for example, to rotate an image or to identify the objects in an image. But others have already solved those problems! There are many advantages to using existing code. You save time, of course. But you also connect to a community of people, making it easier for them to help you, and making it more likely they will be able to use what you create. How will you put other people’s code to use? | The Droste effect – image recursion |

Materials

* Computer with Enthought Canopy distribution of Python
* Webcam or other way to capture a digital picture
* Image files and Python files for Activity 1.4.5

Procedure

1. Form pairs as directed by your teacher. Meet or greet each other to practice professional skills. Set team norms.
2. Launch Canopy. Open an editor window. Set the working directory to your folder. Create a new Python file. Save the file as JDoe\_JSmith\_1\_4\_5.py.
3. Some of your work in this assignment will involve trial and error using the iPython session. You might want a record of commands you have tried, so begin logging the session.

In []: %logstart -ort studentNames\_1\_4\_5.log

1. Obtain and unzip 1.4.5 sourceFiles and 1.4.5 imageFiles. Open mask.py provided in the source files. Execute the code in the code editor. This code defines three new functions but does not actually call any of them, so nothing will visibly occur. Examine the code in the Canopy code editor. What are the names of the three functions?

Round\_corners get\_images round\_courners\_of\_all\_images

1. In the iPython session, change your working directory to the unzipped folder 1.4.5 Images. Examine the contents of that folder using Windows Explorer.
2. In the iPython session, execute the following command. The function will take a moment to execute.

|  |  |
| --- | --- |
| In []: | round\_corners\_of\_all\_images() |

This function will create a new folder modified in the 1.4.5 Images folder. Examine the new folder’s contents. What did the function do?

It only created a new folder named modified

1. A portion of mask.py is shown below. This is the code for the first function, round\_corners(), defined in the program file. Answer the questions below about the code.

|  |  |
| --- | --- |
| 12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | **def** round\_corners(original\_image, percent\_of\_side):  """ Rounds the corner of a PIL.Image  original\_image must be a PIL.Image  Returns a new PIL.Image with rounded corners, where  0 < percent\_of\_side < 1 is the corner radius as  portion of shorter dimension of original\_image  """  *#set the radius of the rounded corners*  width, height = original\_image.size  radius = int(percent\_of\_side \* min(width, height)) *#radius in pixels*  *###*  *#create a mask*  *###*  *#start with transparent mask*  rounded\_mask = PIL.Image.new('RGBA', (width, height), (127,0,127,0))  drawing\_layer = PIL.ImageDraw.Draw(rounded\_mask)  *# Overwrite the RGBA values with A=255.*  *# The 127 for RGB values was used merely for visualizing the mask*  *# Draw two rectangles to fill interior with opaqueness*  drawing\_layer.polygon([(radius,0),(width-radius,0),  (width-radius,height),(radius,height)],  fill=(127,0,127,255))  drawing\_layer.polygon([(0,radius),(width,radius),  (width,height-radius),(0,height-radius)],  fill=(127,0,127,255))  *#Draw four filled circles of opaqueness*  drawing\_layer.ellipse((0,0, 2\*radius, 2\*radius),  fill=(0,127,127,255)) *#top left*  drawing\_layer.ellipse((width-2\*radius, 0, width,2\*radius),  fill=(0,127,127,255)) *#top right*  drawing\_layer.ellipse((0, height-2\*radius, 2\*radius,height),  fill=(0,127,127,255)) *#bottom left*  drawing\_layer.ellipse((width-2\*radius, height-2\*radius, width, height),  fill=(0,127,127,255)) *#bottom right*  *# Uncomment the following line to show the mask*  *# plt.imshow(rounded\_mask)*  *# Make the new image, starting with all transparent*  result = PIL.Image.new('RGBA', original\_image.size, (0,0,0,0))  result.paste(original\_image, (0,0), mask=rounded\_mask)  **return** result |

1. The function round\_corners() was one we made up. It is defined here to take 2 arguments. According to the function’s docstring (lines 13 – 19), what type of variable is each argument? What type of variable is returned by the function?

Argument 1: array (picture)

Argument 2: floating point between 0 and 1

Return value: “result” variable (array) (picture)

1. Line 29 creates a new image filled with a single color. What color is it?

purple

1. Line 30 creates a new ImageDraw object associated with the new PIL.Image object from line 29. What are the names of these two objects?

Object created in line 29: rounded\_mask

Object created in line 30: drawing\_layer

1. In Step 17g of the last activity, you used the PIL.Image.paste() documentation to identify the purpose of the mask argument of paste() in that program. Refer to your answer to that question.

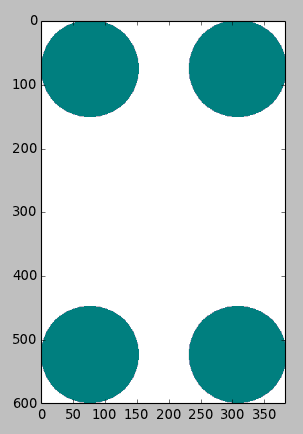
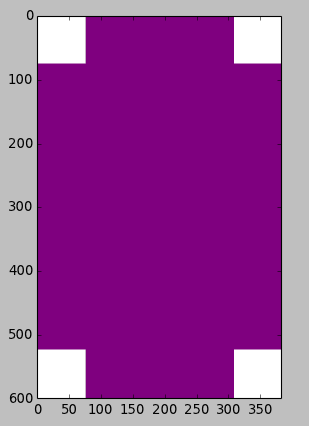
The rounded\_mask object is used as the mask argument in line 58. The paste() function uses only the alpha channel of the mask argument. It uses this alpha value to decide how to combine the pixels of the two other images. To make an image transparent in the corners, what alpha value would we want for the mask in the corners?

0

1. Lines 36 – 51 take advantage of the fact that *Python* allows a line to be continued onto the next line immediately after the comma in a list of arguments. Note the convention to indent the line continuation: the continued line is indented to line up with the parenthesis that begin the argument list.

The following images were produced by uncommenting line 54 and using triple single-quotes to comment out either lines 36 – 41 or lines 44 – 51. The result is that all six shapes created in lines 36 – 51 are shown. You don’t need to repeat that process; just read the code. In the figures below, label each shape with the corresponding line number used to create it.

36 – 41 44 – 51

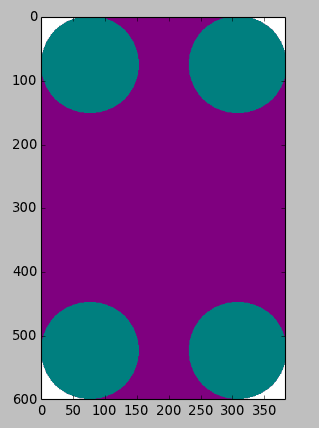
 

1. Line 57 creates another new PIL.Image object called result. It will hold the modified image, but when created, it is filled with a solid color. What color is it?

Fully transparent black

1. Line 58 pastes the original image into result. Pixels from the original\_image are only used if the corresponding pixels from rounded\_mask have alpha>0. The pixels in the corners are left as-is in result. What are the color values in the corners?

(0,0,0,0)



1. The code shown below defines get\_images(), the second function created by mask.py. Refer to the code and answer the following questions.

|  |  |
| --- | --- |
| 61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85 | **def** get\_images(directory=None):  """ Returns PIL.Image objects for all the images in directory.  If directory is not specified, uses current directory.  Returns a 2-tuple containing  a list with a PIL.Image object for each image file in root\_directory,  and a list with a string filename for each image file in root\_directory  """  **if** directory **==** None:  directory = os.getcwd() *# Use working directory if unspecified*  image\_list=[] *# Initialize aggregators*  file\_list = []  directory\_list = os.listdir(directory) *# Get list of files*  **for** entry **in** directory\_list:  absolute\_filename = os.path.join(directory, entry)  **try**:  image = PIL.Image.open(absolute\_filename)  file\_list += [entry]  image\_list += [image]  **except** **IOError**:  **pass** *# do nothing with errors tying to open non-images*  **return** image\_list, file\_list |

1. How many arguments can be passed to the function get\_images()? Because a default value is specified for directory, that argument is optional, so get\_images() can be passed either \_0\_ or \_1\_ arguments.
2. Read the docstring and examine the return statement on line 85. How many objects and what type are returned by the function?

2 lists are returned by the function

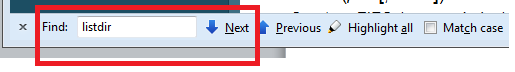
1. This function calls three functions from the os module. Find the three calls to the os module used in the code and list the three functions below.

os.\_\_\_\_\_\_\_\_()

os.\_\_\_\_\_\_\_\_()

os.path.\_\_\_\_\_\_\_\_\_()

1. Use an Internet search engine to find the official documentation for the os module. You could try “os Python” for your search terms. You can identify the official documentation because it will come from a webserver in the python.org domain. Use the find-in-document utility (**Ctrl-F**, then repeatedly choose **Next**) to find the documentation for one of the functions above. Paste a sentence about that function above.



1. Lines 79 – 84 demonstrate some statements new to you. This is a try–except structure, which is the *Python* **exception handler**. An exception handler lists the code to be executed if an error occurs.

The PIL.Image.open(filename) function can cause an error that would halt the program if the filename does not specify an image file. Specifically, open()reports an IOError type of error. If that error is reported to the Python interpreter, the program is halted and the error is printed in the **traceback** at the interpreter prompt. The traceback shows what lines of code caused the error.

By using a try–except structure, such an error is **caught** instead of halting the program. An error that has been caught doesn’t get reported back to the *Python* interpreter. The handler can opt to keep the error invisible to the user and keep the program running. That’s a good thing if the programmer expected the error and wants the program to keep running. That can be a bad thing if the code accidentally catches other exceptions, like the user trying to quit the program. So the program should only catch the specific class of errors that are expected, such as IOError in this case.

Here is how the try–except structure controls the program flow. The statements in the try block are executed one at a time. If one of those statements causes an error, the interpreter checks to see if the type of error matches the type of errors listed in the except statement. If the error type matches the except statement, then the interpreter does not execute the rest of the try block and instead continues execution with the except block of code. If the error doesn’t match the except statement, then the error is not caught and the program will be halted.

In this code the except block only contains the *Python* pass statement, which does nothing. It is used when *Python* syntax requires a statement but no action is required. So the except block catches the error but doesn’t do anything with it.

If the try block of code is executed without any errors, the except block of code is skipped. Execution continues after the try–except structure.

Why does this program use a try-except structure to open all images in a directory?

1. Considering the information above, explain what lines 81 and 82 do. Describe which circumstances allow them to be executed.
2. The code shown below defines round\_corners\_of\_all\_images(), the third function defined in mask.py. Refer to the code and answer the following questions.

|  |  |
| --- | --- |
| 87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117 | **def** round\_corners\_of\_all\_images(directory=None):  """ Saves a modified version of each image in directory.  Uses current directory if no directory is specified.  Puts images in subdirectory 'modified', creating it if needed.  New image files are of type PNG and have transparent rounded corners.  """  **if** directory == None:  directory = os.getcwd() *# Use working directory if unspecified*  *# Create a new directory 'modified'*  new\_directory = os.path.join(directory, 'modified')  **try**:  os.mkdir(new\_directory)  **except** **OSError**:  **pass** *# if the directory already exists, proceed*  *#load all the images*  image\_list, file\_list = get\_images(directory)  *#go through the images and save modified versions*  **for** n **in** range(len(image\_list)):  *#parse the filename*  filename, filetype = file\_list[n].split('.')    *#round the corners with radius = 30% of short side*  new\_image = round\_corners(image\_list[n],.30)  *#save the altered image, using PNG to retain transparency*  new\_image\_filename = os.path.join(new\_directory,filename + '.png')  new\_image.save(new\_image\_filename) |

1. In line 101, mkdir() creates a new directory. Explain why you think this function call needed to be embedded in a try-except structure.
2. In line 109, what is represented by len(image\_list)? In other words, what does that number mean?
3. What is the role being played by n in lines 109, 111, and 114?
4. Save the code using a different file name and modify it to accomplish one of the following objectives. Your code should include two new functions modeled after round\_corners() and round\_corners\_of\_all\_images().
5. Create a function frame\_all\_images(color, wide) that makes a framed version of all pictures in a directory, where the frame is specified by a color  (r,g,b) and has thickness wide.
6. Create a function alter\_all\_images() that makes a new version of all pictures in a directory, with the modification being of your own design.
7. Save your *Python* file in the code editor. If you were logging the iPython session, save it with %logstop.

In []: %logstop

Conclusion

1. Icons on the desktop are not usually rectangular. You can see through the desktop behind their irregular edges. How is this accomplished?
2. You have 2000 images and would like thumbnails of all of them so that they will be transparent in their corners. Describe the algorithm you would use to accomplish this.
3. The code provided was divided into three functions. Describe how this made the code reuse easier.