1.4.5 Image Algorithms

PLTW | Computer Science

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?

Procedure

1. Log-in to Cloud 9 and determine whose account will be used for this lesson (It should be the opposite of whoever has 1.4.4). Create a new folder titled *1.4.5*. Then download `1.4.5_Images.zip`,, and `mask.py` from the Google Classroom and load them into your *1.4.5* folder. Once you have done that, please rename `mask.py` as *Last_Last_1.4.5.py* and start working on this assignment.

2. In the Bash terminal tab don't forget to start your iPython session

   - Type `ipython` to launch the interactive python enviornment.

   ```
   c9username:~/workspace $ ipython
   ```

3. For this assignment you will be working with your partner to get everything done, please record all answers in the code editor document as comments. But please make sure to record all answers neatly and accurately.

4. For this assignment, it is going to be extremely important to tell Cloud 9 exactly where to be, change your working directory to the unzipped folder `1.4.5_Images` and stay in that directory for this entire assignment. Examine the contents of that folder using Windows Explorer.

5. Examine the `mask.py` provided in the source files (should now be named *Last_Last_1.4.5.py*). How would we execute this code? Now, 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 code editor. What are the names of the three functions?

6. On line 116, add the following code to execute the following command. The function will take a moment to execute.

   ```
   116  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?

7. 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.

```
 9 def round_corners(original_image, percent_of_side):
10     """ Rounds the corner of a PIL.Image
11     
12     original_image must be a PIL.Image
13     Returns a new PIL.Image with rounded corners, where
14     0 < percent_of_side < 1 is the corner radius as
15     portion of shorter dimension of original_image
16     """
17     #set the radius of the rounded corners
18     width, height = original_image.size
19     radius = int(percent_of_side * min(width, height)) #radius in pixels
20     
21     ###
22     #create a mask
23     ###
24     
25     #start with transparent mask
26     rounded_mask = PIL.Image.new('RGBA', (width, height), (127,0,127,0))
27     drawing_layer = PIL.ImageDraw.Draw(rounded_mask)
28     
29     # Overwrite the RGBA values with A=255.
30     # The 127 for RGB values was used merely for visualizing the mask
31     
32     # Draw two rectangles to fill interior with opaqueness
33     drawing_layer.polygon([(radius,0),(width-radius,0),
34                           (width-radius,height),(radius,height)],
35                           fill=(127,0,127,255))
36     drawing_layer.polygon([(0,radius),(width,radius),
37                           (width,height-radius),(0,height-radius)],
38                           fill=(127,0,127,255))
39     
40     #Draw four filled circles of opaqueness
41     drawing_layer.ellipse((0,0, 2*radius, 2*radius),
42                           fill=(0,127,127,255)) #top left
43     drawing_layer.ellipse((width-2*radius, 0, width,2*radius),
44                           fill=(0,127,127,255)) #top right
45     drawing_layer.ellipse((0, height-2*radius, 2*radius,height),
46                           fill=(0,127,127,255)) #bottom left
47     drawing_layer.ellipse((width-2*radius, height-2*radius, width, height),
48                           fill=(0,127,127,255)) #bottom right
49     
50     # Uncomment the following line to show the mask
51     # plt.imshow(rounded_mask)
52     
53     # Make the new image, starting with all transparent
54     result = PIL.Image.new('RGBA', original_image.size, (0,0,0,0))
55     result.paste(original_image, (0,0), mask=rounded_mask)
56     return result
```

a. The function `round_corners()` was one we made up. It is defined here to take ___ arguments. According to the function's docstring (lines 10 – 16), what type of variable is each argument? What type of variable is returned by the function?

Argument 1:

Argument 2:

Return value:

b. Line 26 creates a new image filled with a single color. What color is it?

c. Line 27 creates a new `ImageDraw` object associated with the new `PIL.Image` object from line 26. What are the names of these two objects?

   Object created in line 26:

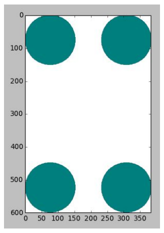
   Object created in line 27:

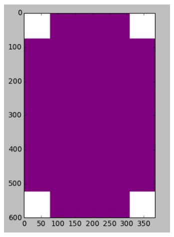
d. In Step 19g of the last activity, you used the `PIL.Image.paste()` documentation to identify the purpose of the mask argument of `paste()` in that program. Refer to your answer to that question.

   The `rounded_mask` object is used as the mask argument in line 55. 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?

e. Lines 33 – 38 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.

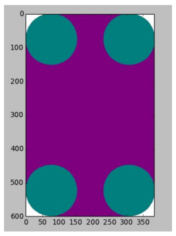
   The following images were produced by uncommenting line 51 and using triple single-quotes to comment out either lines 33 – 38 or lines 41 – 48. The result is that all six shapes created in lines 33 – 48 are shown. You don't need to repeat that process; just read the code. In the figures below, label each shape with the corresponding line number used to create it.





f. Line 54 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?

g. Line 55 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?



8. The code shown below defines `get_images()`, the second function created by `mask.py`. Refer to the code and answer the following questions.

```
58  def get_images(directory=None):
59      """ Returns PIL.Image objects for all the images in directory.
60  
61      If directory is not specified, uses current directory.
62      Returns a 2-tuple containing
63      a list with a PIL.Image object for each image file in root_directory,
64      and a list with a string filename for each image file in root_directory
65      """
66  
67      if directory == None:
68          directory = os.getcwd() # Use working directory if unspecified
69  
70      image_list=[] # Initialize aggregators
71      file_list = []
72  
73      directory_list = os.listdir(directory) # Get list of files
74      for entry in directory_list:
75          absolute_filename = os.path.join(directory, entry)
76          try:
77              image = PIL.Image.open(absolute_filename)
78              file_list += [entry]
79              image_list += [image]
80          except IOError:
81              pass # do nothing with errors tying to open non-images
82      return image_list, file_list
```

a. 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 ___ or ___ arguments.

b. Read the docstring and examine the return statement on line 82. How many objects and what type are returned by the function?

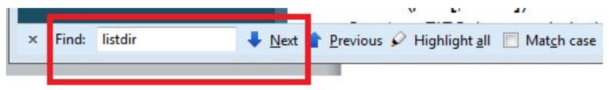
c. This function calls three functions from the `os` module. Find the three calls to the `os` module used in the code and list the three functions below.

`os.__________()`

`os.__________()`

`os.path.__________()`

d. 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 runfind-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.


Find: listdir

e. Lines 76 – 81 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? What are some of the advantages/disadvantages of using a `try`-`except` structure?

f. Considering the information above, explain what lines 80 and 81 do. Describe which circumstances allow them to be executed.

9. 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.

```python
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
```

```
 94
 95     # Create a new directory 'modified'
 96     new_directory = os.path.join(directory, 'modified')
 97     try:
 98         os.mkdir(new_directory)
 99     except OSError:
100         pass # if the directory already exists, proceed
101
102     #load all the images
103     image_list, file_list = get_images(directory)
104
105     #go through the images and save modified versions
106     for n in range(len(image_list)):
107         #parse the filename
108         filename, filetype = file_list[n].split('.')
109
110         #round the corners with radius = 30% of short side
111         new_image = round_corners(image_list[n],.30)
112         #save the altered image, using PNG to retain transparency
113         new_image_filename = os.path.join(new_directory,filename + '.png')
114         new_image.save(new_image_filename)
```

a. In line 98, `mkdir()` creates a new directory. Explain why you think this function call needed to be embedded in a `try`-`except` structure.

b. In line 106, what is represented by `len(image_list)`? In other words, what does that number mean?

c. What is the role being played by `n` in lines 106, 108, and 111?

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. The code provided was divided into three functions. Describe how this made the code reuse easier.
3. Alice and Barb have different ideas about what a "manipulated image" is. Decide whether you think that each of them is right, wrong, or somewhere between. Write an argument in support of your ideas.

   Alice: "All images are manipulated. For one thing a camera is sensitive to certain kinds of light and the developer controls the exposure level. Even our human eyes have a limited number of pixels! There are 'only' 120 million rods and 6 million cones in each retina, so our vision is pixelated just like a digital image. And our vision is also highly processed – even the blind spot in each eye gets filled in. Out of all those millions of light detectors, only about 1 million ganglia neurons go from the eye to the brain. There is no such image as seeing the 'real' thing."

   Barb: "Of course there is a real image. Certain kinds of manipulations are accurate and others tell lies."
4. Under what circumstances is an image yours to use? Yours to distribute? Yours to sell? Write about your thoughts on this question in the context of downloaded images and images you take with a camera.
5. Reflect on the team dynamic and on the design process. What were areas for improvement? What steps could you take next time to make those improvements?