1. Write a function `count_green()` which takes as input an rgb image and returns a count (an integer) of all the pixels which are dominantly green (the green channel's value is larger than the red or blue channels' values). Write the docstring, but don’t include examples.

```python
def count_green(image):
    '''
    (image) -> integer
    Precondition: image is an RGB image
    Returns a count (an integer) of all the pixels which are dominantly green.
    '''
    (w, h) = image.size
    count = 0
    for i in range(w):
        for j in range(h):
            (r, g, b) = image.getpixel((i, j))
            if g > r and g > b:
                count += 1
    return count
```
2. Define the following a function `drawgreenbox()` according to its recipe:

   ```python
   def drawgreenbox(image):
     '''
     (image) -> None
     Accepts an image and draws a green rectangular box in the center of the image. The size of the box is half of the image's size. That is the width and the height of the box is half of the width and the height of the image.
     '''
     
     Note on solutions: Depending on how you interpret the specification for this function, there are several different approaches to this problem that I would consider correct. They vary in how precisely they center the box vs. make the box half the width / half the height, and there are tradeoffs to each. Two solutions are shown below, and both are precise enough to meet the general specification for the function.

     **Solution 1:**

     ```python
     (w, h) = image.size
     start_i = w//4
     start_j = h//4
     end_i = 3*w//4
     end_j = 3*h//4

     for i in range(start_i, end_i):
       for j in range(start_j, end_j):
         image.putpixel((i, j), (0, 255, 0))
     
     return
     ```

     This solution results in a green box that is always half the image width / height (when the # of pixels is even), or half the width / height rounded to the closest pixel (when the # of pixels is odd). The tradeoff is that when the height / width are not divisible by 4, the box will not be centered.

---

<table>
<thead>
<tr>
<th>8 x 8 Image</th>
<th>9 x 7 Image</th>
<th>10 x 8 Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="8x8 Image" /></td>
<td><img src="image" alt="9x7 Image" /></td>
<td><img src="image" alt="10x8 Image" /></td>
</tr>
</tbody>
</table>
Solution 2:

```python
(w, h) = image.size
start_i = w//4
start_j = h//4
end_i = 3*w//4
end_j = 3*h//4

for i in range(start_i, end_i+1):
    for j in range(start_j, end_j+1):
        image.putpixel((i, j), (0,255,0))

return
```

When the number of pixels is not divisible by 4, this solution does a better job of centering the image than the previous solution-- in these cases, the box becomes bigger than half the image width / height in order to achieve an equal # of pixels on either side.