Homework1_denizaytemiz

September 9, 2021

1 ECE 4554/ ECE 5554 / Computer Vision

This file contains Problems 5 and 6 (the coding problems) for Homework 1. Your job is to implement/modify the sections within this notebook that are marked with "TO DO".

1.1 TO DO: Enter your Virginia Tech Username (PID) here: ___________906470552_________

1.2 Honor Code reminder

Once again, please review the Honor Code statement in the syllabus.

1.3 Submission guidelines for the coding problems (Google Colab)

1. Please verify that you have entered your Virginia Tech Username in all of the appropriate places.
2. After clicking Runtime->Run all, verify that all of your solutions are visible in this notebook.
3. Click File->Save near the top of the page to save the latest version of your notebook at Google Drive.
4. Verify that the last 2 cells have executed, creating a PDF version of this notebook at Google Drive. (Note: if you face difficulty with this step, please refer to https://pypi.org/project/notebook-as-pdf/)
5. Look at the PDF file and check that all of your solutions are displayed correctly there.
6. Download your notebook file and the PDF version to your laptop.
7. On your laptop, create a ZIP version of this notebook file. (Please don't include the separate data files.) Use file name Homework1_Code_USERNAME.zip, with your own Username.
8. For your PDF version, use file name Homework1_Notebook_USERNAME.pdf, with your own Username.
9. **Submit these 2 files and your PDF file for Problems 1-4 SEPARATELY to Canvas.** Do not zip them all together.

2 Environment Setup

[59]:
```
# Mount your Google Drive to this notebook
# The purpose is to allow your code to access to your files
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[60]:
```
# Change the directory to your own working directory
# Any files under your working directory are available to your code
# TO DO: enter the name of your directory
import os
os.chdir('/content/drive/MyDrive/ECE5554/HW1')
#os.chdir('/content/drive/MyDrive/Colab Notebooks')
```

[61]:
```
# Import library modules
import sys
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image # PIL is the Python Imaging Library
```

---

3 Problem 5: Working with Python and NumPy (10 points)

a) Write a brief description of the result of each of the following Python commands. Try to guess the result before running the commands interactively. If needed, check the document that is posted at Canvas (An Introduction to NumPy and SciPy) or the online API documentation at https://docs.scipy.org/doc/.

[62]:
```
a = np.array([[1,2,3], [4,5,6], [7,8,9]])
b = a[2,:]
c = a.reshape(-1)
d = np.random.randn(5,1)
e = d[d>0]
x = np.zeros(10)+0.5
y = 0.5*np.ones(len(x))
z = x + y
f = np.arange(1,100)
g = a[::-1]
h = np.random.permutation(10)
```

TO DO: Write your descriptions here.
a: The command creates a 2D array.
b: The command creates a new array 'b' with the elements of array 'a's, starting from a[2] to end of the array a. The a[2] equals the list [7,8,9].

c: We converted the 2D array ’a’, into 1D array and stored it in a new array ’c’.

d: The command generates a 2D array with 5 rows each containing 1 element. The elements are random floats sampled from a univariate Gaussian distribution of mean 0 and variance 1 .

e: The command creates a new array with positive elements of array d with same sequence.

x: np.zeros(10) creates an array of size 10 with all elements are 0. The +0.5 at the end adds every array element a 0.5. Therefore it becomes an array of size 10 with all elements are 0.5.

y: len(x) equals to 10. np.ones(10) creates an array of size 10 with every element being equal to 1. Multiplying the array np.ones(10) by 0.5 means every element in array np.ones(10) is multiplied by 0.5. Therefore we get an array of size 10 with all elements being equal to 0.5.

z: The command adds every corresponding indexes of array x and y in the same index of z. Since elements of x and y are in float type, array z is also in float type (1.).

f: The command returns evenly spaced values starting from 1 to 100(100 excluded),with default step size 1.

g: The command gets the elements in array ’a’ beginning from its last index(due to negative index) with step size 1.

h: The function ouputs a randomly permuted version of np.arange(10).

[63]:
```
# Verify your answers
print(f'a: {a}\n\n b: {b}\n\n c: {c}\n\n d: {d}\n\n e: {e}\n\n x: {x}\n\n y:␣
 ↪{y}\n\n z: {z}\n\n f: {f}\n\n g: {g}\n\n h: {h}')
```

```
a: [[1 2 3]
 [4 5 6]
 [7 8 9]]

 b: [7 8 9]

 c: [1 2 3 4 5 6 7 8 9]

 d: [[ 0.84586429]
 [-2.15262242]
 [-0.235942  ]
 [ 0.23109972]
 [ 2.32978729]]

 e: [0.84586429 0.23109972 2.32978729]

 x: [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]

 y: [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]

 z: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

 f: [ 1  2  3  4  5  6  7  8  9 10 11 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 60 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 86 87 88 89 90 91 92 93 94 95 96
 97 98 99]
```

```
g: [[7 8 9]
 [4 5 6]
 [1 2 3]]

h: [6 2 7 4 1 9 5 0 8 3]
```

b) Write a few lines of code to do each of the following.

Let y be the vector: y = np.array([1, 2, 3, 4, 5, 6]). Use the reshape command to form a new matrix z that looks like this: [[1,2],[3,4],[5,6]]

[64]:
```
y = np.array([1, 2, 3, 4, 5, 6])
###################################
# TO DO: write the code to create z
z=y.reshape(3,2)

###################################
print(f'z: {z}')
```

```
z: [[1 2]
 [3 4]
 [5 6]]
```

Use np.max and np.where to set x to the maximum value that occurs in z (from the previous part of this problem), and set r to the row location and c to the column location where that maximum value occurs. Remember that Python uses 0-indexing.

[65]:
```
###################################
# TO DO: write the code to assign values to x, r, and c
x=np.max(z)
[r,c] = np.where(z==x)

###################################
print(f'x: {x}\nr: {r}\nc: {c}')
```

```
x: 6
r: [2]
c: [1]
```

Let v be the vector: v = np.array([1, 8, 8, 2, 1, 3, 9, 8]). To a new variable x, assign the number of 1's that are present in the vector v.

[66]:
```
v = np.array([1, 8, 8, 2, 1, 3, 9, 8])
###################################
# TO DO: write the code
x = np.count_nonzero(v == 1)

###################################
print(f'x: {x}')
```

```
x: 2
```

Use np.random.randint and create an array named u that contains the result of rolling a six-sided die over N trials. (You can pick a convenient value for N.)

[67]:
```
####################################
# TO DO: write the code
u=np.random.randint(6,size=(12))+1

####################################
print(f'dice: {u}')
```

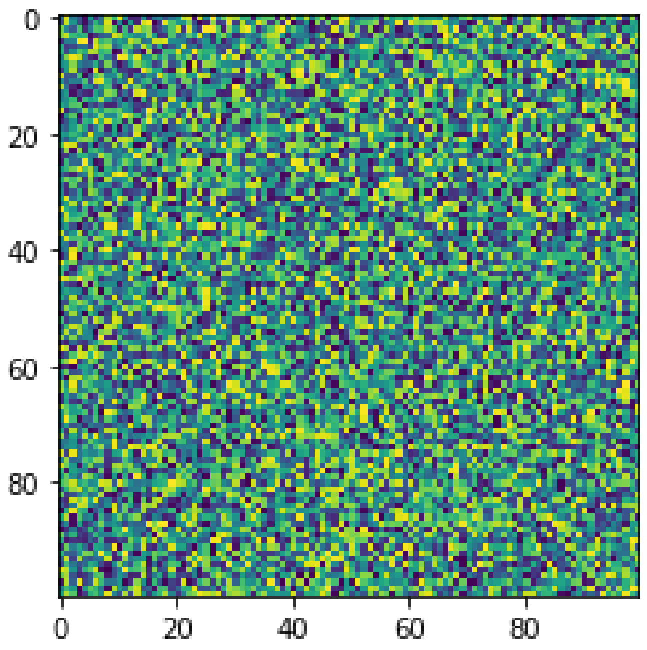
```
dice: [1 3 4 2 5 3 2 6 2 4 1 6]
```

c) Write code to do the following things.

Create a 100 x 100 matrix named mat. Assign any convenient *integer* values to mat, without all elements being identical. (In a later step you will look at the histogram of mat, and create new matrices based on mat.)

[68]:
```
####################################
# TO DO: write the code
# The following is a temporary assignment;
#  try it, just for illustration of imshow(), and then replace with your code
mat=np.random.randint(256,size=(100,100))
####################################
# The next line should display your matrix mat as an image
# Notice that matplotlib.pyplot may color-code the values that are displayed
plt.imshow(mat, interpolation='none')
print(f'shape of mat: {mat.shape}')
```

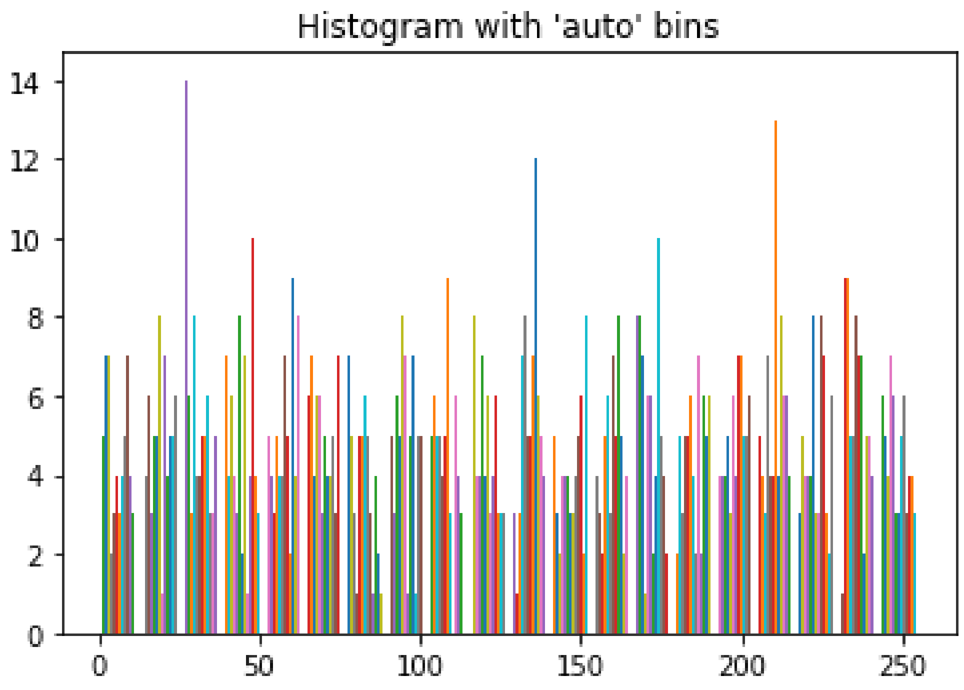
```
shape of mat: (100, 100)
```



Plot a histogram of mat's intensities with 20 bins.

[69]:
```
####################################
# TO DO: write the code
plt.hist(mat, bins= 20)  # arguments are passed to np.histogram
plt.title("Histogram with 'auto' bins")

####################################
```

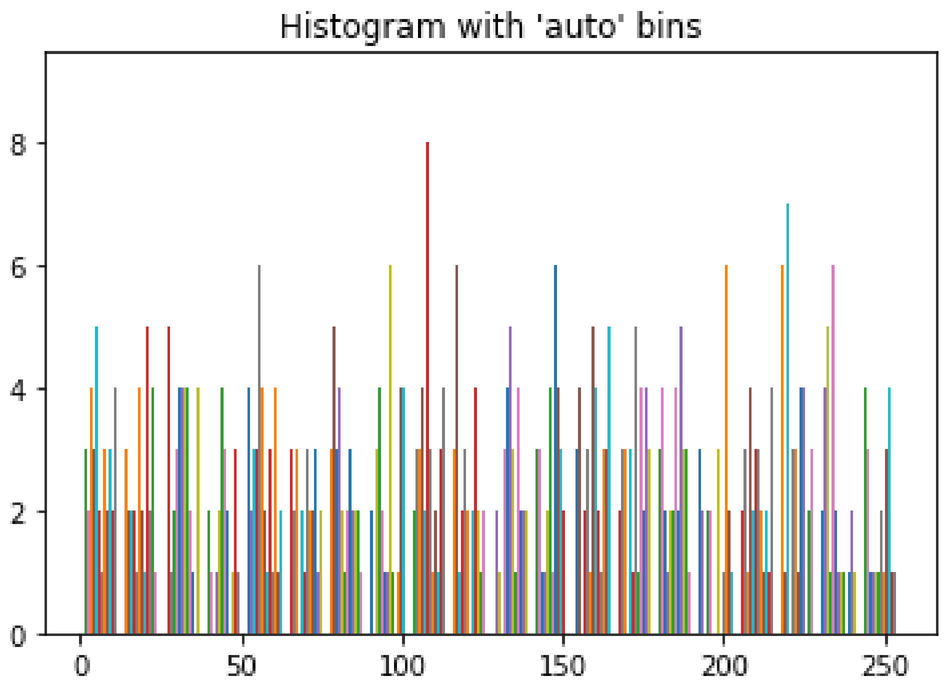
[69]:
```
Text(0.5, 1.0, "Histogram with 'auto' bins")
```



Create a new matrix mat2 that consists of the lower left quadrant of mat.

[70]:
```
###################################
# TO DO: write the code
mat2 =  mat[50:, :50]

plt.hist(mat2, bins= 20)  # arguments are passed to np.histogram
plt.title("Histogram with 'auto' bins")


###################################
```

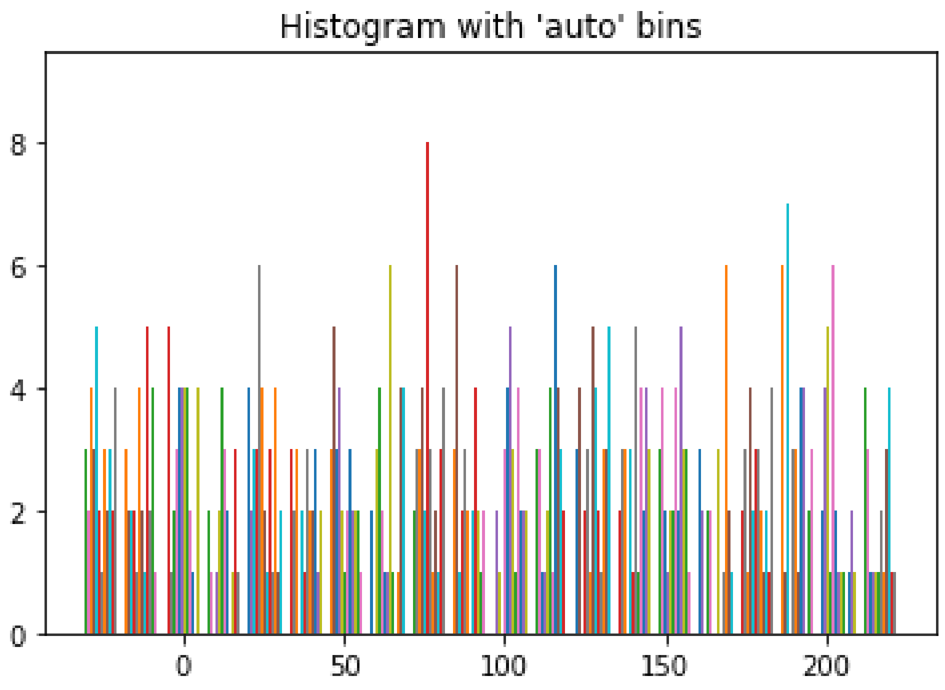
[70]:
```
Text(0.5, 1.0, "Histogram with 'auto' bins")
```



Create a new matrix mat3 that is the same as mat2, but with mat2's mean intensity subtracted from every element. Display mat3 just as the previous examples displayed mat and mat2.

[71]:
```
###################################
# TO DO: write the code
a=mat2.sum()
mean_int=a/10000
mat3=mat2-mean_int
plt.hist(mat3, bins= 20)  # arguments are passed to np.histogram
plt.title("Histogram with 'auto' bins")

###################################
```

[71]:
```
Text(0.5, 1.0, "Histogram with 'auto' bins")
```



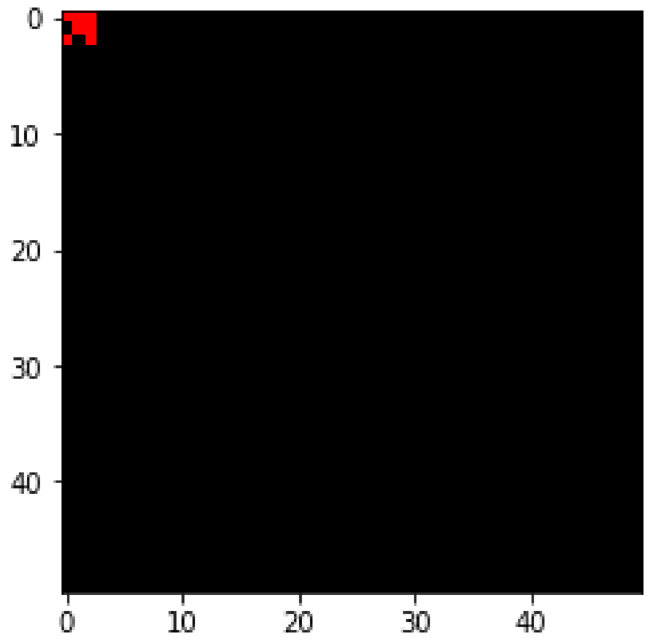
Create a new matrix mat4 that represents a color image that is the same size as mat3, but with 3 channels to represent Red, Green, and Blue values. Set the values in mat4 to be red (i.e., R = 255, G = 0, B = 0) wherever the intensity in mat3 is greater than a threshold t = the average intensity in mat2, and black everywhere else. Be careful with type-casting. Display mat4 just as was done for the previous examples.

[72]:
```
###################################
# TO DO: write the code

mat4=np.zeros((50,50,3))
#print(mat3)
#print(mean_int)
for i in range(3):
  for j in range(3):
    if mat3[i][j]>mean_int:
      mat4[i][j]=[255,0,0]
#print(mat4)
plt.imshow(mat4,interpolation='none')
#mat4=np.where(mat4==[255,255,255],[255,0,0],0)
print('check')
print('mean int is:')
print(mean_int)
print('mat3[1][1] is:')
print(mat3[1][1])
print('mat4[1][1] is:')
print(mat4[1][1])
```

```python
print('mat3[2][2] is:')
print(mat3[2][2])
print('mat4[2][2] is:')
print(mat4[2][2])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
check
mean int is:
31.9073
mat3[1][1] is:
53.0927
mat4[1][1] is:
[255.   0.   0.]
mat3[2][2] is:
59.0927
mat4[2][2] is:
[255.   0.   0.]
```

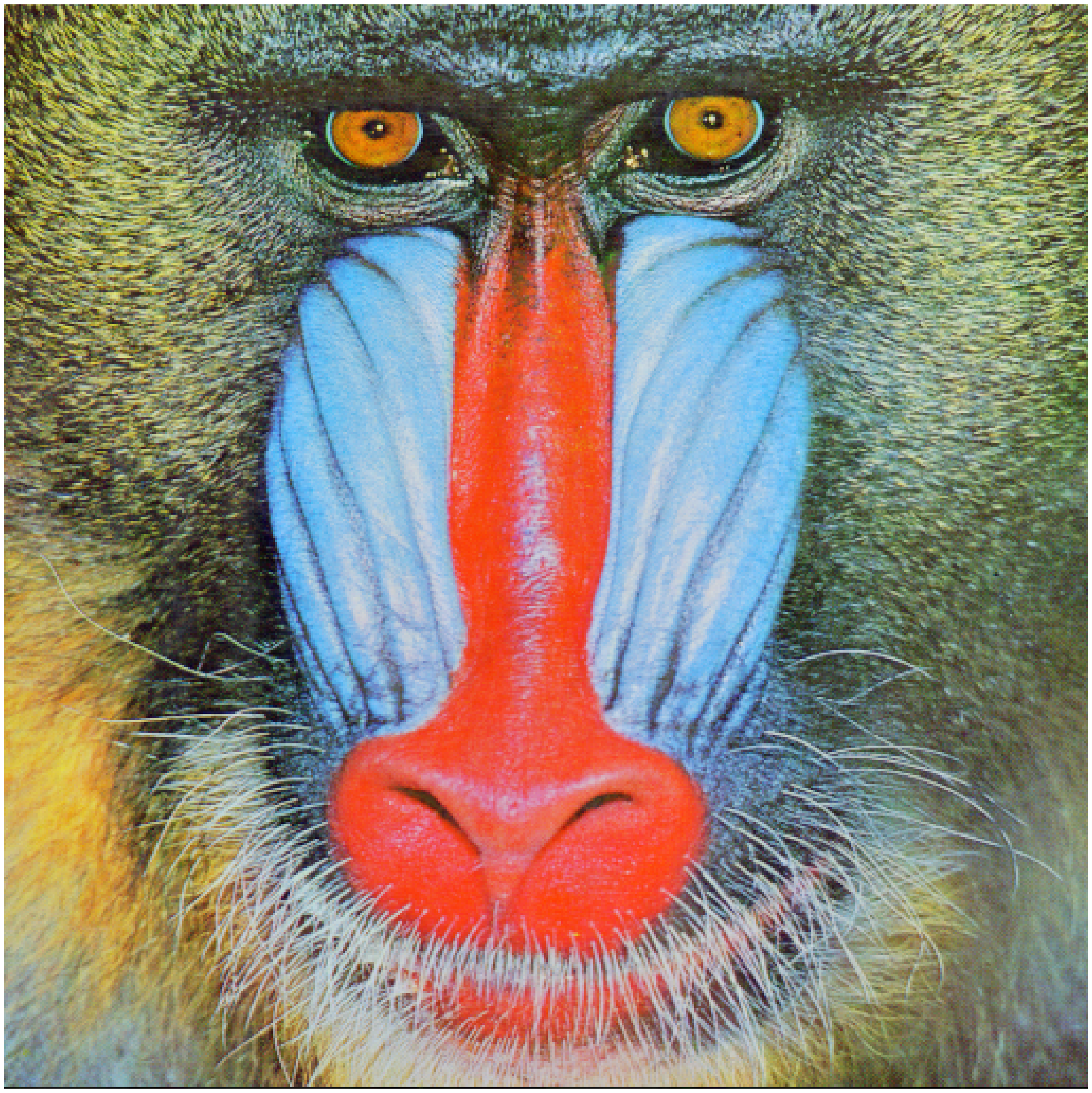


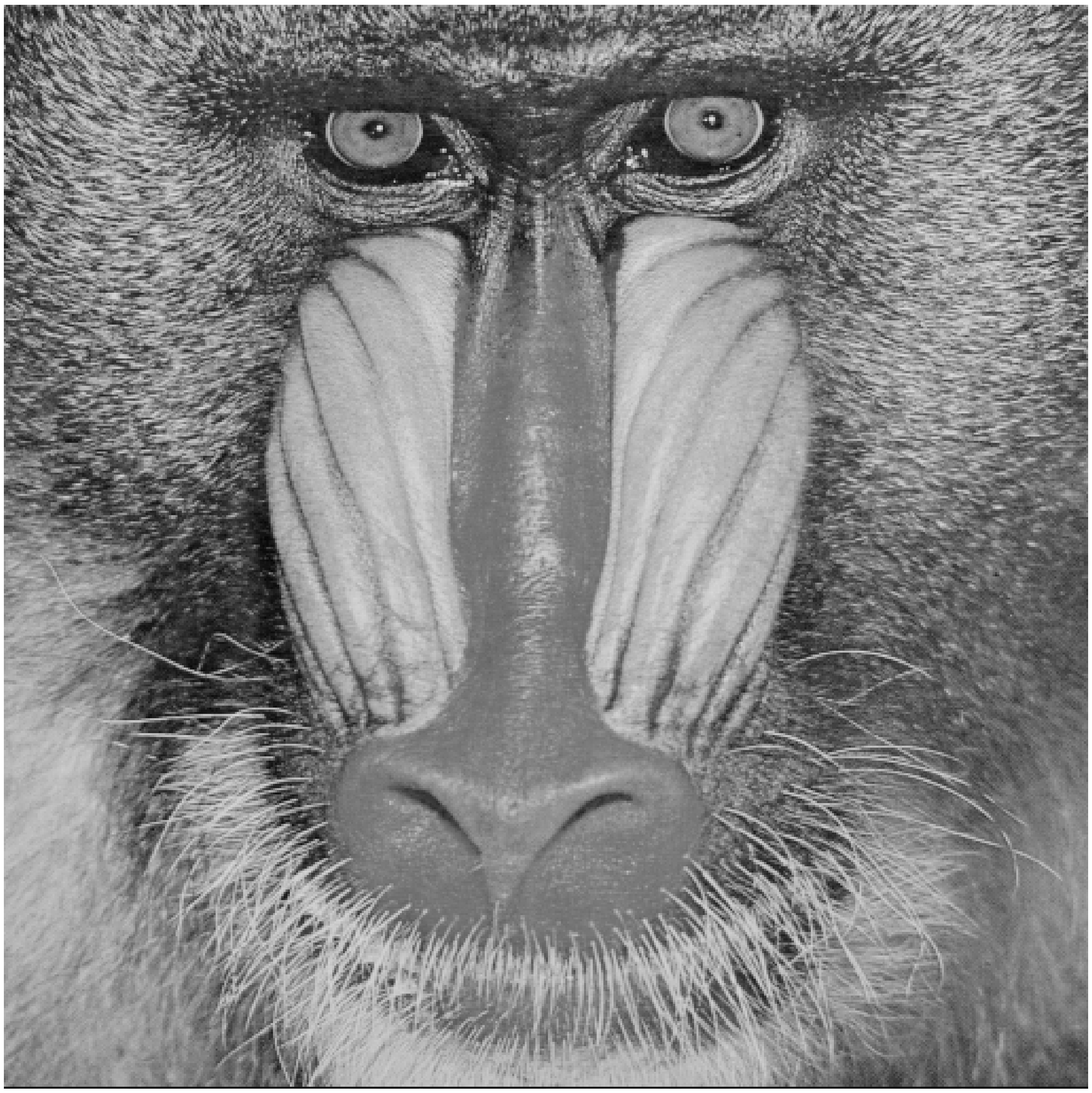
4 Problem 6: Generating new images (10 points)

Write Python/OpenCV code that will input an image, and then create new images to simulate reduced resolution similar to Dr. Jones's example (family portrait) in the first set of lecture slides.

First, verify that you can read an image and convert it to grayscale format. (You must first upload mandrill.tif to your working directory)

[73]:
```python
# This example uses cv2_imshow to display an image
# (Note: cv2.imshow is not allowed in Colab)
from google.colab.patches import cv2_imshow

img_color = cv2.imread("mandrill.tif", cv2.IMREAD_COLOR)
cv2_imshow(img_color)

print ('\n')
img_grayscale = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
cv2_imshow(img_grayscale)
```





Next, write a Python function that accepts a grayscale image as input. You may assume that the input image is of size 512 x 512. The function must iteratively create and display new images *of the same size*. (All new images should be of size 512 x 512.) However, in these new images, groups of pixels appear to be "merged" to form larger pixels. Each new image *appears* to have half as many pixels in both height and width: 512x512 --> 256x256 --> 128x128 --> . . . --> 4x4 --> 2x2 --> 1x1. To go from apparent image size N x N to N/2 x N/2, your code should simply compute the average intensity from 2x2 pixel blocks in the first image, and assign that average value to all of the cooresponding pixels in the next image.

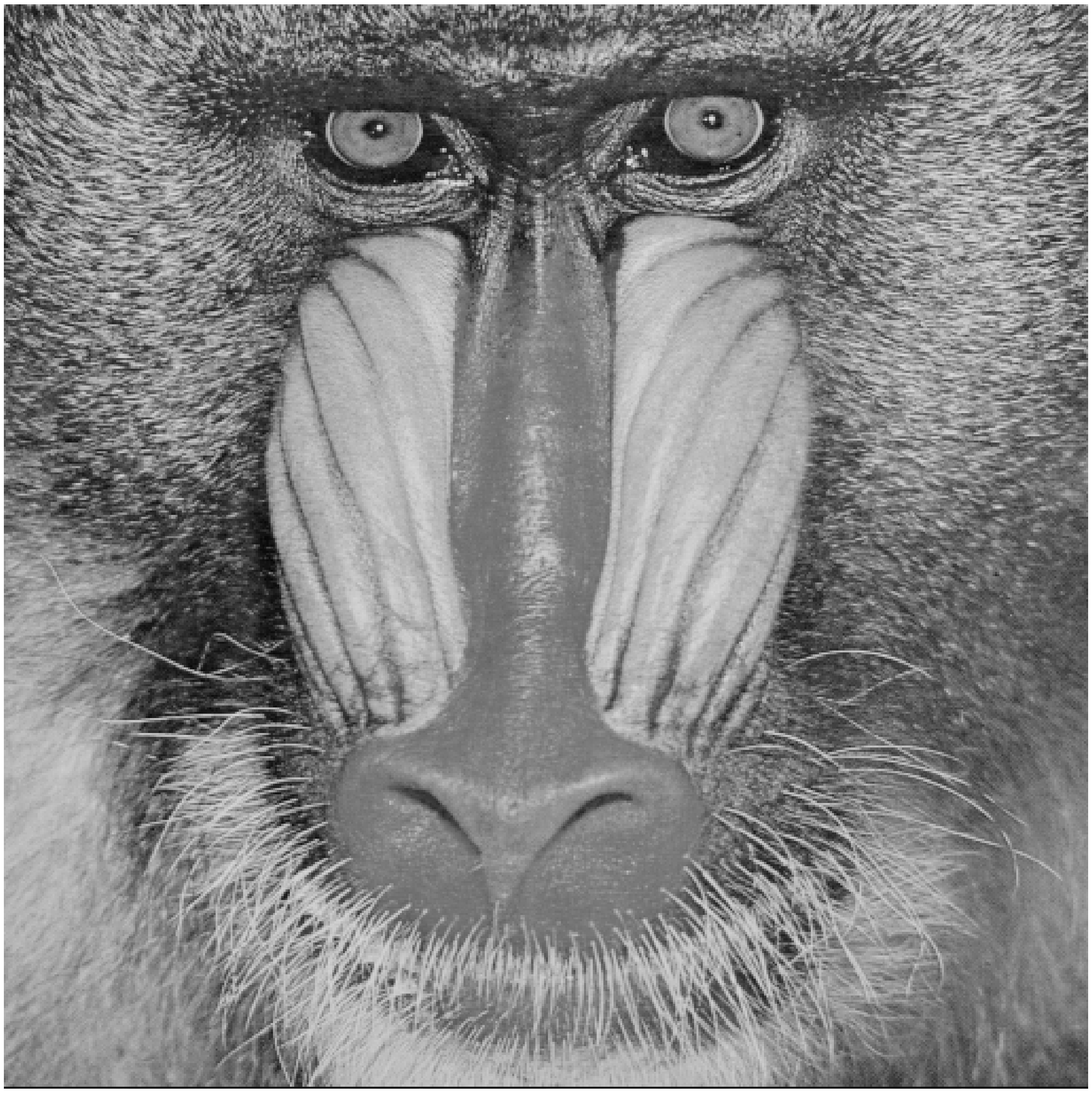
For full credit, use a loop to generate the new images.

[74]:
```
def simulate_lower_resolution(img):
  x= img
  print('x size is:')
  print(x.shape)
  cv2_imshow(img) # temporary statement; replace with your own code
  a=int((len(x)-2)/2)
  sum=0
  leny=int(len(x)/2)
  y=np.zeros((leny,leny))

  for k in range(9):
    for i in range(a+1):
      for j in range(a+1):
        sum = x[2*i][2*j]+x[2*i+1][2*j]+x[2*i][2*j+1]+x[2*i+1][2*j+1]
        y[i][j]=sum/4
        #print('*************************')
        #print(y)
    x=y
    #print('x is:')
    #print(x)
    print('x size is:')
    print(x.shape)
    a=int((len(x)-2)/2)
    lenz=int(len(x)/2)
    z=np.zeros((lenz,lenz))
    y=z
    cv2_imshow(cv2.resize(x,(512,512)))
    print('********************')

  return
```

Test your function. The output should be a display of 10 images, starting with the grayscale image at full resolution, and ending with an image that appears to be a single, very large pixel (one gray value).

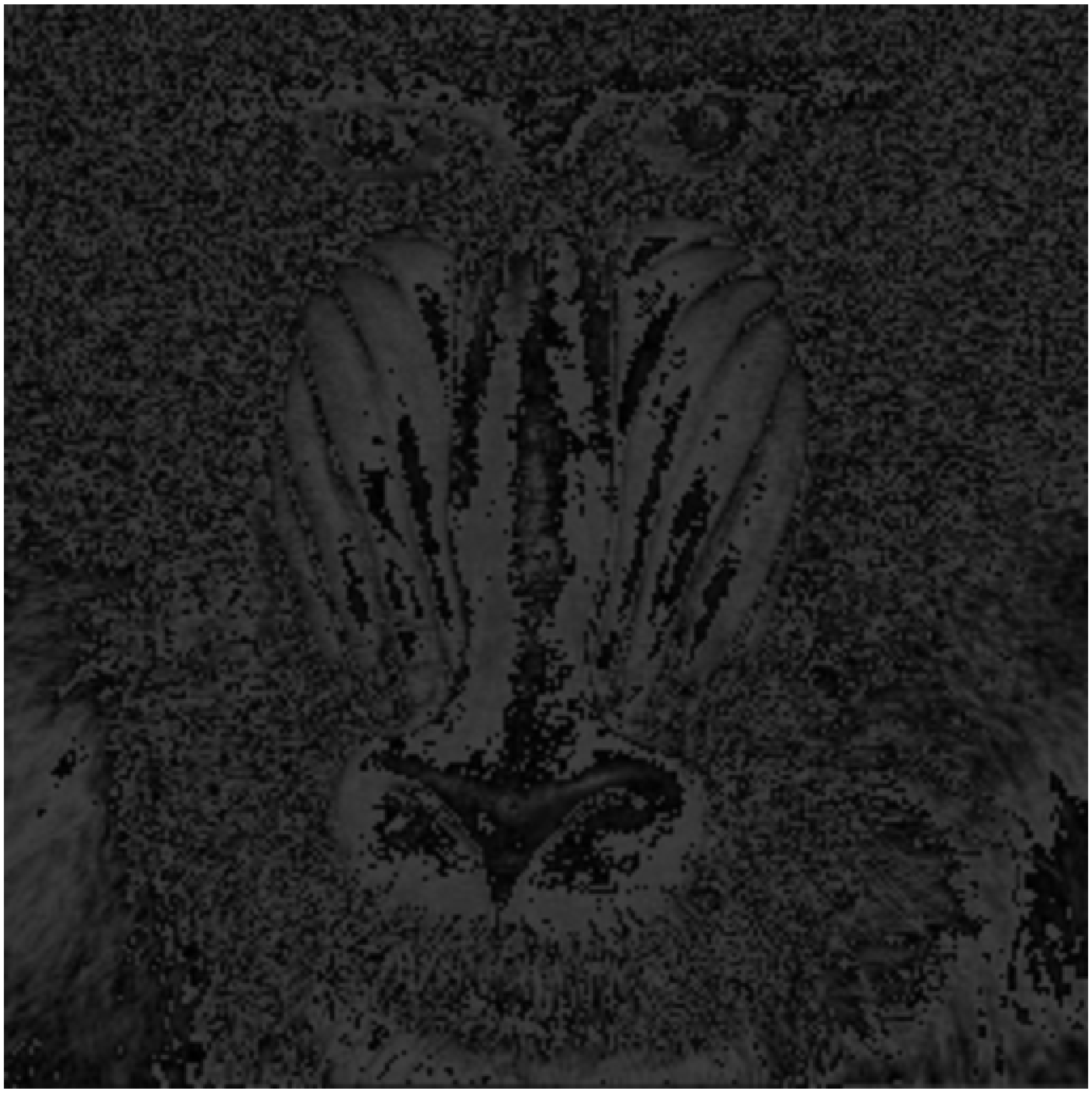
[75]:
```
simulate_lower_resolution(img_grayscale)
```

```
x size is:
(512, 512)
```

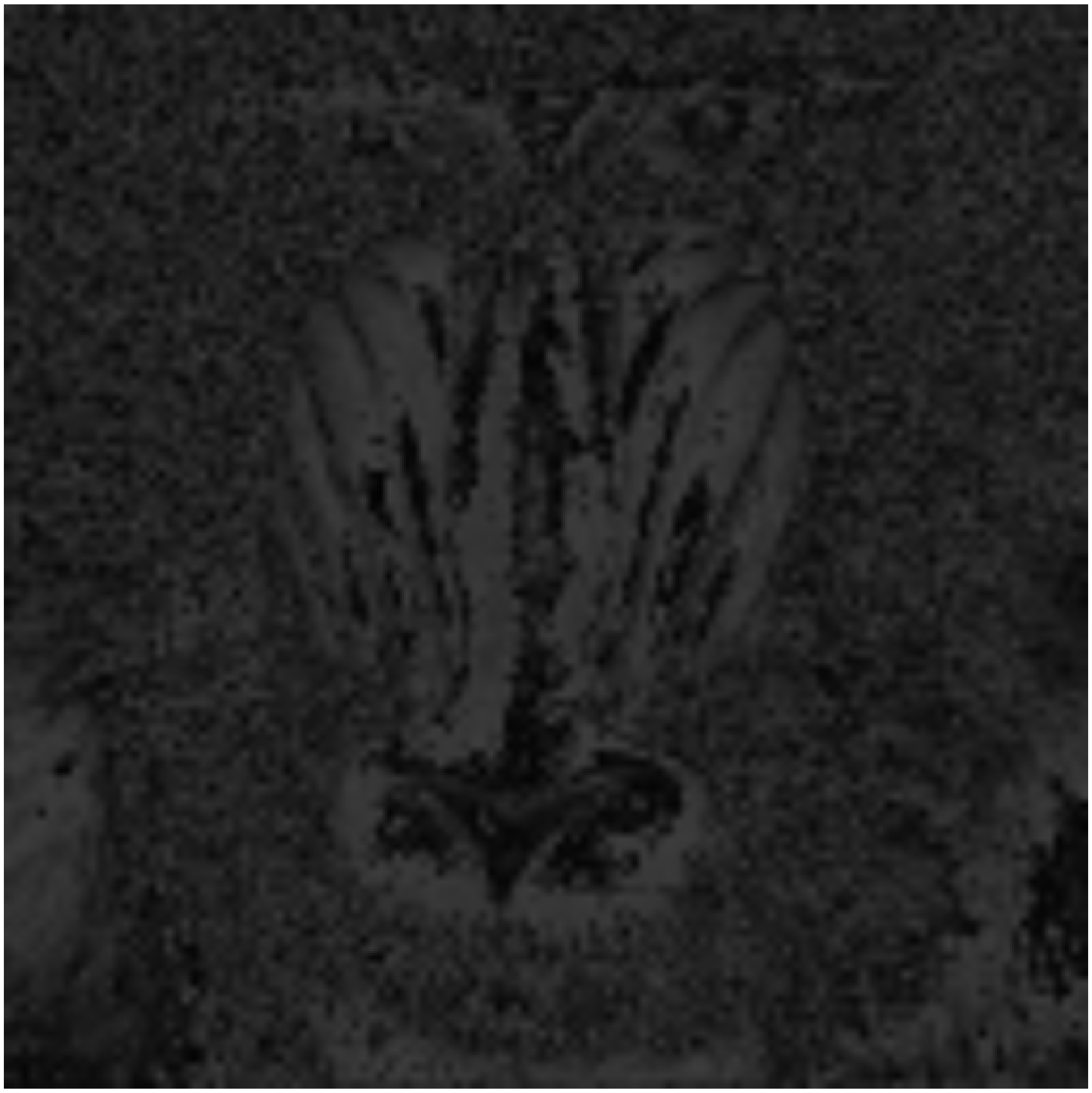


```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: RuntimeWarning:
overflow encountered in ubyte_scalars
```

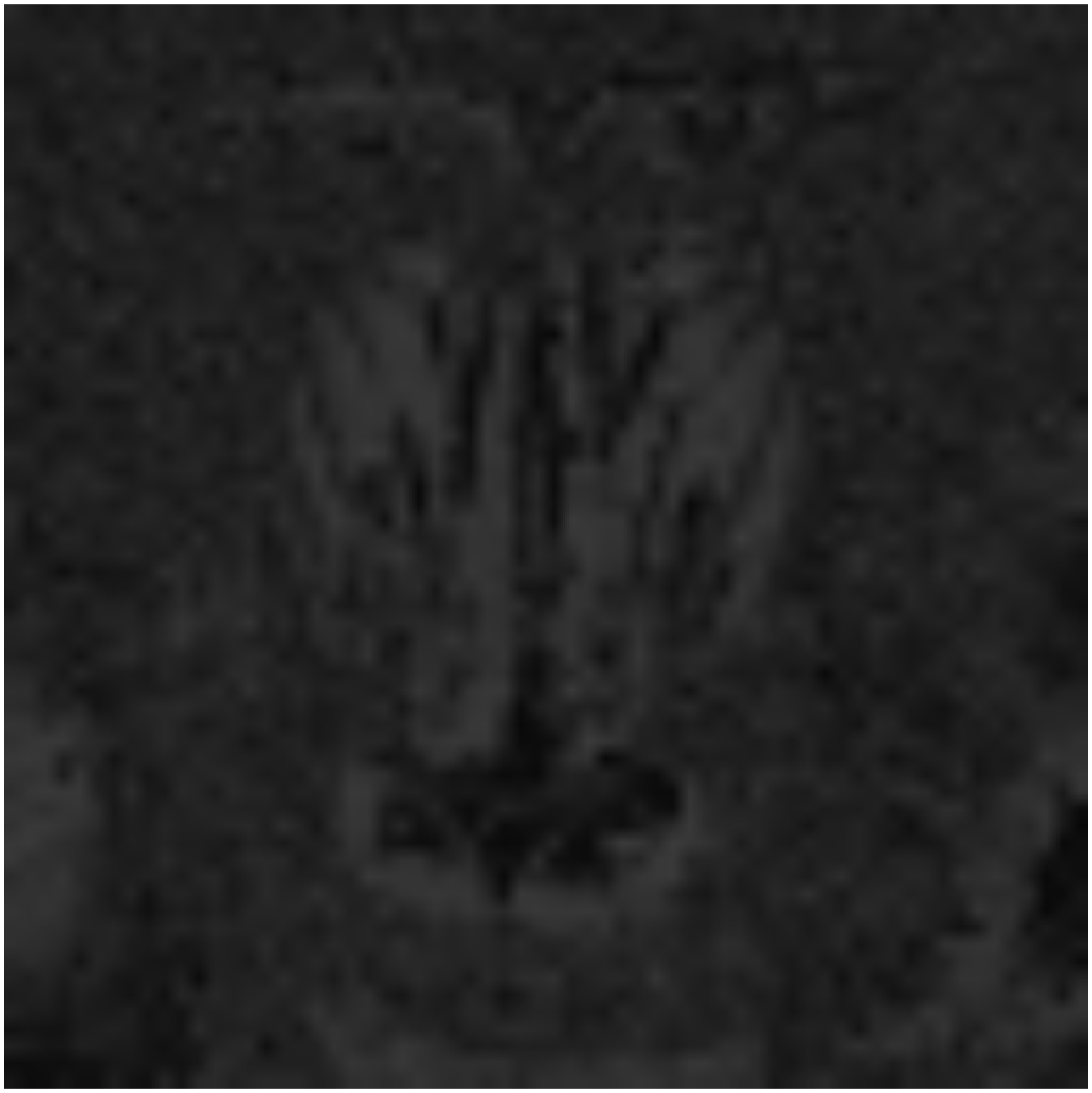
```
x size is:
(256, 256)
```



```
********************
x size is:
(128, 128)
```



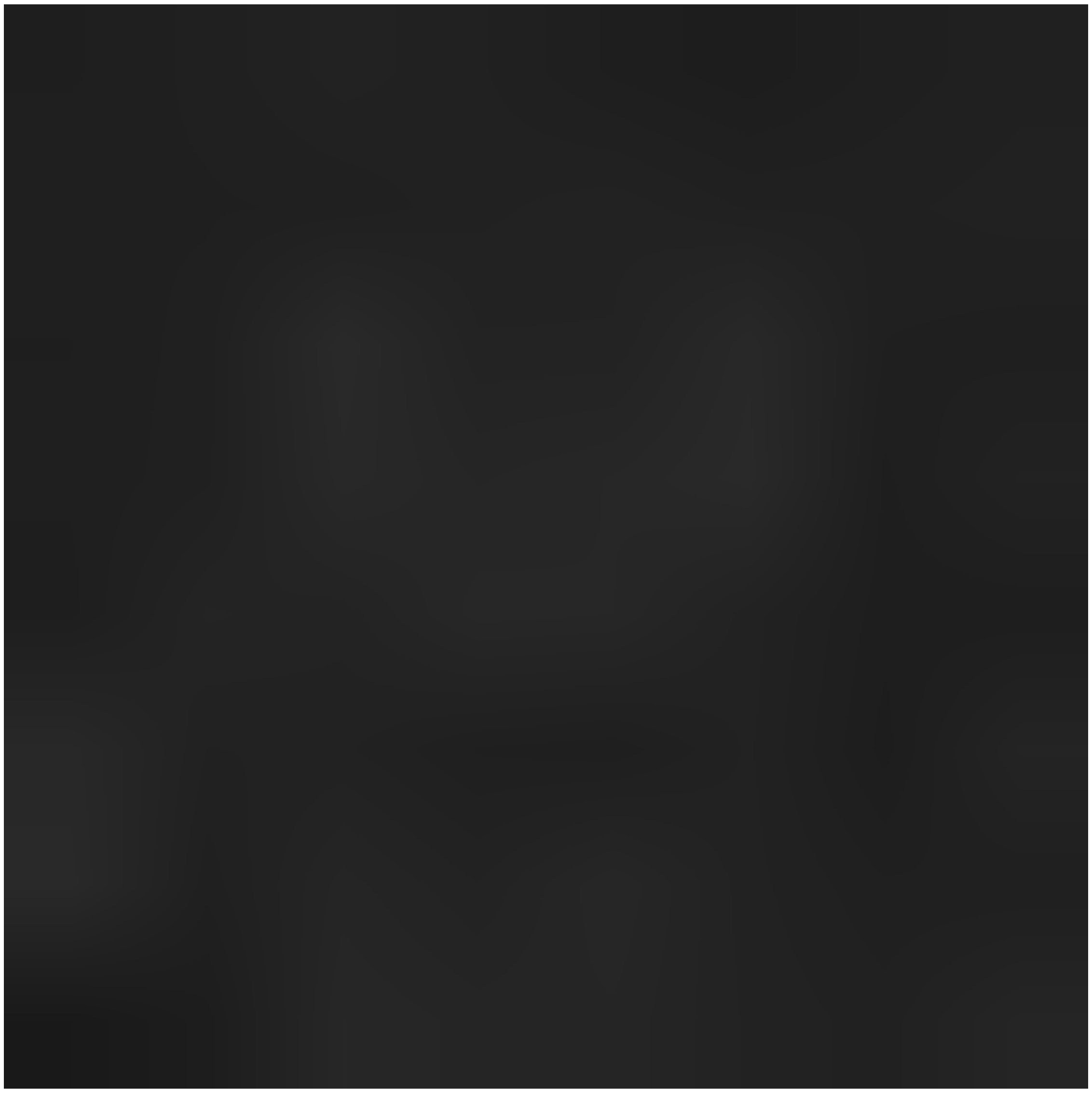
```
********************
x size is:
(64, 64)
```



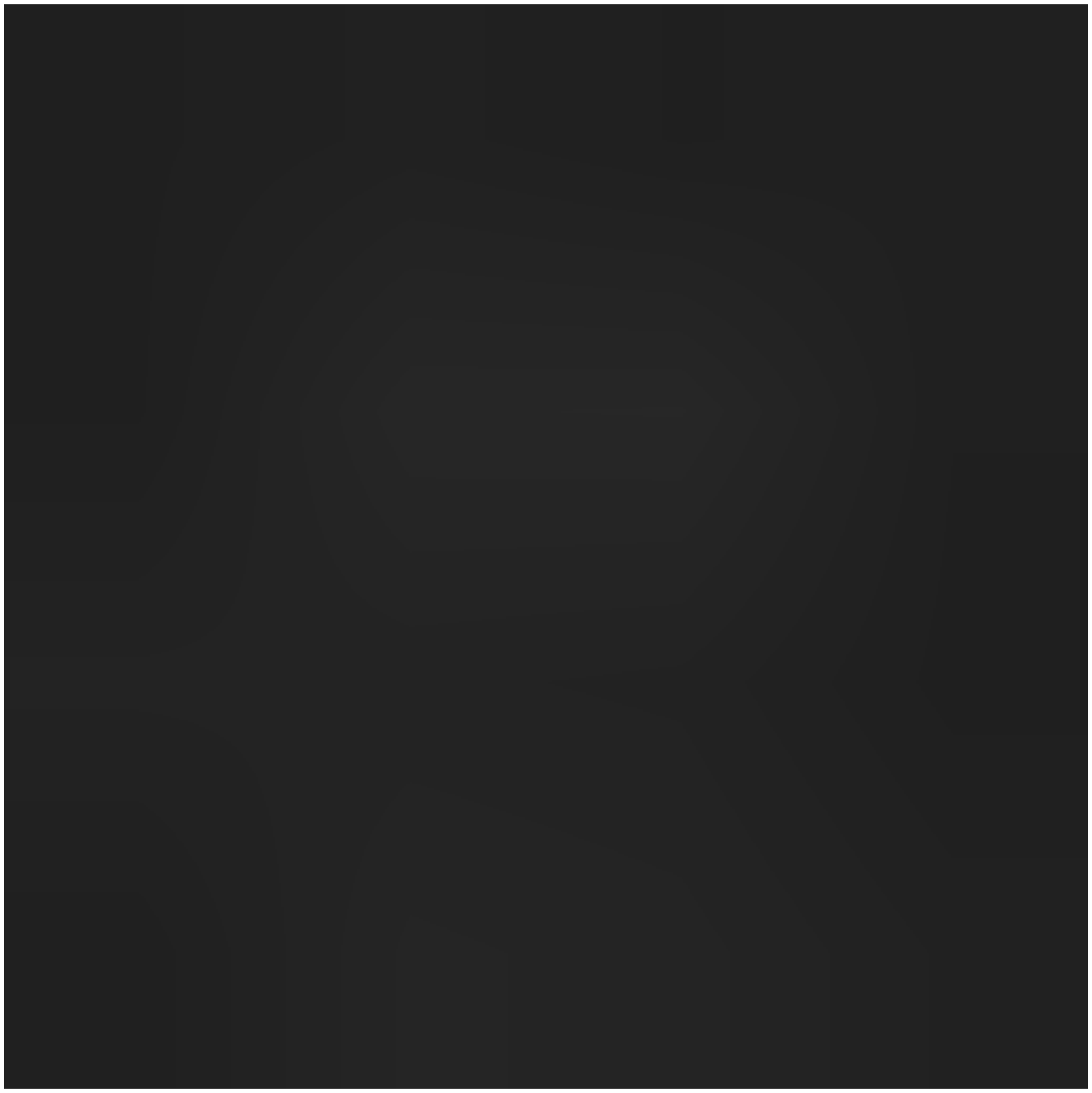
```
********************
x size is:
(32, 32)
```

```
********************
x size is:
(16, 16)
```

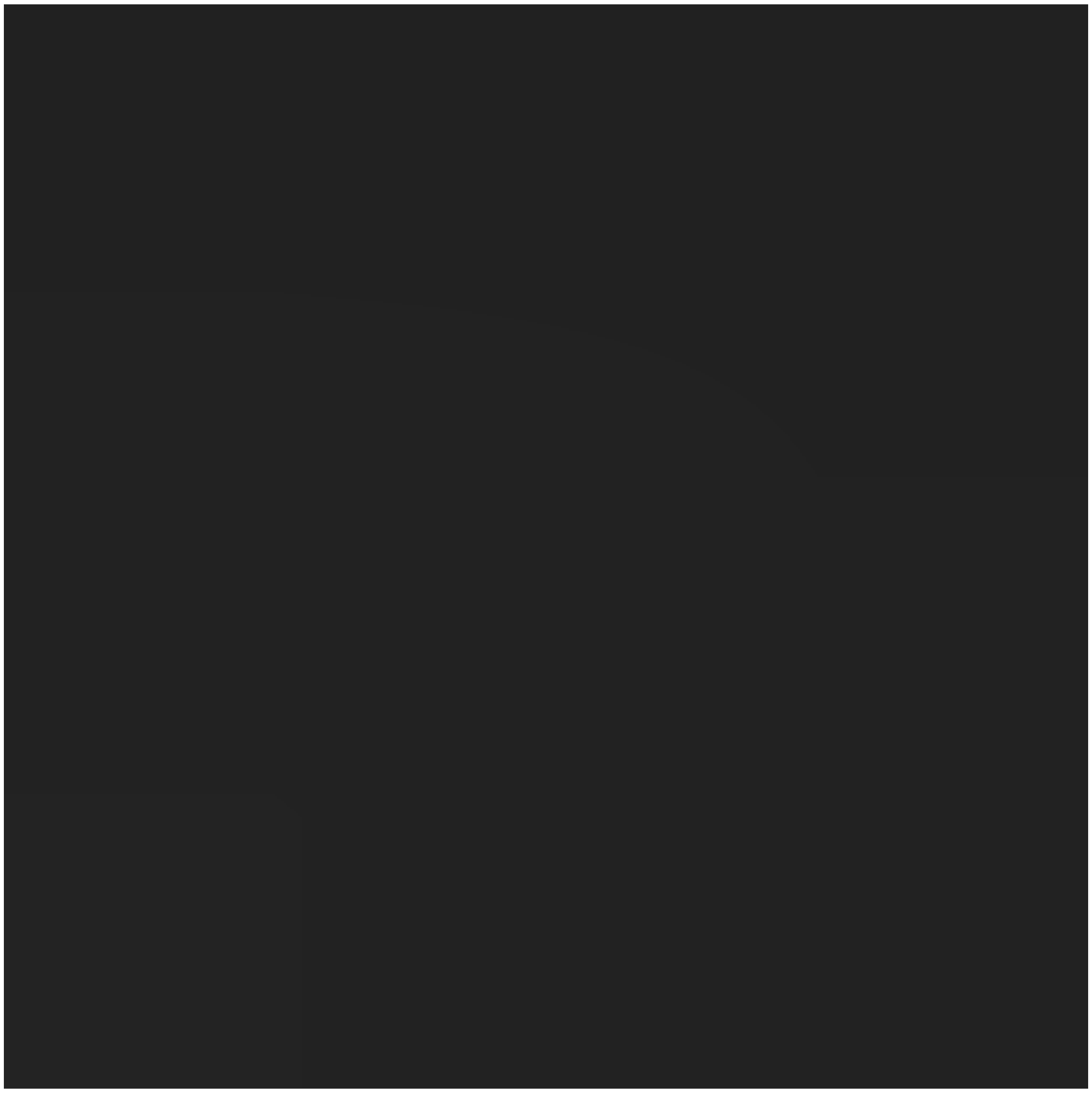
```
********************
x size is:
(8, 8)
```



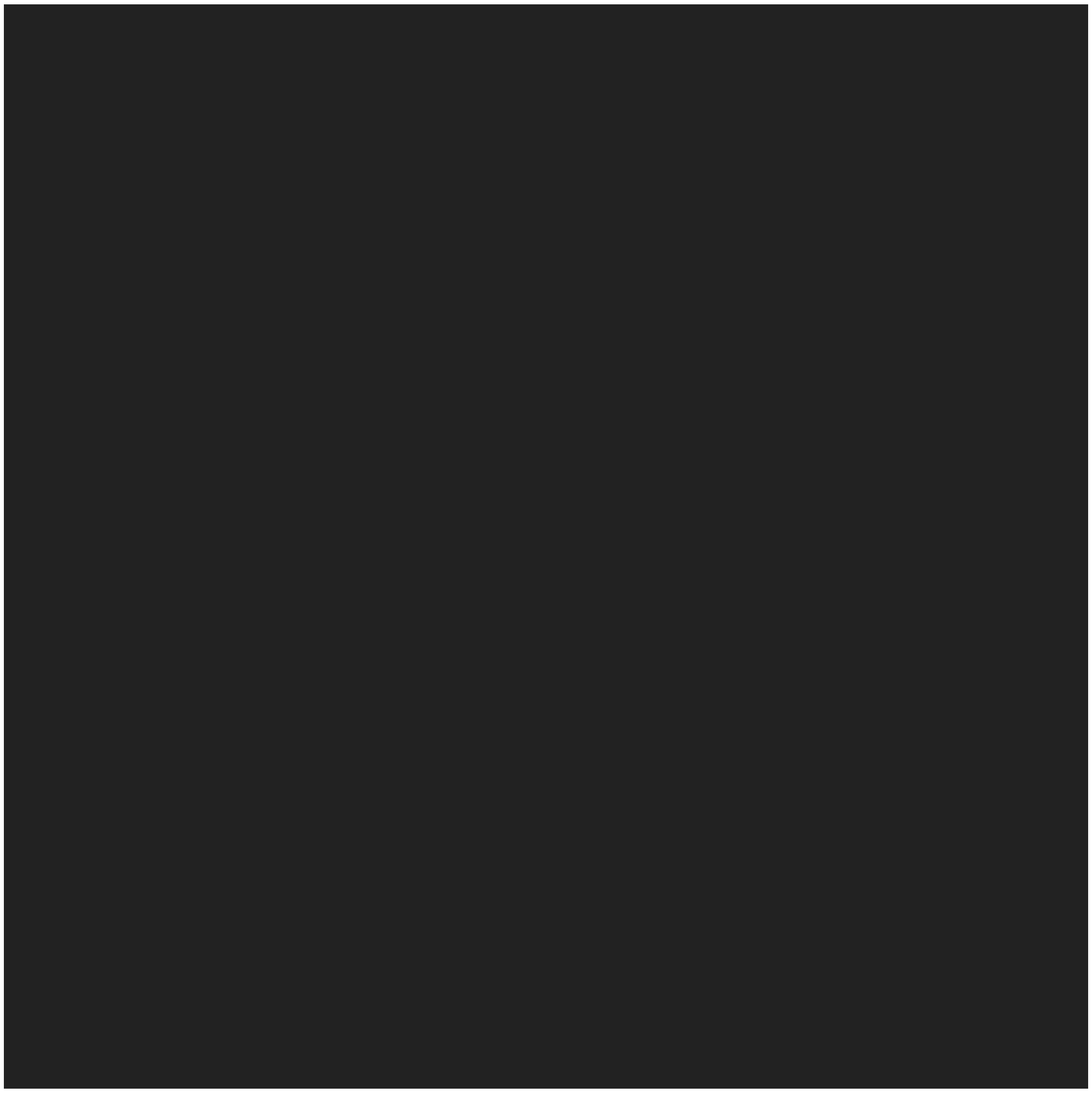
```
********************
x size is:
(4, 4)
```



```
********************
x size is:
(2, 2)
```



```
********************
x size is:
(1, 1)
```



****

---

5 Creating a PDF version of your current notebook

[76]:
```
#The following two installation steps are needed to generate a PDF version of␣
 ↪the notebook
#(These lines are needed within Google Colab, but are not needed within a local␣
 ↪version of Jupyter notebook)
!apt-get -qq install texlive texlive-xetex texlive-latex-extra pandoc
!pip install --quiet pypandoc
```

[77]:
```
# TO DO: Provide the full path to your Jupyter notebook file
!jupyter nbconvert --to PDF "/content/drive/MyDrive/ECE5554/HW1/
 ↪Homework1_denizaytemiz.ipynb"
#!jupyter nbconvert --to PDF "/content/drive/MyDrive/Colab Notebooks/
 ↪Homework1_USERNAME.ipynb"
```

```
[NbConvertApp] WARNING | pattern
u'/content/drive/MyDrive/ECE5554/HW1/Homework1_USERNAME.ipynb' matched no files
This application is used to convert notebook files (*.ipynb) to various other
formats.

WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
-------

Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

--execute
    Execute the notebook prior to export.
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--stdout
    Write notebook output to stdout instead of files.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
-y
    Answer yes to any questions instead of prompting.
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--debug
    set log level to logging.DEBUG (maximize logging output)
--no-prompt
    Exclude input and output prompts from converted document.
--generate-config
```

```
    generate default config file
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    Choices: [1, 2, 3, 4]
    The nbformat version to write. Use this to downgrade notebooks.
--output-dir=<Unicode> (FilesWriter.build_directory)
    Default: ''
    Directory to write output(s) to. Defaults to output to the directory of each
    notebook. To recover previous default behaviour (outputting to the current
    working directory) use . as the flag value.
--writer=<DottedObjectName> (NbConvertApp.writer_class)
    Default: 'FilesWriter'
    Writer class used to write the  results of the conversion
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL')
    Set the log level by value or name.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
    Default: u''
    The URL prefix for reveal.js (version 3.x). This defaults to the reveal CDN,
    but can be any url pointing to a copy  of reveal.js.
    For speaker notes to work, this must be a relative path to a local  copy of
    reveal.js: e.g., "reveal.js".
    If a relative path is given, it must be a subdirectory of the current
    directory (from which the server is run).
    See the usage documentation
    (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-
    slideshow) for more details.
--to=<Unicode> (NbConvertApp.export_format)
    Default: 'html'
    The export format to be used, either one of the built-in formats
    ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
    'python', 'rst', 'script', 'slides'] or a dotted object name that represents
    the import path for an `Exporter` class
--template=<Unicode> (TemplateExporter.template_file)
    Default: u''
    Name of the template file to use
--output=<Unicode> (NbConvertApp.output_base)
    Default: ''
    overwrite base name use for output files. can only be used when converting
    one notebook at a time.
--post=<DottedOrNone> (NbConvertApp.postprocessor_class)
    Default: u''
    PostProcessor class used to write the results of the conversion
--config=<Unicode> (JupyterApp.config_file)
    Default: u''
    Full path of a config file.
```

```
To see all available configurables, use `--help-all`

Examples
--------

    The simplest way to use nbconvert is

    > jupyter nbconvert mynotebook.ipynb

    which will convert mynotebook.ipynb to the default format (probably HTML).

    You can specify the export format with `--to`.
    Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

    > jupyter nbconvert --to latex mynotebook.ipynb

    Both HTML and LaTeX support multiple output templates. LaTeX includes
    'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
    can specify the flavor of the format used.

    > jupyter nbconvert --to html --template basic mynotebook.ipynb

    You can also pipe the output to stdout, rather than a file

    > jupyter nbconvert mynotebook.ipynb --stdout

    PDF is generated via latex

    > jupyter nbconvert mynotebook.ipynb --to pdf

    You can get (and serve) a Reveal.js-powered slideshow

    > jupyter nbconvert myslides.ipynb --to slides --post serve

    Multiple notebooks can be given at the command line in a couple of
    different ways:

    > jupyter nbconvert notebook*.ipynb
    > jupyter nbconvert notebook1.ipynb notebook2.ipynb

    or you can specify the notebooks list in a config file, containing::

        c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

    > jupyter nbconvert --config mycfg.py
```