

# SciComp with Py

## CVIP

### Basic Image Processing with OpenCV

Vladimir Kulyukin  
Department of Computer Science  
Utah State University



# Outline

- OpenCV Background
- Basic Image Processing w/ OpenCV



# OpenCV Background



# OpenCV

- OpenCV ([opencv.org](https://opencv.org)) is a open source CV library
- Free for both commercial and academic use under a BSD license
- OpenCV has C++, Java, & Python interfaces
- Available on Linux, Mac OS, Android, iOS, Windows



# Image Formats

- Image formats are different standards of storing digital images
- Broadly speaking, there are three kinds of data storage: compressed, uncompressed, and vector
- There are two types of compression: lossless and lossy
- Common formats: JPEG (compressed, lossy), BMP (uncompressed, lossless), PNG (compressed, lossless), SVG (scalable vector graphics)
- OpenCV supports all common formats



# Pixels

- Pixel is the smallest addressable element in a raster image, i.e., a matrix of pixels
- Each pixel is a sample of an original image
- Pixels are typically represented as 3-tuples (red, green, blue) or (blue, green red)
- Number of representable colors is denoted by bits per pixel:  
 $1 \text{ bpp} = 2^1 = 2$ ;  $2 \text{ bpp} = 2^2 = 4$ ;  $2^3 = 8$  , etc.



## Switching to CV Workspace

If you want to use OpenCV on your pi within the cv workspace, do:

```
$ source ~/.profile
```

```
$ workon cv
```



# Working with CV2.SO

- Another way to ensure that your Python programs run with OpenCV is to place cv2.so into your current working directory
- cv2.so is the share object file installed on your machine when you install OpenCV from scratch





# Checking OpenCV Installation

```
$ python
```

```
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
```

```
[GCC 4.8.2] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import cv2
```

```
>>> cv2.__version__
```

```
'3.0.0'
```

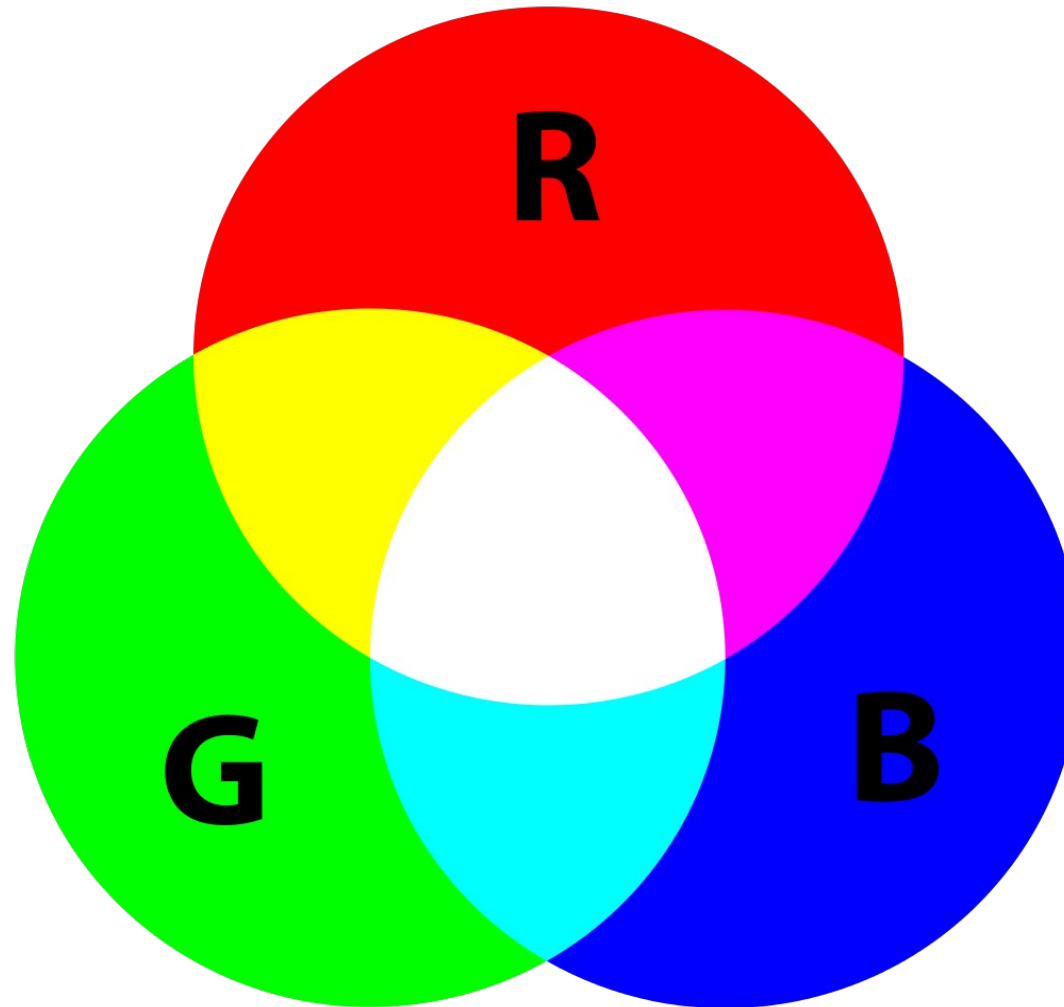
```
>>>
```



# Basic IP



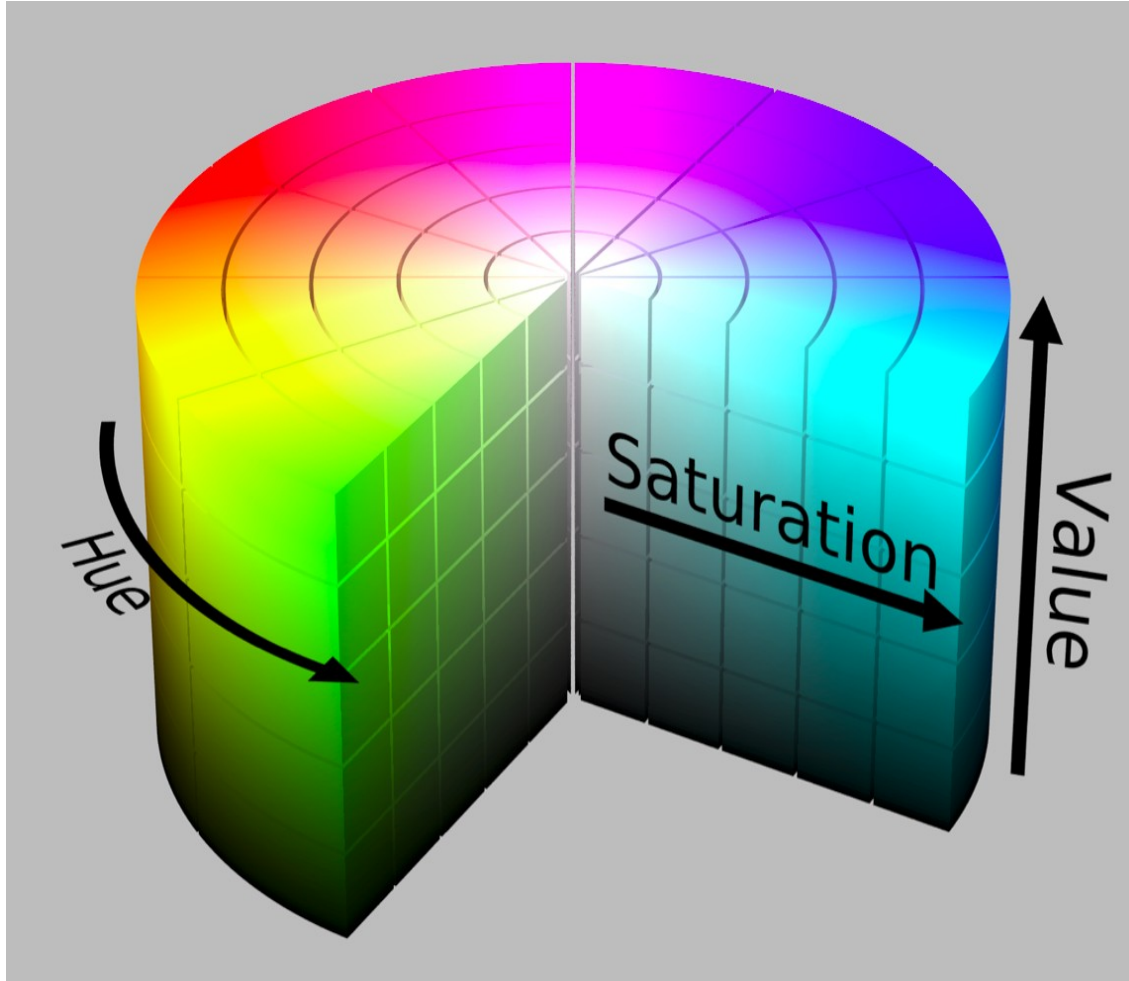
# RGB Color Space



In OpenCV, the standard pixel representation is B, G, R, e.g., [10, 234, 50]. The values of B, G, R are in [0, 255]. [0, 0, 0] is black; [255, 255, 255] is white.



# HSV Color Space

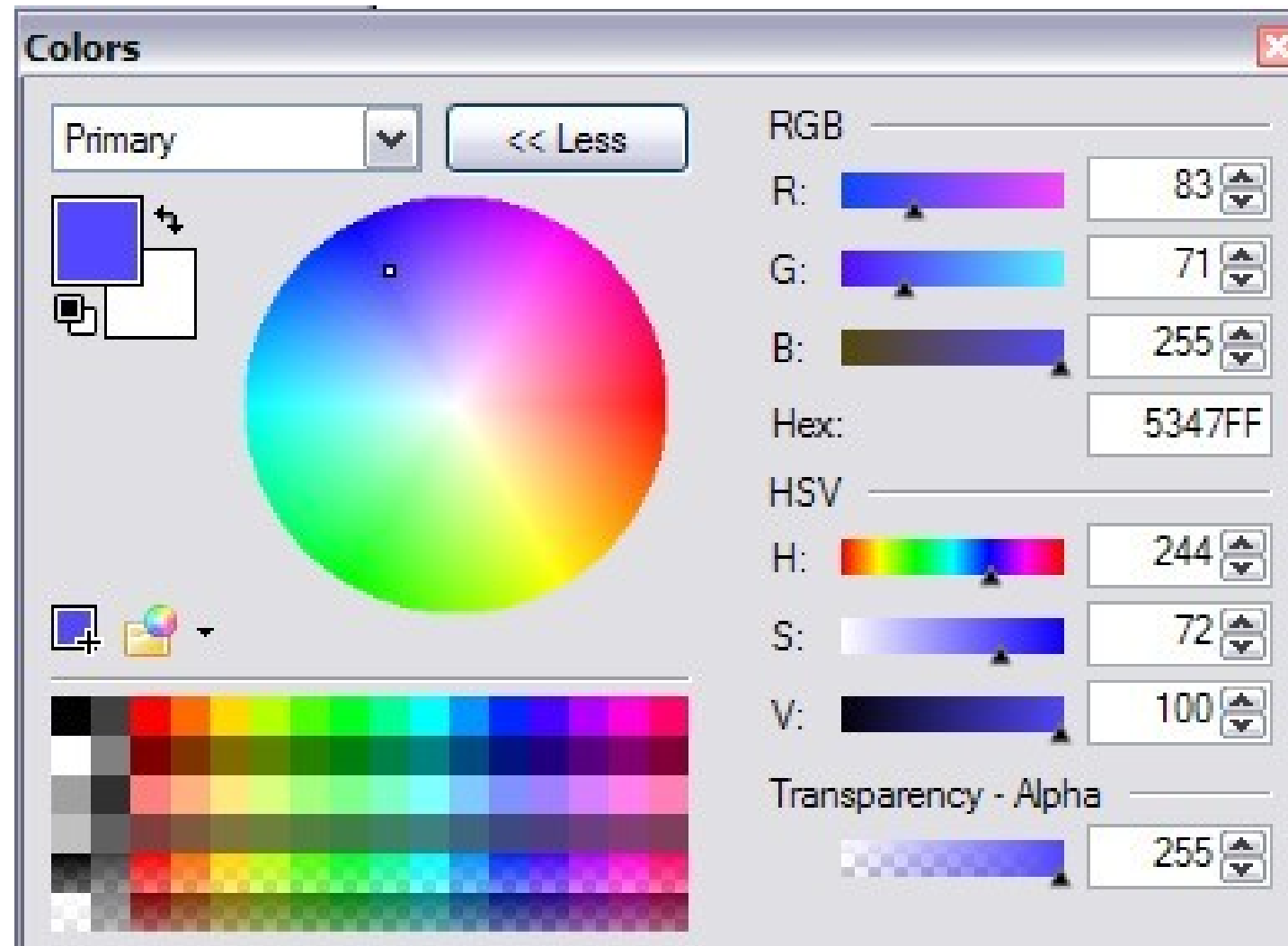


- Hue is color value [0, 179]
- Saturation is color vibrancy [0, 255]; at lower saturation in the center everything is white
- Value is brightness/intensity of color [0, 255]; it goes from dark (below) to bright (above)



# RGB & HSV Spaces

It is possible to map an RGB point to a HSV point and vice versa;  
most image processing applications have this option



# Problem

Write a program that loads a user specified image, converts it into grayscale, displays it in a window, and waits for the user to press a key before closing the window.

## Sample Call

```
$ python load_image.py -i truck.jpg
```



# Loading Images

load\_image.py

```
import argparse
import cv2

ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required = True, help = 'Path to image')
args = vars(ap.parse_args())

image = cv2.imread(args['image'])

cv2.imshow('Image', image)
cv2.waitKey(0)
```

Parse user args

Load image from user-specified file

Show Image

Wait for user to press a key



# Problem

Write a program that loads a user specified image, prints out the shape of the image (height, width, number of channels), displays the image in a window, and waits for the user to press a key before closing the window.

## Sample Call

```
$ python image_shape.py -i truck.jpg
```





# Getting Image's Shape (Height, Width, Num of Channels)

image\_shape.py

```
import argparse
import cv2
```

```
ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required = True, help = 'Path to image')
args = vars(ap.parse_args())
```

```
image = cv2.imread(args['image'])
(h, w, num_channels) = image.shape
print 'h=' + str(h) + '; ' + 'w=' + str(w) + '; ' + 'c=' + str(num_channels)
```

```
cv2.imshow('Loaded Image', image)
cv2.waitKey(0)
```

Load image from user-specified file

Get & print image's shape

Show image



# Problem

Write a program that loads a user specified image, converts it into grayscale, displays the original image and the grayscale image in two windows, and waits for the user to press any key before closing the windows.

## Sample Call

```
$ python grayscale_image.py -i truck.jpg
```



# Grayscale

Grayscale is an operation of converting RGB pixels to grayscale intensity pixels. Here is a commonly used conversion formula:

```
def luminosity(rgb, rcoeff=0.2126, gcoeff=0.7152, bcoeff=0.0722):  
    return rcoeff*rgb[0]+gcoeff*rgb[1]+bcoeff*rgb[2]
```



# Grayscale

## grayscale\_image.py

```
import cv2
import argparse

ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required = True, help = 'Path to image')
args = vars(ap.parse_args())

image = cv2.imread(args['image'])

cv2.imshow('Original', image)
cv2.waitKey()

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow('Grayscaled', gray_image)
cv2.waitKey()

cv2.destroyAllWindows()
```

Grayscale image



# Grayscale

grayscale\_image2.py

```
import cv2
import argparse

ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required = True, help = 'Path to image')
args = vars(ap.parse_args())

image = cv2.imread(args['image'], 0)
cv2.imshow('Grayscaled', image)
cv2.waitKey()

cv2.destroyAllWindows()
```

Grayscale image: the image is grayscaled, because the 2<sup>nd</sup> parameter is 0.



# Problem

Write a program that loads a user specified image, splits it into R, G, B channels, displays each channel in a separate window, and then merges the three channels back and displays the merged image and the original images, displays images where R, G, and B are amplified.

## Sample Call

```
$ python bgr_channels.py -i truck.jpg
```



# Splitting Image into B, G, R Channels

bgr\_channels.py

```
image = cv2.imread(args['image'])
## split the image into 3 channels
B, G, R = cv2.split(image)
## show each channel
print "B's shape:", B.shape
print "G's shape:", G.shape
print "R's shape:", R.shape

cv2.imshow('Red', R)
cv2.imshow('Green', G)
cv2.imshow('Blue', B)
cv2.waitKey(0)
cv2.destroyAllWindows()

## merge B, G, R channels back to get the original image
merged = cv2.merge([B, G, R])
cv2.imshow('Merged', merged)

## amplifying blue by adding 100 to it
amplified_blue = cv2.merge([B+100, G, R])
cv2.imshow('Amplified Blue', amplified_blue)
```



# A Better Way of Splitting Image into B, G, R Channels

split\_merge.py

```
image = cv2.imread(args['image'])

B, G, R = cv2.split(image)

zeros = np.zeros(image.shape[:2], dtype='uint8')

cv2.imshow('Red', cv2.merge([zeros, zeros, R]))
cv2.imshow('Green', cv2.merge([zeros, G, zeros]))
cv2.imshow('Blue', cv2.merge([B, zeros, zeros]))

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Get the height and width  
of the image





# Splitting Image into B, G, R, Channels

```
$ python split_merge.py -i truck.jpg
```



# Blur, Erosion, Dilation



# 3 Big M's

- Mean – is the average of a set of values
- Median – a numerical value  $v$  right in the middle of the sorted sequence of values so that exactly half of the values in the set are less than  $v$  and half are greater than  $v$
- Mode – the most frequent value in a set of values



# Plotting Mean & Median

source in mean\_median\_mode.py

```
import numpy as np

import matplotlib.pyplot as plt

from scipy import stats

## use normal distribution to generate 10,000 points

## centered on 30,000 with an STD = 15,000

incomes = np.random.normal(30000, 15000, 10000)

## if you want to see an outlier, add this

## billionaire's income to the list of incomes

#incomes = np.append(incomes, [10000000000])

mn = np.mean(incomes)

print('mean  = ' + str(mn))

md = np.median(incomes)

print('median = ' + str(md))

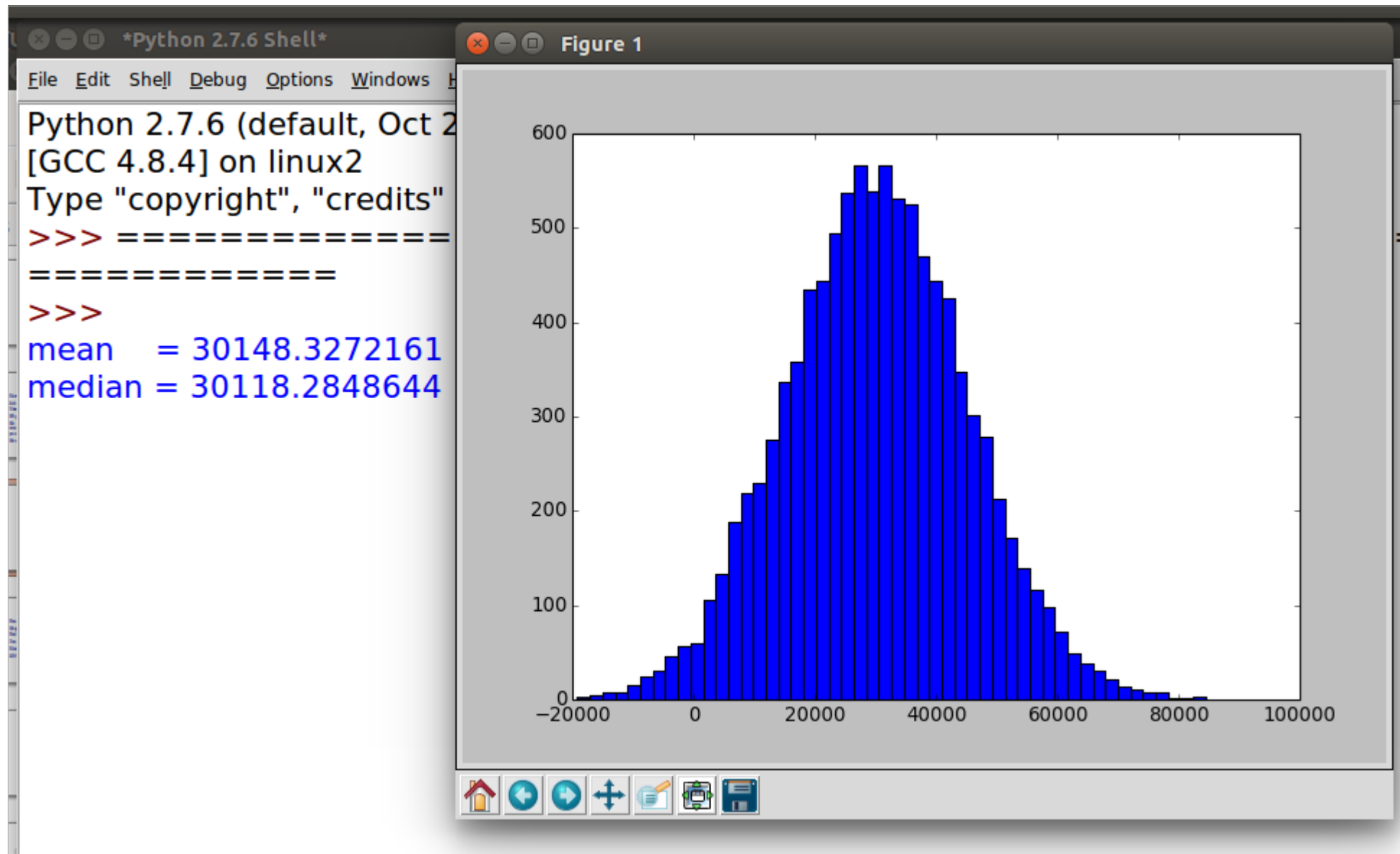
## if you want to see a plot of incomes

plt.hist(incomes, 50)

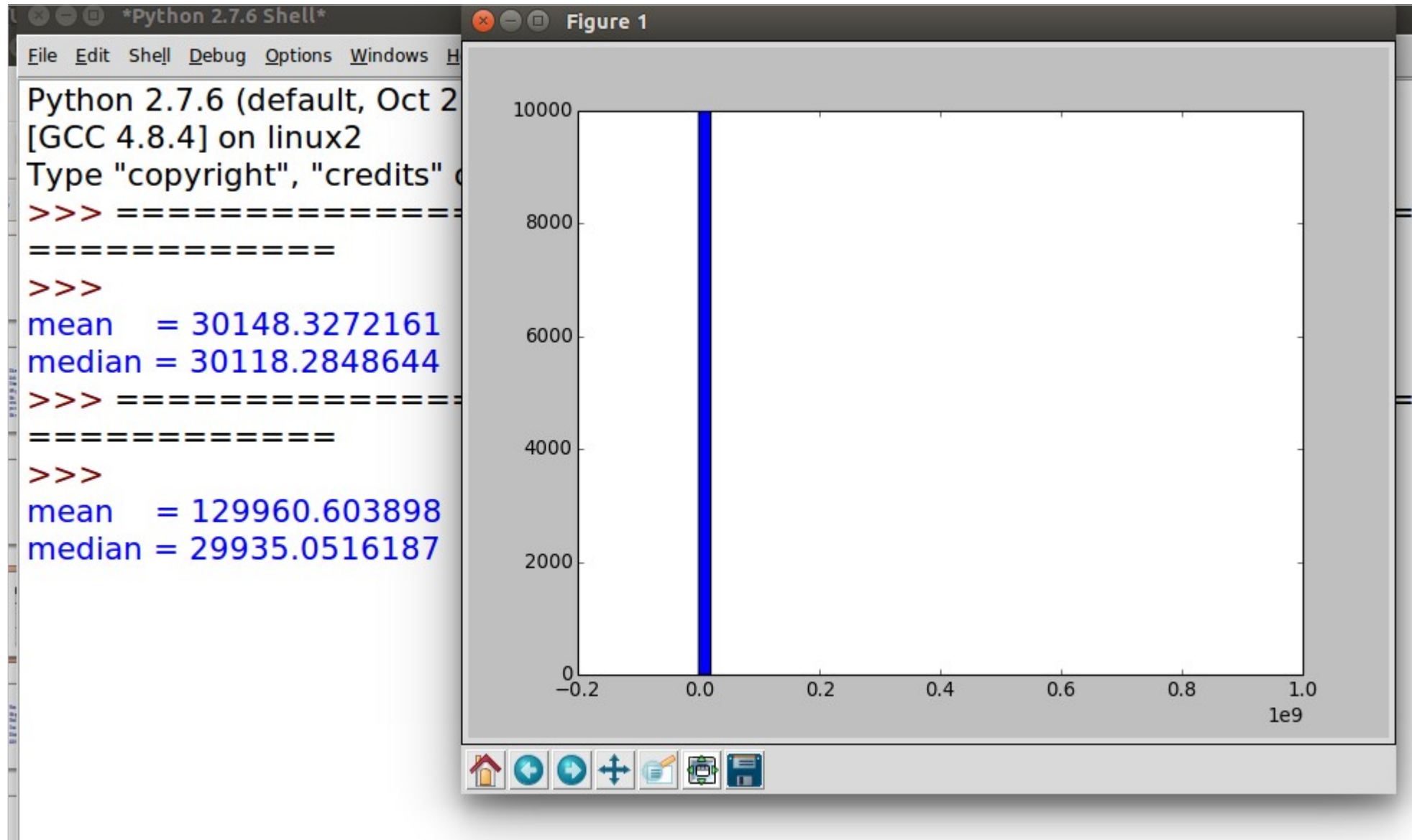
plt.show()
```



# Incomes w/o Billionaire's Salary



# Incomes w/ Billionaire's Salary



# Computing Mode

source in mean\_median\_mode.py

```
import numpy as np

import matplotlib.pyplot as plt

from scipy import stats

## create a random array of 500 ages from 10 up to 90.
ages = np.random.randint(10, high=90, size=500)

print(ages)

mo = stats.mode(ages)

## the result will print like (array([ 22.]), array([ 18.])),
## where the first number, 22, is the most frequent age
## and the second number, 18, is the number of times
##that age occurs in ages.

print(mo)
```



# Mode Test Run

```
[26 45 69 33 68 66 22 66 65 34 48 46 88 53 84 87 85 68 11 54 12 81 38 84 49 16 81 66 22 47 19 69 89 33 80 79 62 63 46 59 88 51 63 83 60 59 61 19 17 60
52 40 49 56 56 69 43 33 56 68 21 39 31 41 75 35 79 50 25 80 43 16 38 72 32 22 86 41 10 89 19 27 72 39 29 77 88 36 84 26 14 38 16 64 73 20 25 29 26 53
28 54 45 45 62 85 51 74 39 30 25 17 59 32 58 49 70 52 37 13 72 12 82 86 88 72 12 31 35 31 13 74 30 89 85 17 22 78 53 67 52 47 86 77 40 15 62 30 41 52
40 27 76 26 65 86 81 36 76 32 41 24 24 24 83 72 61 60 51 80 47 33 47 32 82 22 18 61 79 34 36 28 54 41 80 58 38 56 46 12 64 84 65 59 55 66 72 50 54 60
65 45 32 64 55 85 56 11 20 67 56 85 15 46 84 48 70 72 23 60 89 73 71 55 23 27 87 85 70 77 42 27 87 22 12 35 33 13 53 37 11 62 78 75 20 11 41 51 11 33
29 83 29 70 33 33 75 65 10 14 89 19 33 76 85 63 80 57 76 22 84 88 42 53 34 33 74 69 41 14 13 26 51 58 89 86 48 64 21 31 41 10 76 73 65 30 83 61 40 35
76 58 81 73 36 46 71 61 53 37 18 78 29 48 60 14 10 55 77 69 35 76 82 21 35 66 61 20 59 18 26 40 36 77 42 22 57 86 40 86 65 16 41 70 19 71 85 60 68 52
52 51 84 28 27 82 34 84 59 73 58 65 35 37 40 67 36 56 39 11 17 71 75 48 12 30 17 24 53 27 35 40 60 17 69 37 83 85 28 16 61 40 67 44 73 21 70 71 47 69
69 56 86 44 44 67 41 76 49 24 46 11 13 56 15 55 56 10 55 63 40 28 87 53 32 62 82 79 66 40 26 80 30 65 34 49 47 78 17 23 65 62 62 52 66 29 64 15 58 78
41 40 71 71 48 73 41 75 73 66 51 60 23 31 31 79 23 16 31 23 67 78 34 59 13 74 26 16 37 75 14 58 55 82 74 29 30 25 37 63 41 15 17 66 64 84 11 49 52 18]
(array([ 41.]), array([ 13.]])
```





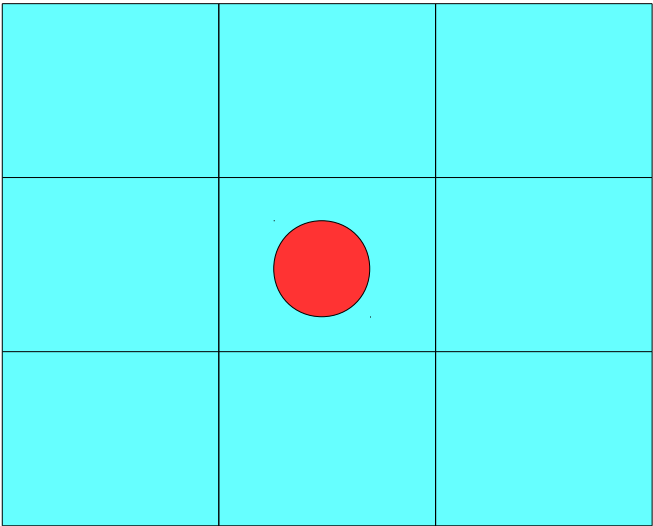
# Pixel Masks



# Pixel Masks

	0	1	2	...	N
0	128	255	10	201	203
1	120	35	50	25	137
2	34	89	190	197	108
...	180	178	215	37	45
M	24	25	91	225	225


3 x 3 Pixel Mask



Pixel masks are used to compute various properties of the center pixel




# Pixel Masks

	0	1	2	...	N
0	128	255	10	201	203
1	120	35	50	25	137
2	34	89		197	108
...	180	178	215	37	45
M	24	25	91	225	225

Pixel masks are superimposed on the image to compute various properties of the center pixel.



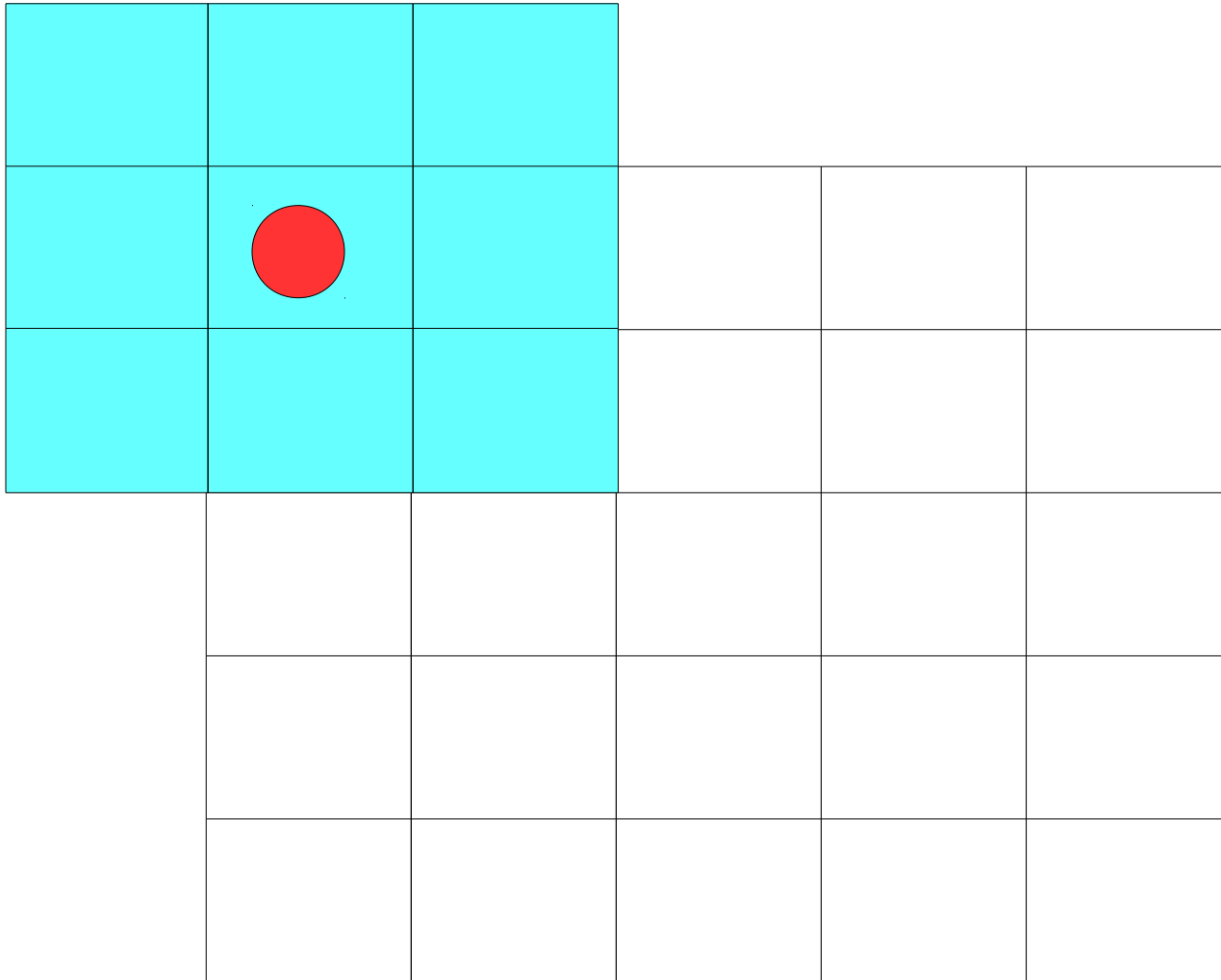
# Computing Cell Properties

	0	1	2	3	4
0	128	255	10	201	203
1	120	35	50	25	137
2	34	89		197	108
3	180	178	215	37	45
4	24	25	91	225	225

Properties of cell  $I[2, 2]$  can be computed in terms of cells  $I[1,1]$ ,  $I[2, 1]$ ,  $I[3, 1]$ ,  $I[1, 2]$ ,  $I[3,2]$ ,  $I[1, 3]$ ,  $I[2, 3]$ , and  $I[3,3]$ .



# Border Pixel Problem



What happens when the mask is centered at a border pixel? Some pixels covered by the mask do not exist.

Two possible approaches:

- 1) Pad the image; this can be done virtually.
- 2) Do not center the mask at the border pixels.



# Image Convolution: Applying Masks to Images

- Given an image  $I$  and a mask  $M$ ,  $M$  is centered at each possible pixel and a value  $v$  is computed
- This value  $v$  is saved in a new image or the value of the current pixel on which  $M$  is centered is destructively modified with  $v$
- This process is sometimes called *image convolution*



**Blur**



# Blurring

- Blurring is another type of filtering operation
- A sharp image is an image where one can clearly see all objects
- Sharpness is a consequence of clear edges
- Why do we need to blur?
- We may want to blur to make the image smoother (remove some small edges here and there) in the image to make subsequent processing more effective
- We may want to blur to create an artistic effect (e.g., motion blur)





# Types of Blurring

- Mean filter
- Weighted average filter
- Median filter
- Gaussian filter
- All these filters (and many more) are available in OpenCV



# Gaussian Blurring

Pixel's x, y coordinates

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Standard deviation either in  
a kernel or entire image



# Problem

Write a program that takes a command line arguments that specify a path to an image, applies various blurring filters to the image and displays the results.

Sample run:

```
$ python blurring.py road01.png
```



# Solution

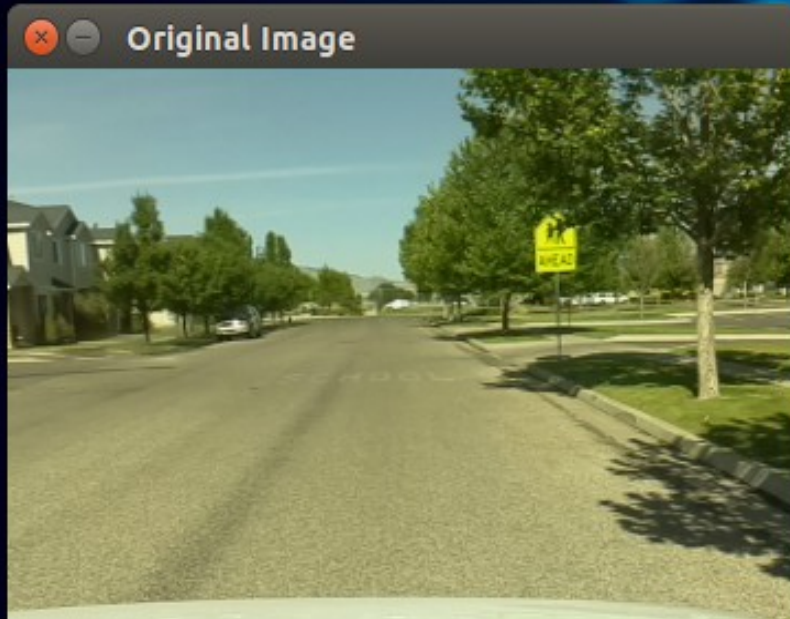
```
image = cv2.imread(sys.argv[1])  
cv2.imshow('Original Image', image)  
  
kernel_3x3 = np.ones((3, 3), np.float32) / 9  
  
blurred = cv2.filter2D(image, -1, kernel_3x3)  
cv2.imshow('3x3 Kernel Blurring', blurred)  
  
kernel_7x7 = np.ones((7, 7), np.float32) / 49  
  
blurred2 = cv2.filter2D(image, -1, kernel_7x7)  
cv2.imshow('7x7 Kernel Blurring', blurred2))
```

What is -1? This means the depth (number of bits for each color in a single pixel) of the blurred image (blurred) will be the same as the depth of the original image (image)

source in blurring.py



# Sample Run



# Solution

blurring2.py

```
image = cv2.imread(sys.argv[1])
cv2.imshow('Original Image', image)

blur = cv2.blur(image, (3, 3))
cv2.imshow('Mean (3x3)', blur)

gauss = cv2.GaussianBlur(image, (7, 7), 0)
cv2.imshow('Gaussian (7x7)', gauss)

median = cv2.medianBlur(image, 5)
cv2.imshow('Median (5x5)', median)

## bilateral is great for keeping edges sharp.
bilateral = cv2.bilateralFilter(image, 9, 75, 75)
cv2.imshow('Bilateral 9', bilateral)
```

Replaces the pixel in the center of a 5x5 kernel with the median value of the kernel's pixels

Similar to Gaussian but keeps edges sharper

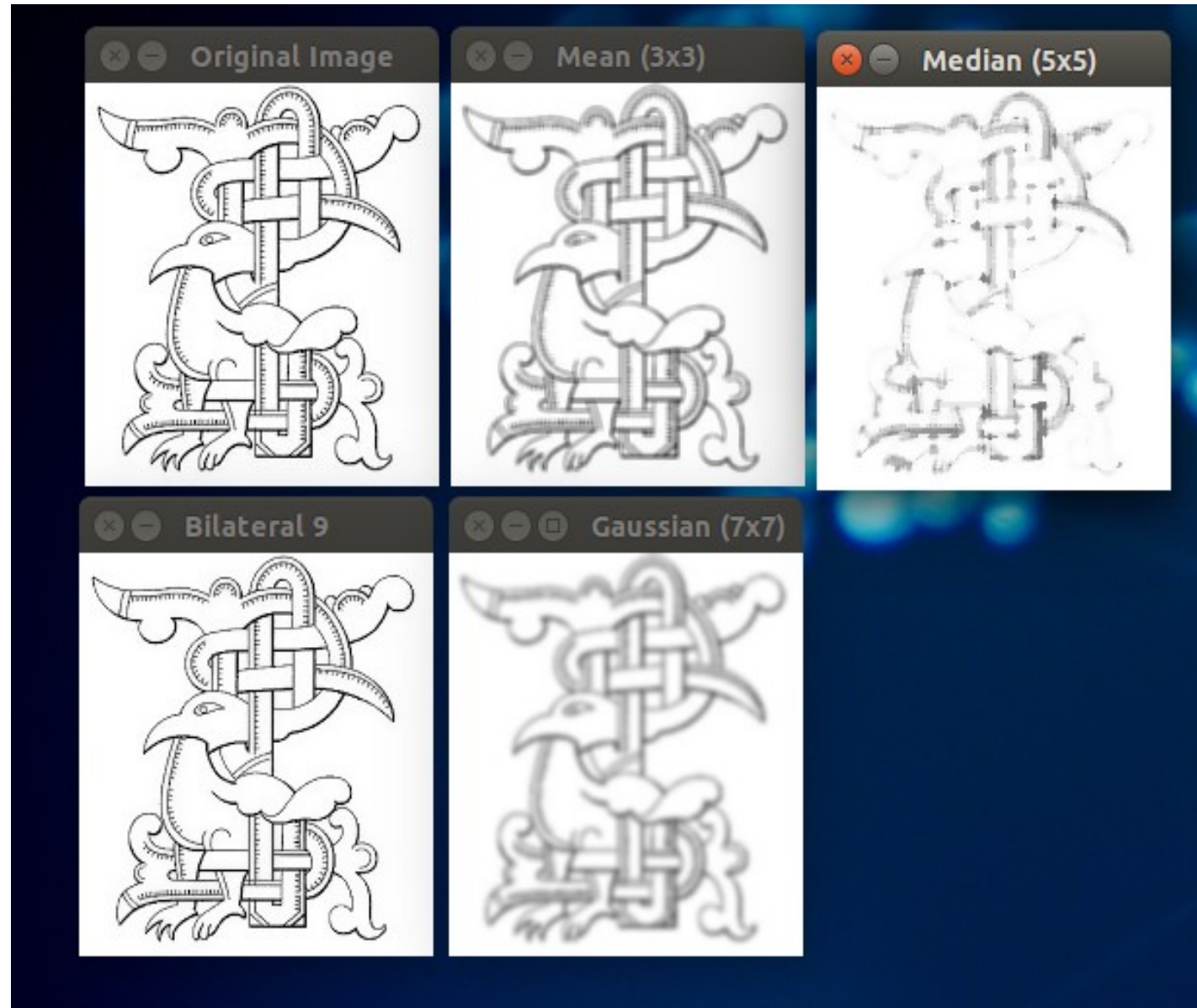




# Test Run



# Test Run





# Erosion & Dilation



# Erosion & Dilation

- Two most common morphological filters are **erosion** and **dilation**
- Erosion replaces the current pixel with the minimum pixel found in the kernel
- Dilation replaces the current pixel with the maximum pixel found in the kernel



# Erosion & Dilation

- Let us suppose that we apply erosion and dilation to a binary image (**0 – black, 255 – white**)
- We expect erosion to increase the amount of blackness in the image (since the minimum pixel value is chosen in each shape element)
- We expect dilation to increase the amount of whiteness in the image (since the maximum pixel value is chosen in each shape element)



# Erosion & Dilation



Image Source: R. Laganriere. “**OpenCV 2 Cookbook**”, Ch. 05



# Erosion & Dilation

```
import cv2
import sys

img = cv2.imread(sys.argv[1])
cv2.imshow('Original', img)

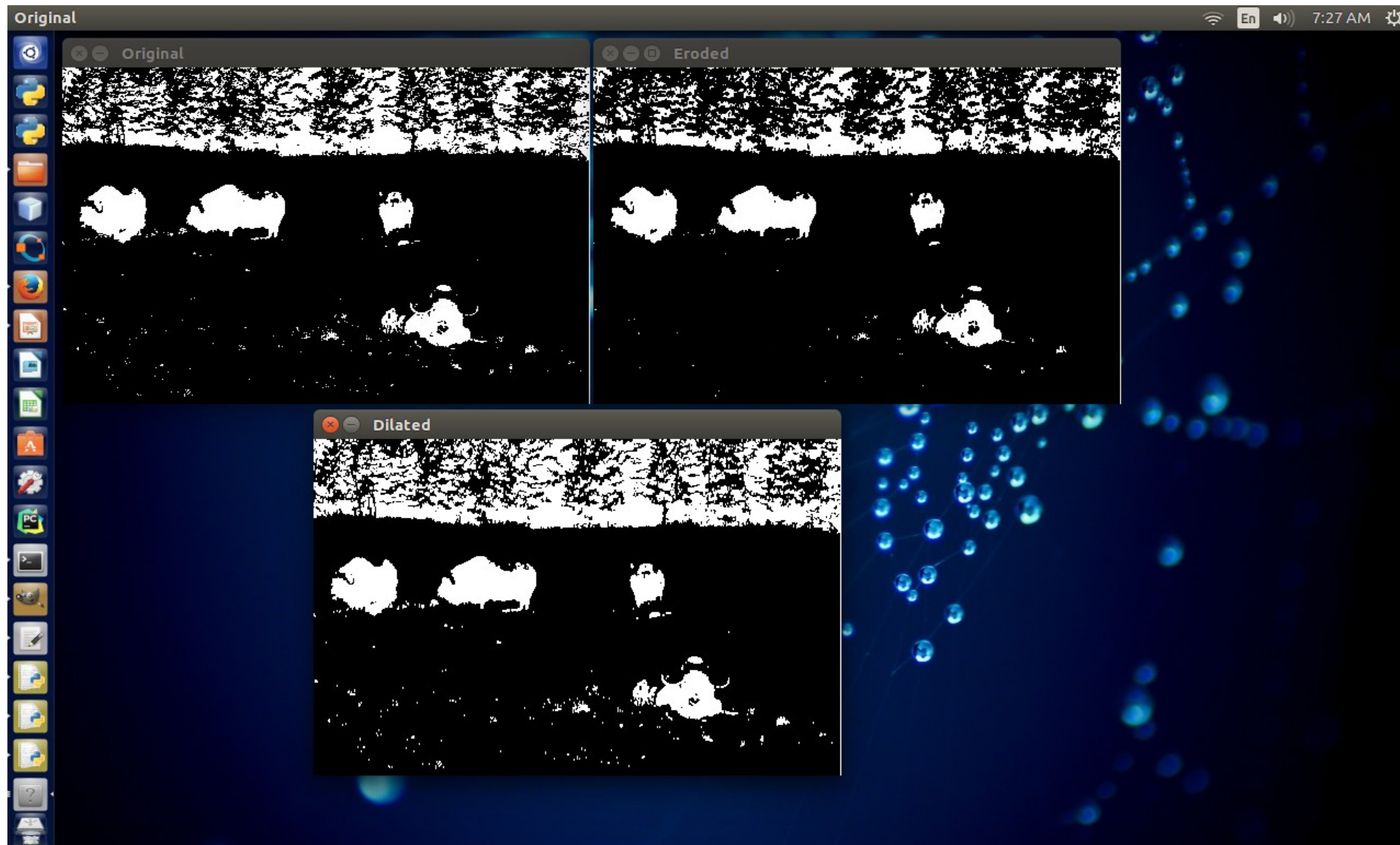
er_img = cv2.erode(img, (5, 5))
cv2.imshow('Eroded', er_img)

dl_img = cv2.dilate(img, (5, 5))
cv2.imshow('Dilated', dl_img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```



# Test Run



# References

- [https://en.wikipedia.org/wiki/Gaussian\\_blur](https://en.wikipedia.org/wiki/Gaussian_blur)
- [https://en.wikipedia.org/wiki/Erosion\\_\(morphology\)](https://en.wikipedia.org/wiki/Erosion_(morphology))
- [https://en.wikipedia.org/wiki/Dilation\\_\(morphology\)](https://en.wikipedia.org/wiki/Dilation_(morphology))
- [www.opencv.org](http://www.opencv.org)

