

Objectives

- To understand pixel based image processing
- To use nested iteration
- To use and understand tuples
- To implement a number of image processing algorithms
- To understand passing functions as parameters

Digital Image Processing

- Editing and manipulating digital images
- A digital image is a collection of pixels
- A pixel is the smallest amount of information available in a digital picture
- Each pixel represents a single color
- Pixels are organized in a grid

RGB Color Model

- RGB means RED, GREEN, BLUE
- Each color is made up from amounts of red, green and blue
- Color intensities range from a minimum of 0 to a maximum of 255

clmage module

- Pixel
- ImageWin
- EmptyImage
- FileImage

Pixel objects

- getRed
- getGreen
- getBlue
- setRed
- setGreen
- setBlue

FileImage and EmptyImage

- getWidth
- getHeight
- getPixel
- setPixel
- draw

Negative Image

- Reverse the colors in each pixel

Listing 6.1

```
def negativePixel(oldPixel):  
    newred = 255 - oldPixel.getRed()  
    newgreen = 255 - oldPixel.getGreen()  
    newblue = 255 - oldPixel.getBlue()  
    newPixel = Pixel(newred, newgreen, newblue)  
    return newPixel
```


Figure 6.1

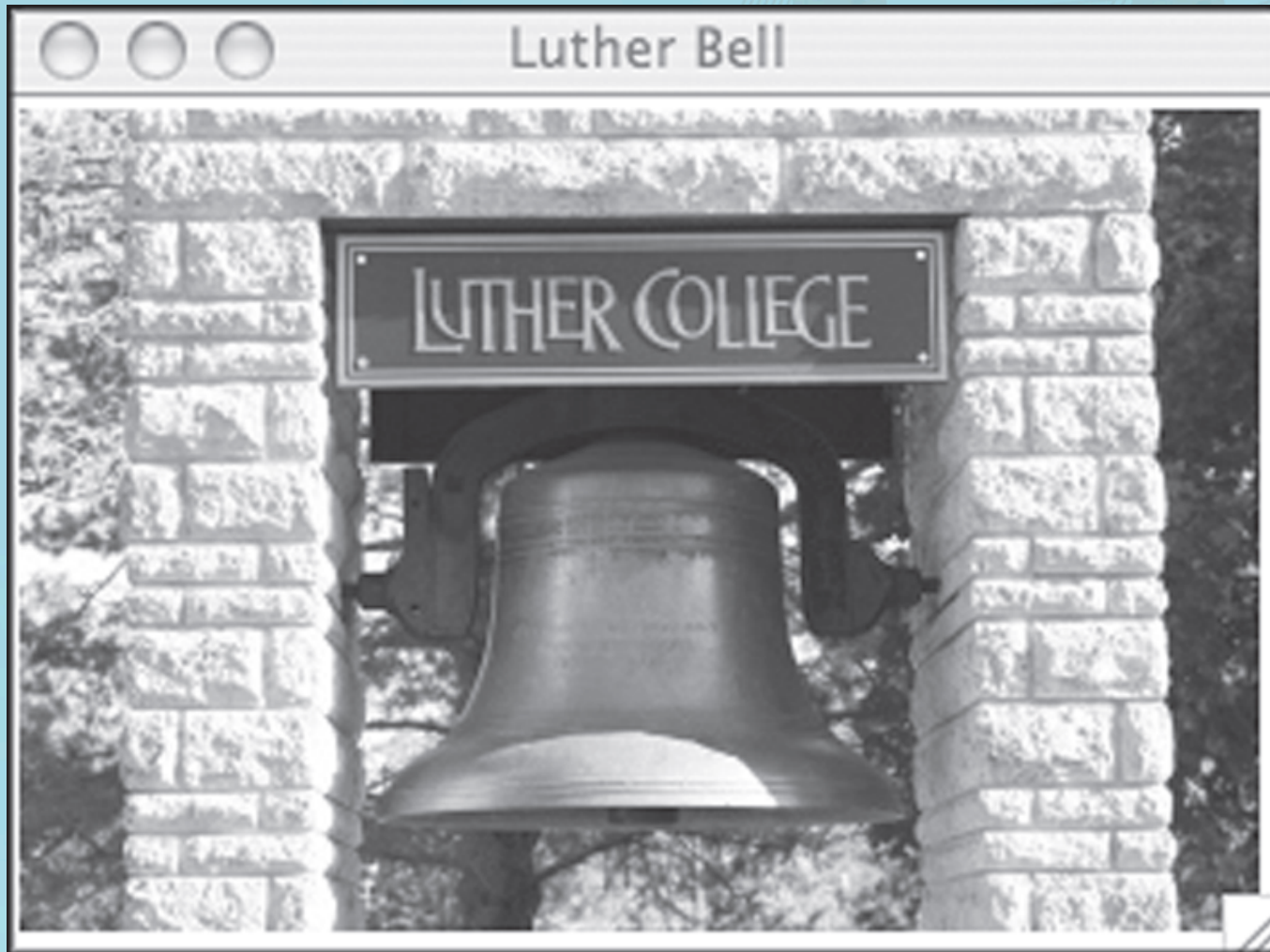
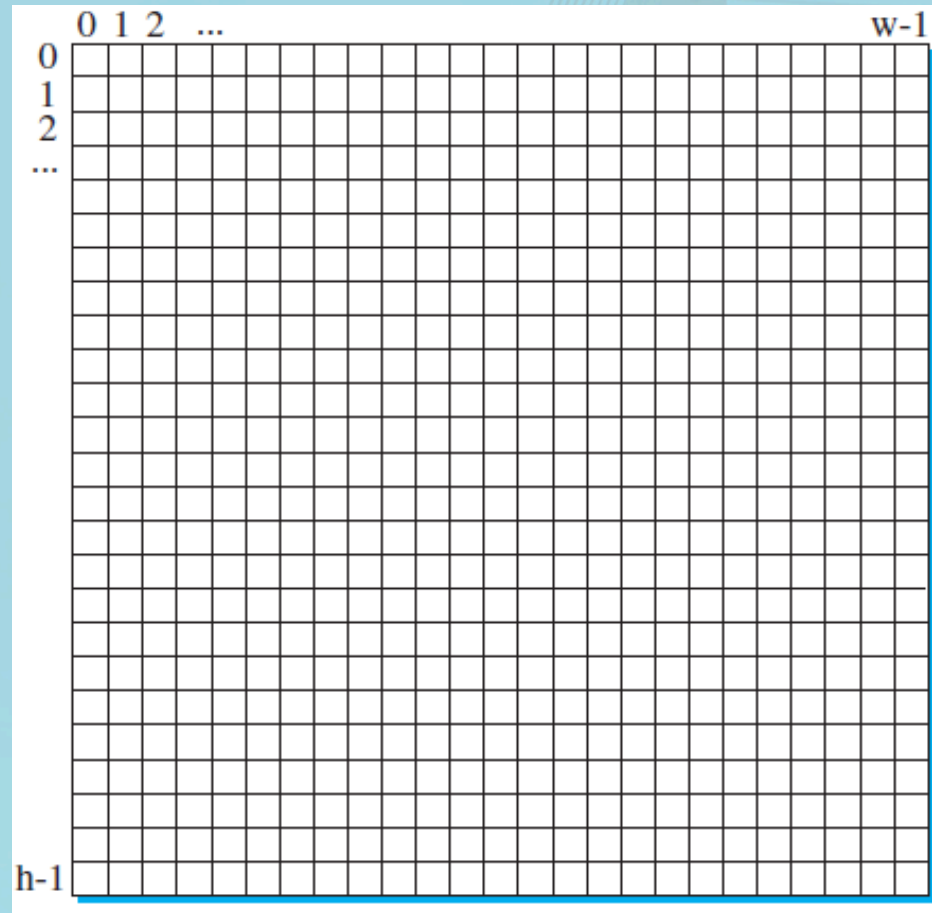


Figure 6.2



Listing 6.2

```
def makeNegative(imageFile):  
    myimagewindow = ImageWin("Image Processing",600,200)  
    oldimage = FileImage(imageFile)  
    oldimage.draw(myimagewindow)  
  
    width = oldimage.getWidth()  
    height = oldimage.getHeight()  
    newim = EmptyImage(width,height)  
  
    for row in range(height):  
        for col in range(width):  
            originalPixel = oldimage.getPixel(col,row)  
            newPixel = negativePixel(originalPixel)  
            newim.setPixel(col,row,newPixel)  
  
    newim.setPosition(width+1,0)  
    newim.draw(myimagewindow)  
    myimagewindow.exitOnClick()
```

Figure 6.3

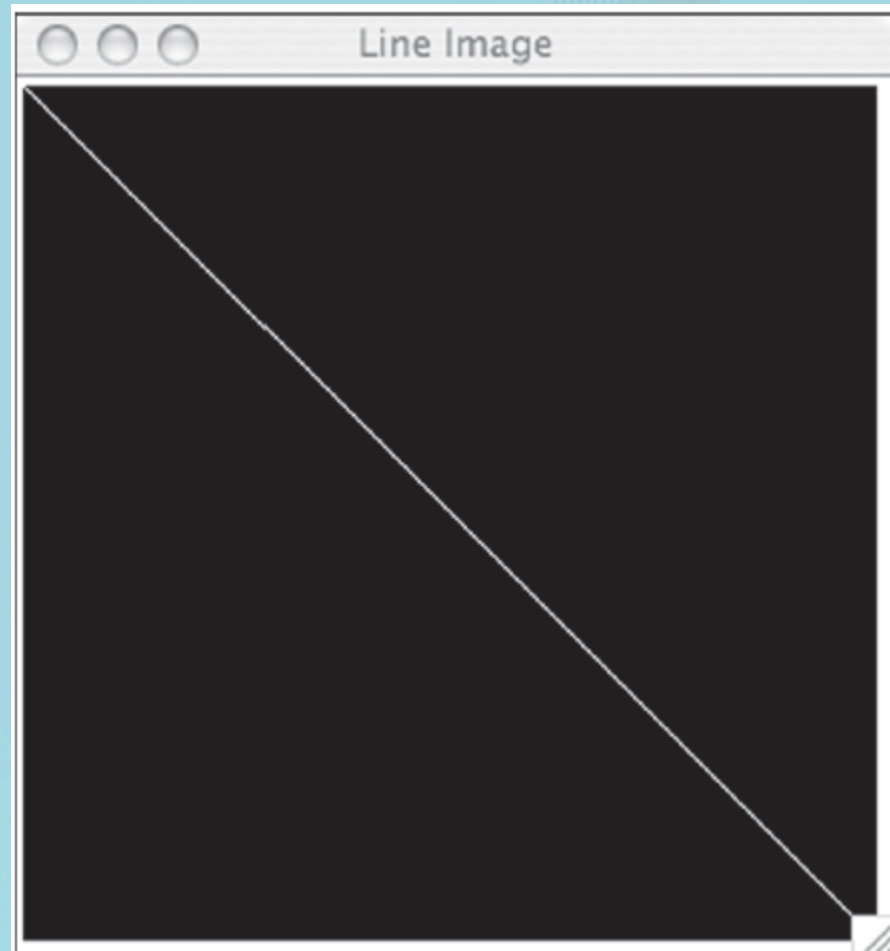
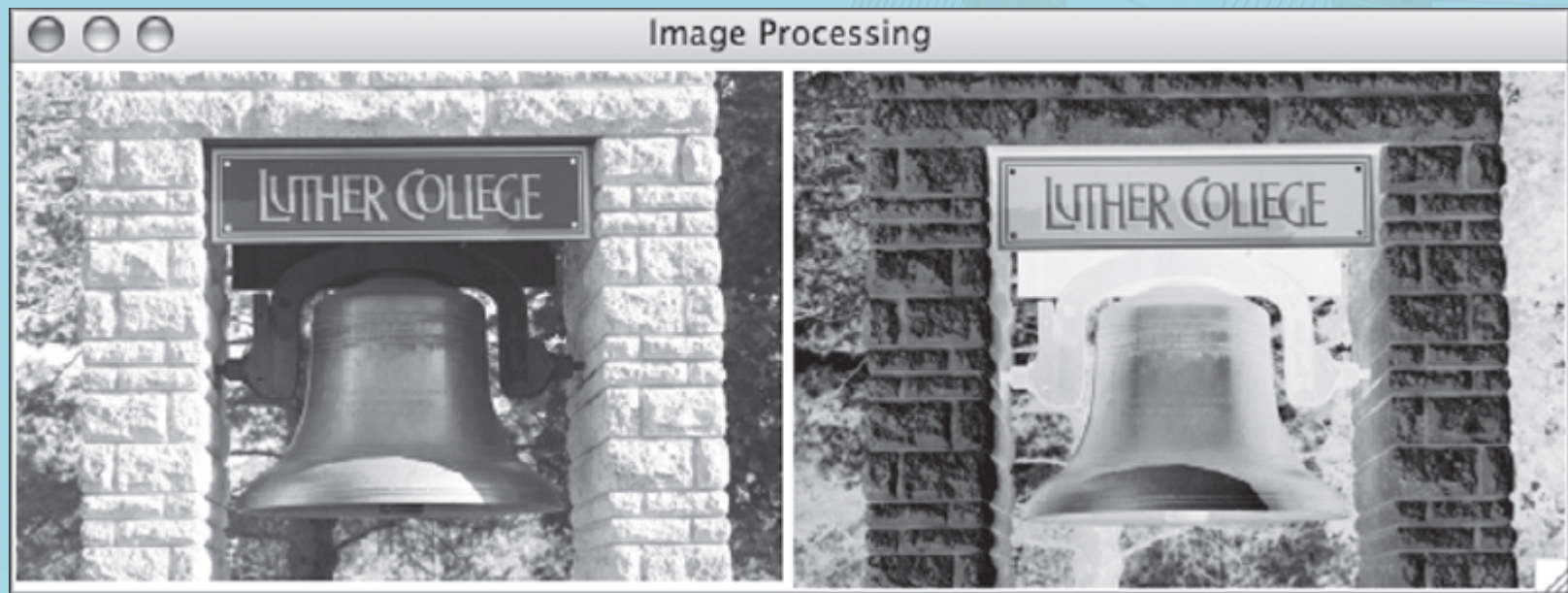


Figure 6.4



Grayscale Image

- Convert each pixel to a shade of gray
- Use average of RGB values

Listing 6.3

```
def grayPixel(oldpixel):  
    intensitySum = oldpixel.getRed() + oldpixel.getGreen() + oldpixel.getBlue()  
    aveRGB = intensitySum // 3  
  
    newPixel = Pixel(aveRGB,aveRGB,aveRGB)  
    return newPixel
```


Listing 6.4

```
def makeGrayScale(imageFile):  
    myimagewindow = ImageWin("Image Processing",600,200)  
    oldimage = FileImage(imageFile)  
    oldimage.draw(myimagewindow)  
  
    width = oldimage.getWidth()  
    height = oldimage.getHeight()  
    newim = EmptyImage(width,height)  
  
    for row in range(height):  
        for col in range(width):  
            originalPixel = oldimage.getPixel(col,row)  
            newPixel = grayPixel(originalPixel)  
            newim.setPixel(col,row,newPixel)  
  
    newim.setPosition(width+1,0)  
    newim.draw(myimagewindow)  
    myimagewindow.exitOnClick()
```


Figure 6.5



Generalize Pixel Mapping

- Create a function that takes a pixel manipulation function as a parameter

Figure 6.6

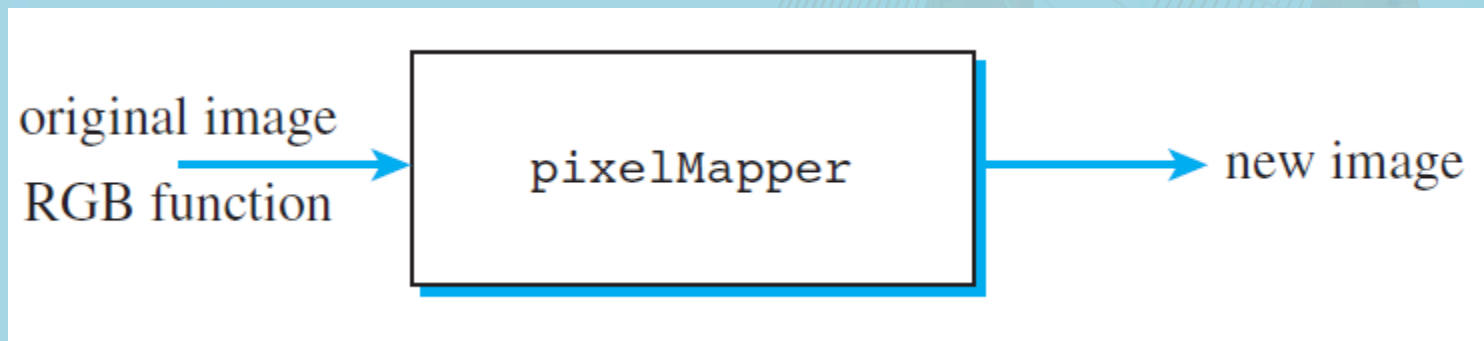


Figure 6.7

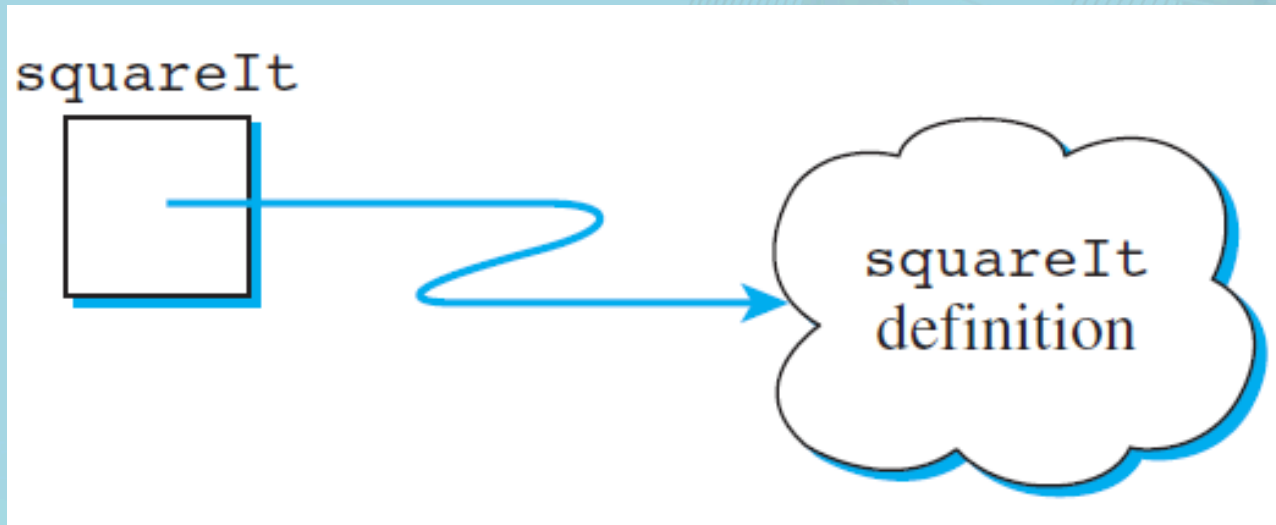
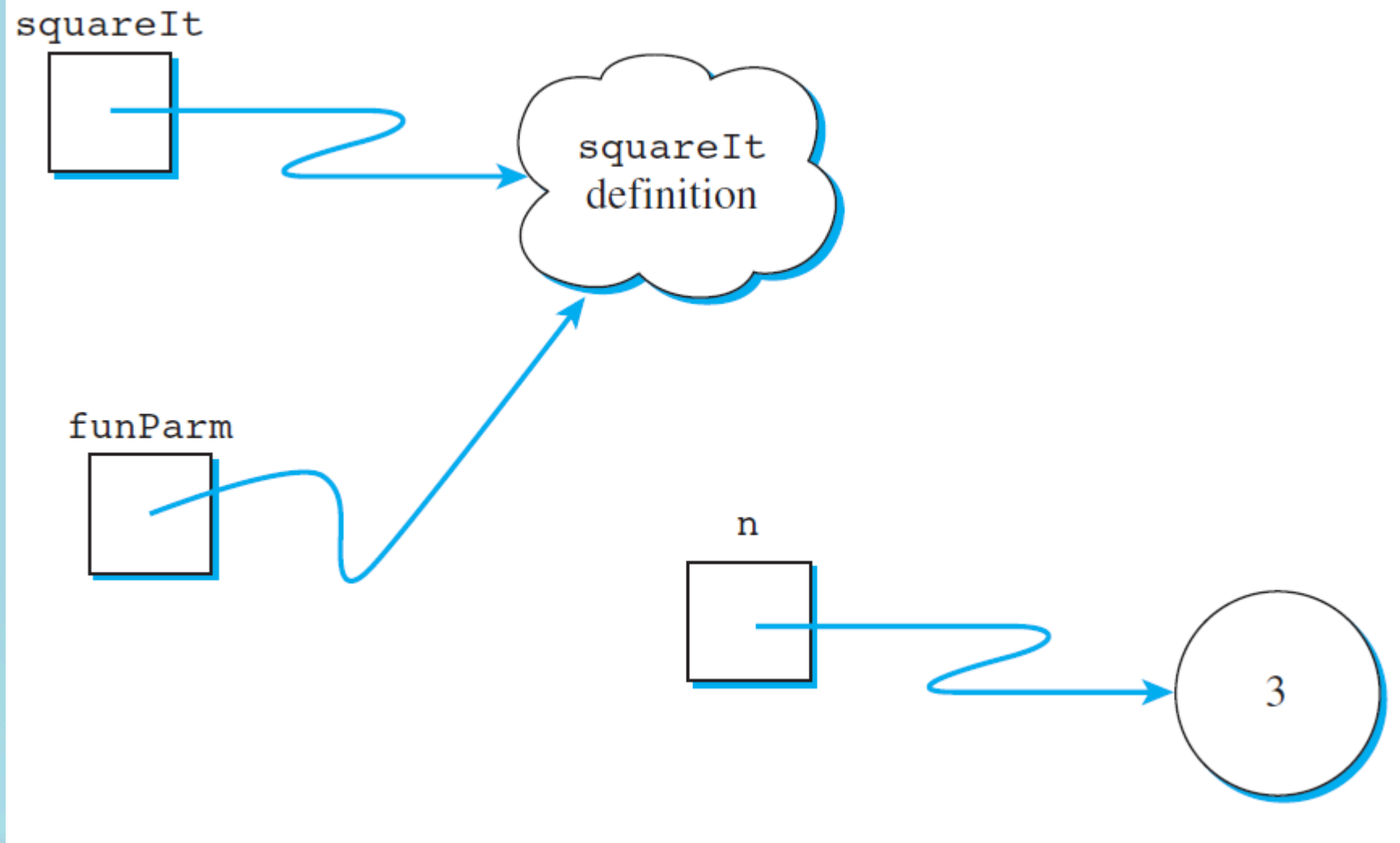


Figure 6.8



Listing 6.5

```
def pixelMapper(oldimage,rgbFunction):  
  
    width = oldimage.getWidth()  
    height = oldimage.getHeight()  
    newim = EmptyImage(width,height)  
  
    for row in range(height):  
        for col in range(width):  
            originalPixel = oldimage.getPixel(col,row)  
            newPixel = rgbFunction(originalPixel)  
            newim.setPixel(col,row,newPixel)  
  
    return newim
```


Listing 6.6

```
def generalTransform(imageFile):  
    myimagewindow = ImageWin("Image Processing",600,200)  
    oldimage = FileImage(imageFile)  
    oldimage.draw(myimagewindow)  
  
    newimage = pixelMapper(oldimage,grayPixel)  
    newimage.setPosition(oldimage.getWidth()+1,0)  
    newimage.draw(myimagewindow)  
    myimagewindow.exitOnClick()
```

Namespaces

- Parameters, Parameter Passing, and Scope
- How are the names and object organized in Python
- Builtin
- Main
- Local

Listing 6.7

```
import math
```

```
def hypotenuse(a,b):
```

```
    c = math.sqrt(a**2 + b**2)
```

```
    return c
```

Figure 6.9

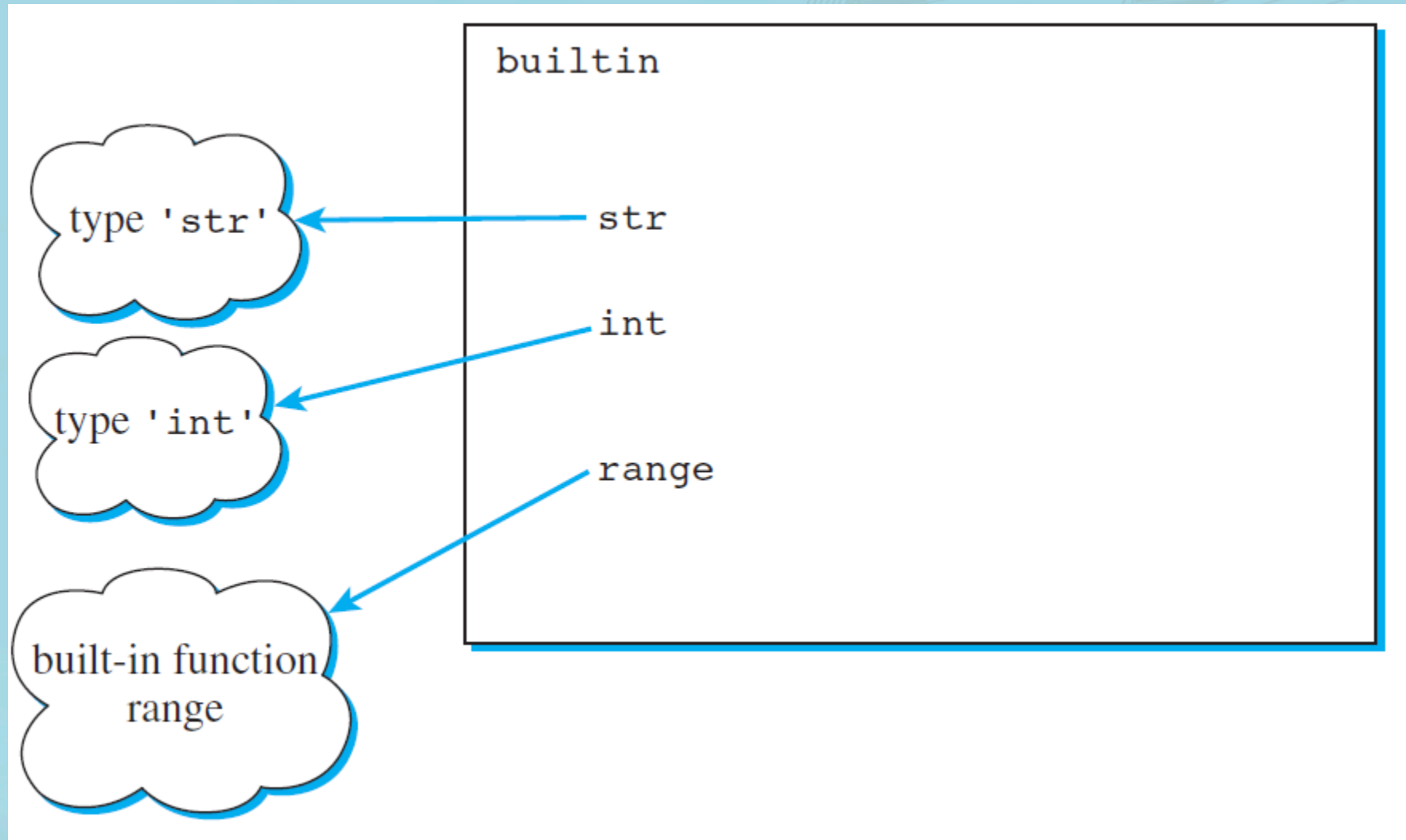


Figure 6.10

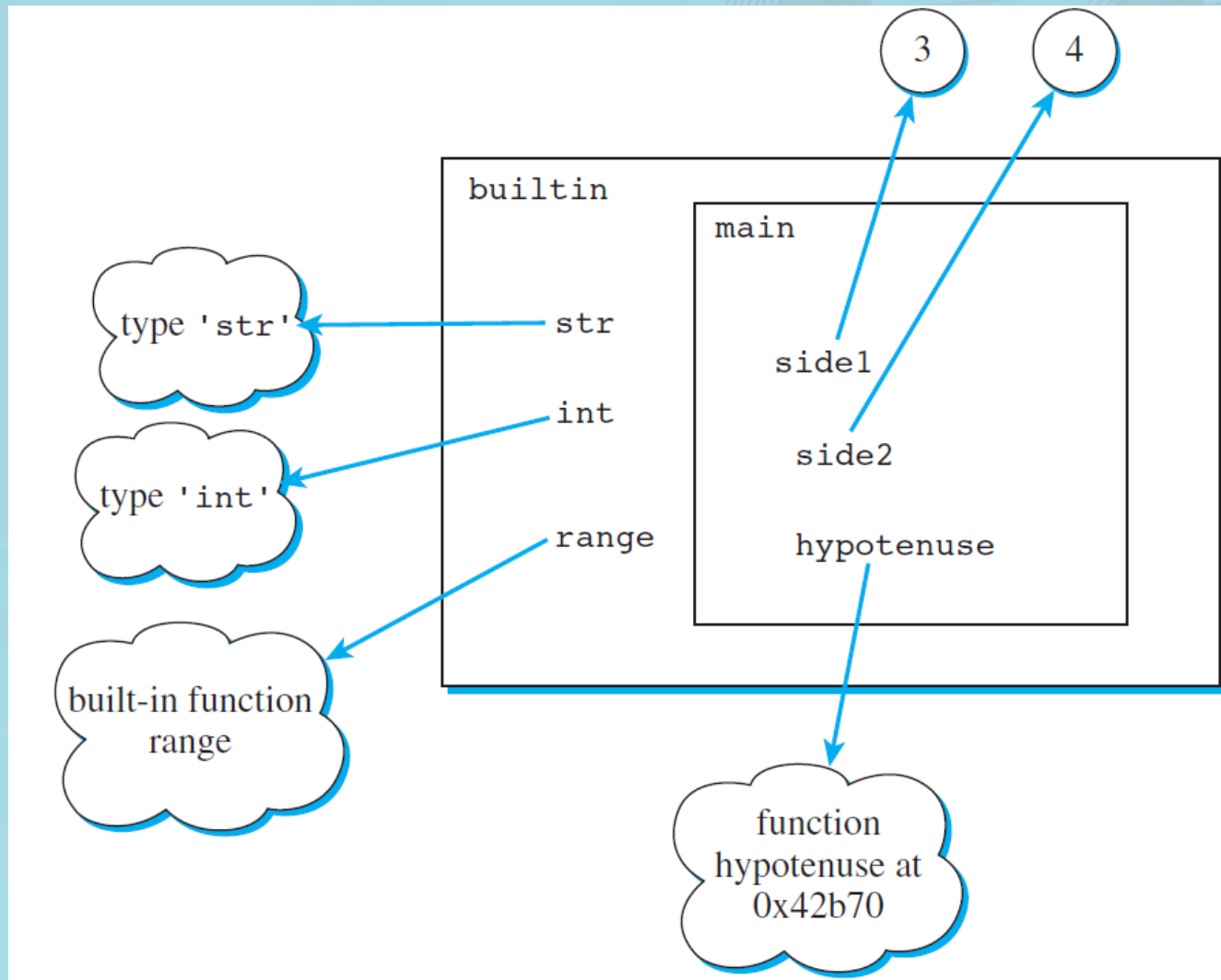


Figure 6.11

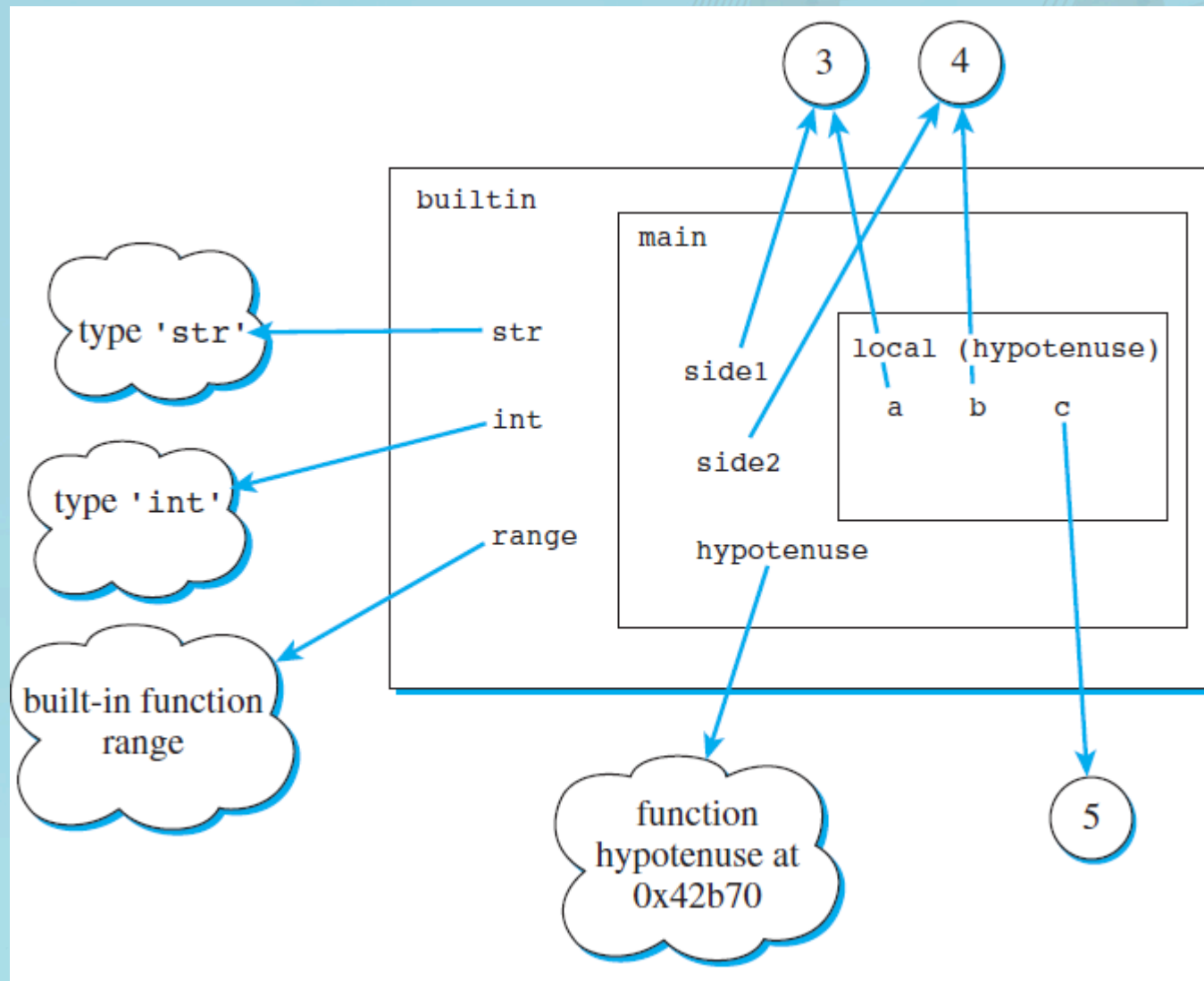


Figure 6.12

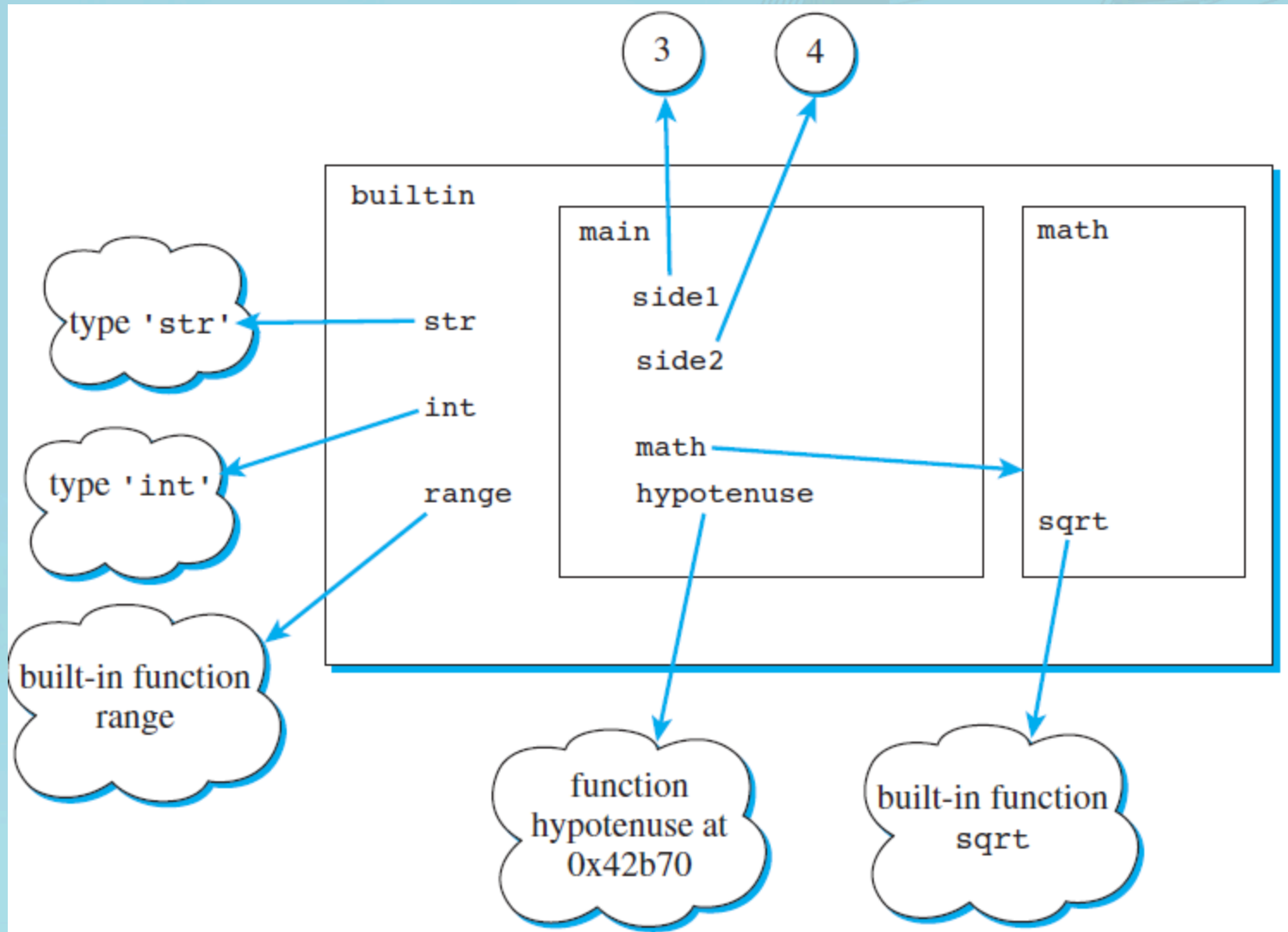
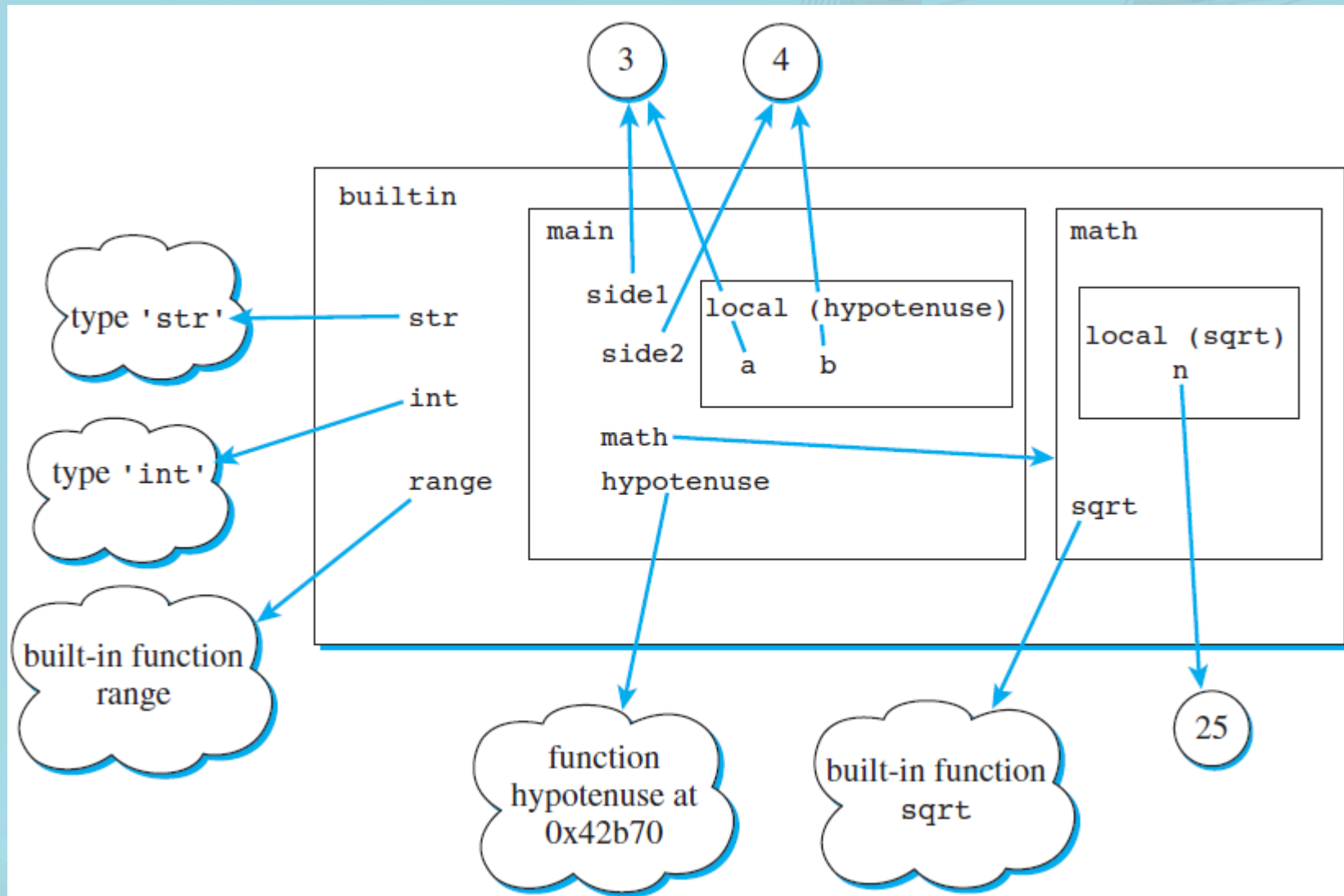


Figure 6.13



Enlarging an Image

- Stretching an image
- Mapping pixels from original to enlargement

Figure 6.14

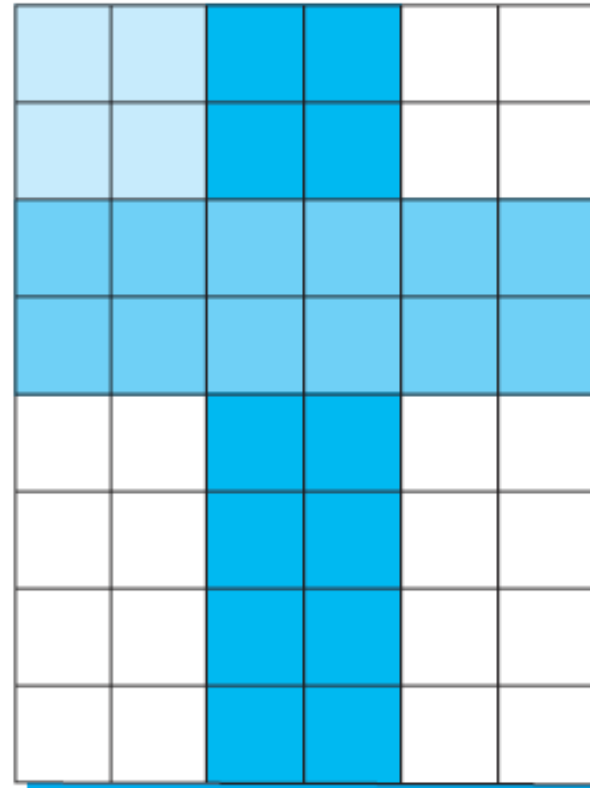
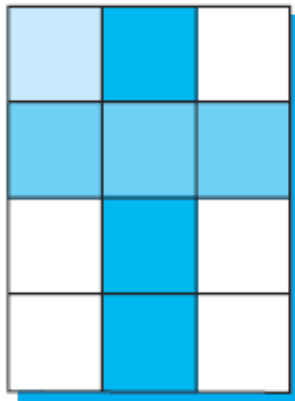


Figure 6.15

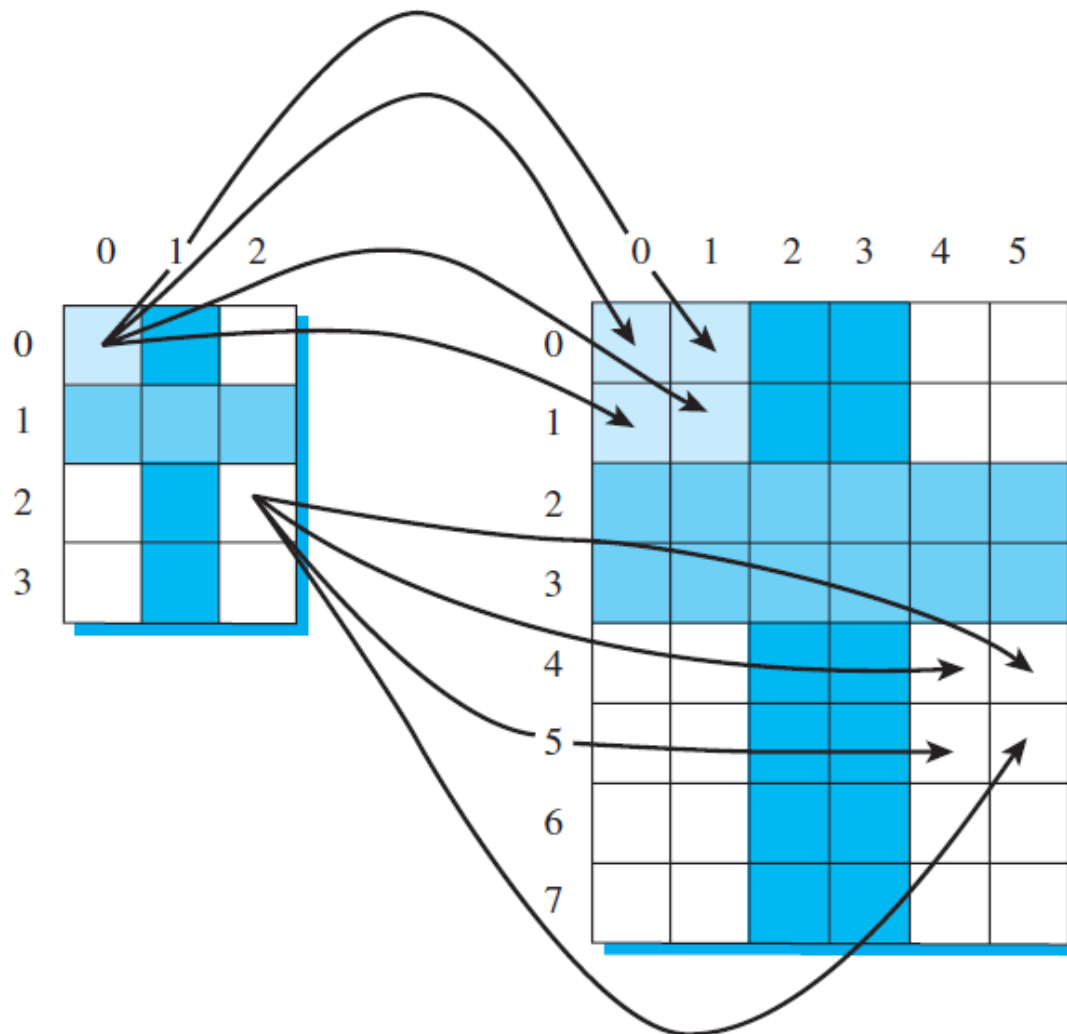
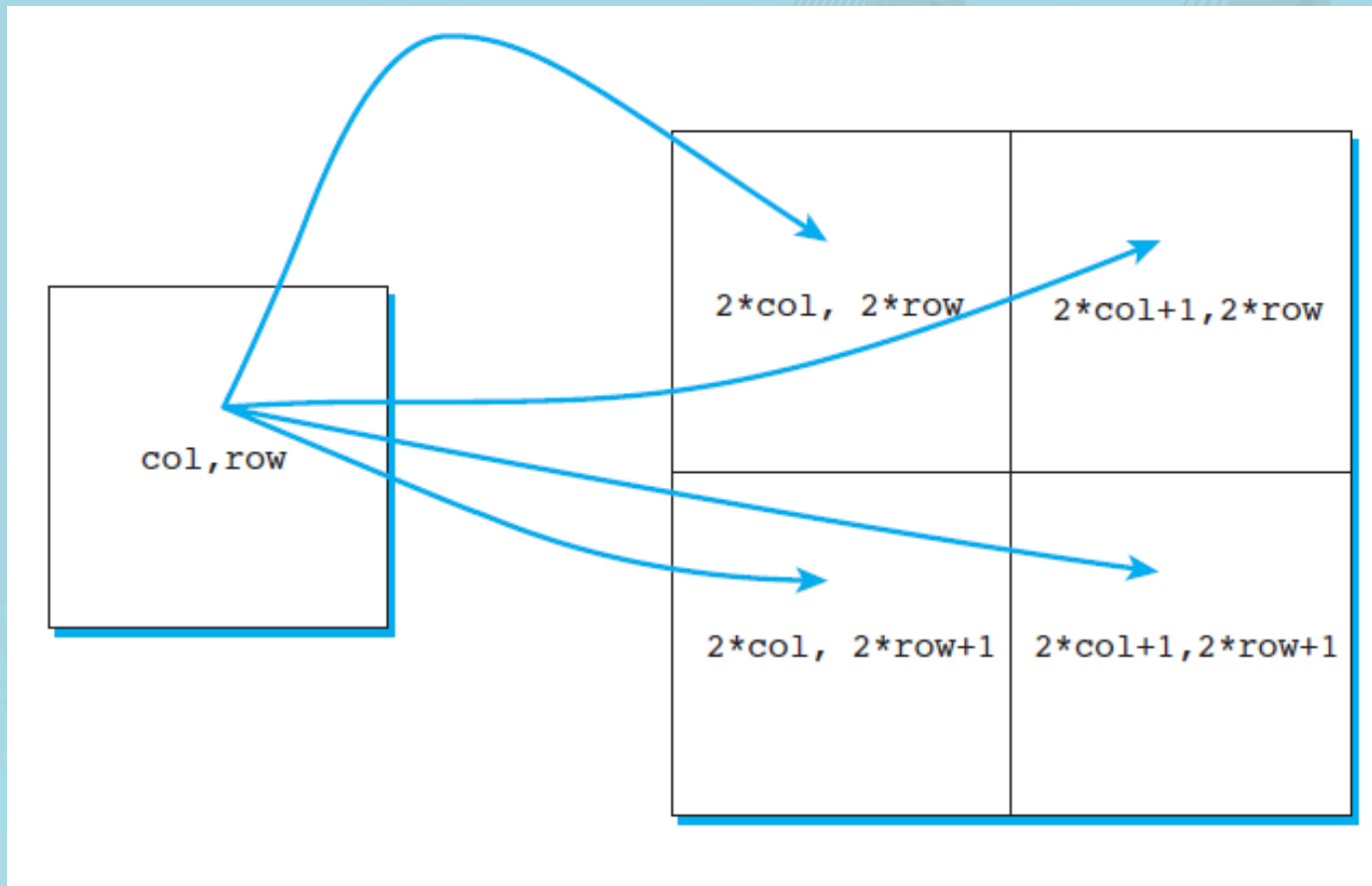


Figure 6.16



Listing 6.8

```
def double(oldimage):  
    oldw = oldimage.getWidth()  
    oldh = oldimage.getHeight()  
  
    newim = EmptyImage(oldw*2,oldh*2)  
  
    for row in range(oldh):  
        for col in range(oldw):  
            oldpixel = oldimage.getPixel(col,row)  
  
            newim.setPixel(2*col,2*row,oldpixel)  
            newim.setPixel(2*col+1,2*row,oldpixel)  
            newim.setPixel(2*col,2*row+1,oldpixel)  
            newim.setPixel(2*col+1,2*row+1,oldpixel)  
  
    return newim
```

Figure 6.17

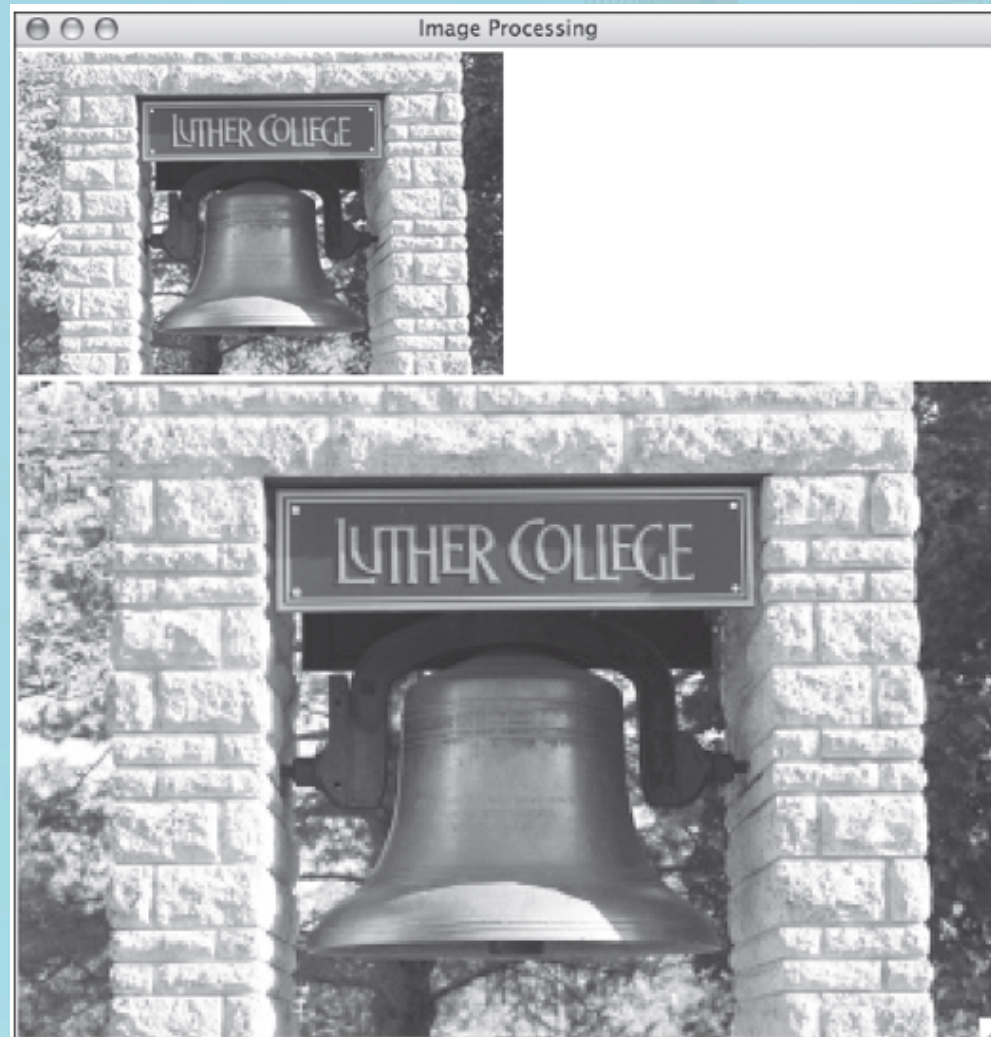
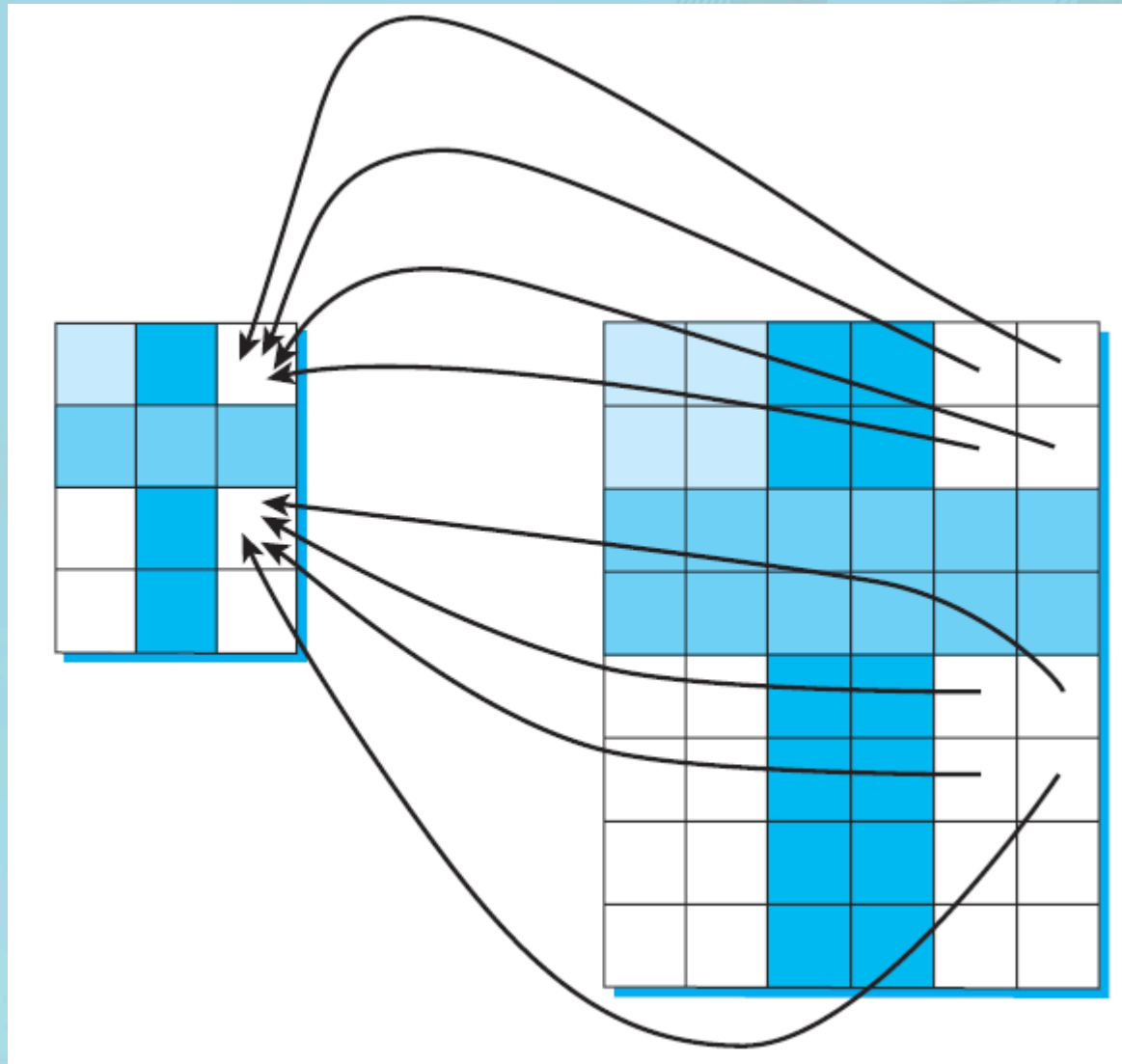


Image © 1shostak/Shutterstock, Inc.
Copyright © 2014 by Jones & Bartlett Learning, LLC an Ascend Learning Company
www.jblearning.com

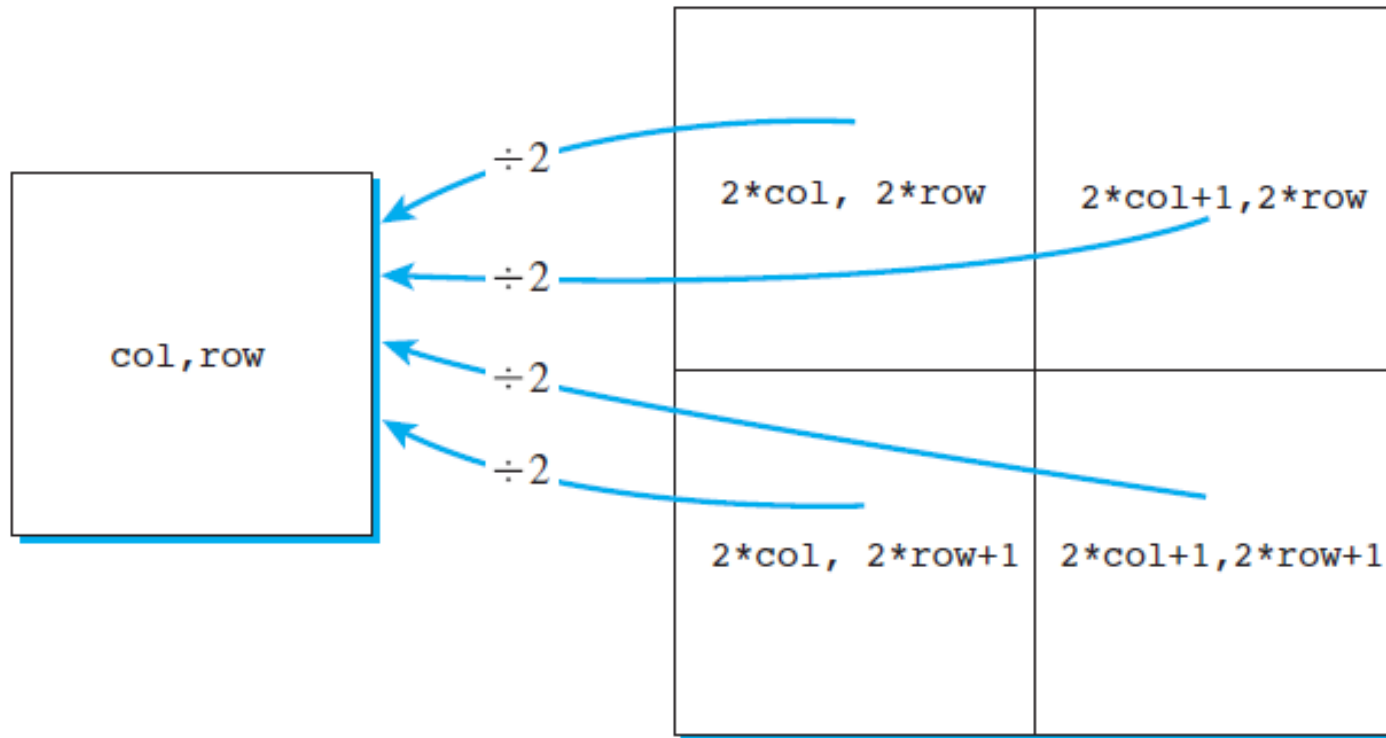
Figure 6.18



Listing 6.9

```
def double(oldimage):  
    oldw = oldimage.getWidth()  
    oldh = oldimage.getHeight()  
  
    newim = EmptyImage(oldw*2,oldh*2)  
  
    for row in range(newim.getHeight()):  
        for col in range(newim.getWidth()):  
  
            originalCol = col//2  
            originalRow = row//2  
            oldpixel = oldimage.getPixel(originalCol,originalRow)  
  
            newim.setPixel(col,row,oldpixel)  
  
    return newim
```

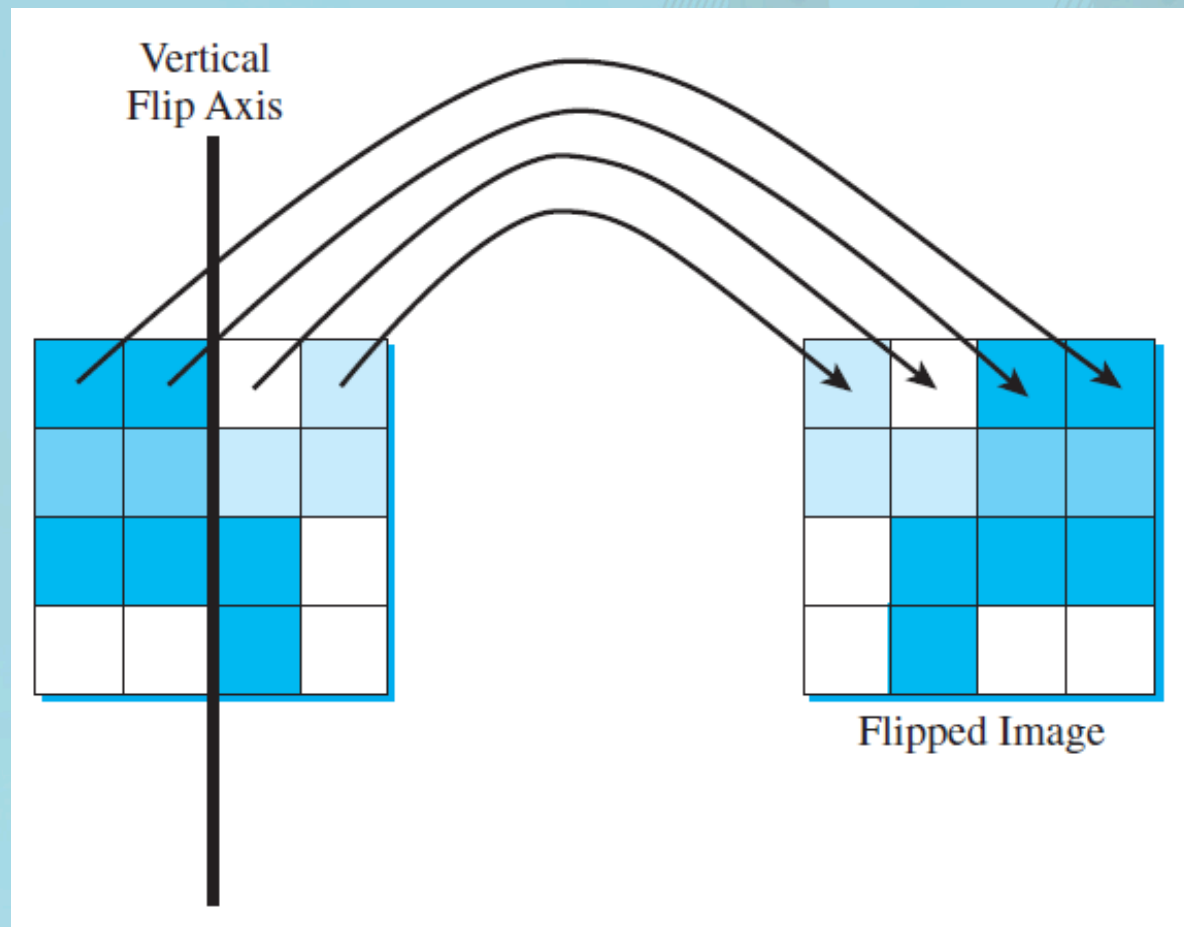

Figure 6.19



Flipping an Image

- Moving pixels to a new location
- Flip axis

Figure 6.20



Listing 6.10

```
def verticalFlip(oldimage):  
    oldw = oldimage.getWidth()  
    oldh = oldimage.getHeight()  
  
    newim = EmptyImage(oldw,oldh)  
  
    maxp = oldw -1  
    for row in range(oldh):  
        for col in range(oldw):  
  
            oldpixel = oldimage.getPixel(maxp-col,row)  
  
            newim.setPixel(col,row,oldpixel)  
  
    return newim
```


Figure 6.21



Edge Detection

- Convert to grayscale
- Look for changes from light to dark or dark to light

Figure 6.22



Original Image



Edges

Figure 6.23

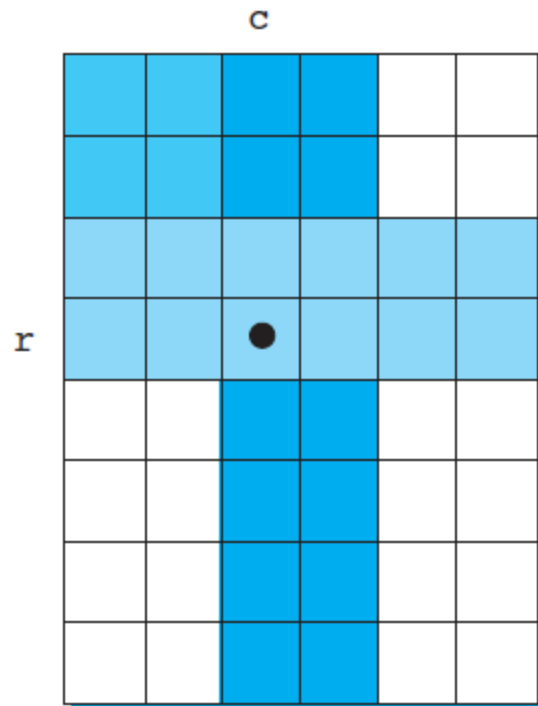
-1	0	1
-2	0	2
-1	0	1

XMask

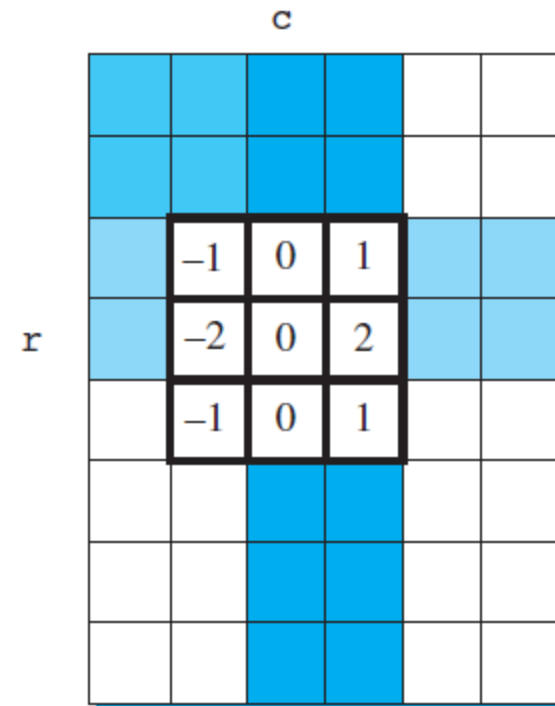
1	2	1
0	0	0
-1	-2	-1

YMask

Figure 6.24



Original Grayscale



Placing the xMask on Pixel c, r

Listing 6.11

```
def convolve(anImage,pixelRow,pixelCol,kernel):  
    kernelColumnBase = pixelCol - 1  
    kernelRowBase = pixelRow - 1  
  
    sum = 0  
    for row in range(kernelRowBase,kernelRowBase+3):  
        for col in range(kernelColumnBase,kernelColumnBase+3):  
            kColIndex = col-kernelColumnBase  
            kRowIndex = row-kernelRowBase  
  
            apixel = anImage.getPixel(col,row)  
            intensity = apixel.getRed()  
  
            sum = sum + intensity * kernel[kRowIndex][kColIndex]  
  
    return sum
```

Listing 6.12

```
import math

def edgeDetect(theImage):
    grayImage = pixelMapper(theImage,grayPixel)
    newim = EmptyImage(grayImage.getWidth(), grayImage.getHeight())
    black = Pixel(0,0,0)
    white = Pixel(255,255,255)
    xMask = [ [-1,-2,-1],[0,0,0],[1,2,1] ]
    yMask = [ [1,0,-1],[2,0,-2],[1,0,-1] ]

    for row in range(1,grayImage.getHeight()-1):
        for col in range(1,grayImage.getWidth()-1):
            gx = convolve(grayImage,row,col,xMask)
            gy = convolve(grayImage,row,col,yMask)
            g = math.sqrt(gx**2 + gy**2)

            if g > 175:
                newim.setPixel(col,row,black)
            else:
                newim.setPixel(col,row,white)

    return newim
```


Figure 6.25

