Introduction to Computing and Programming in Python:

A Multimedia Approach

Chapter 4: Modifying Pixels in a Range

Chapter Learning Goals

The media learning goals for this chapter are:

- To mirror pictures horizontally or vertically.
- To compose pictures into one another and create collages.
- To rotate pictures.
- To scale pictures smaller and larger.

The computer science goals for this chapter are:

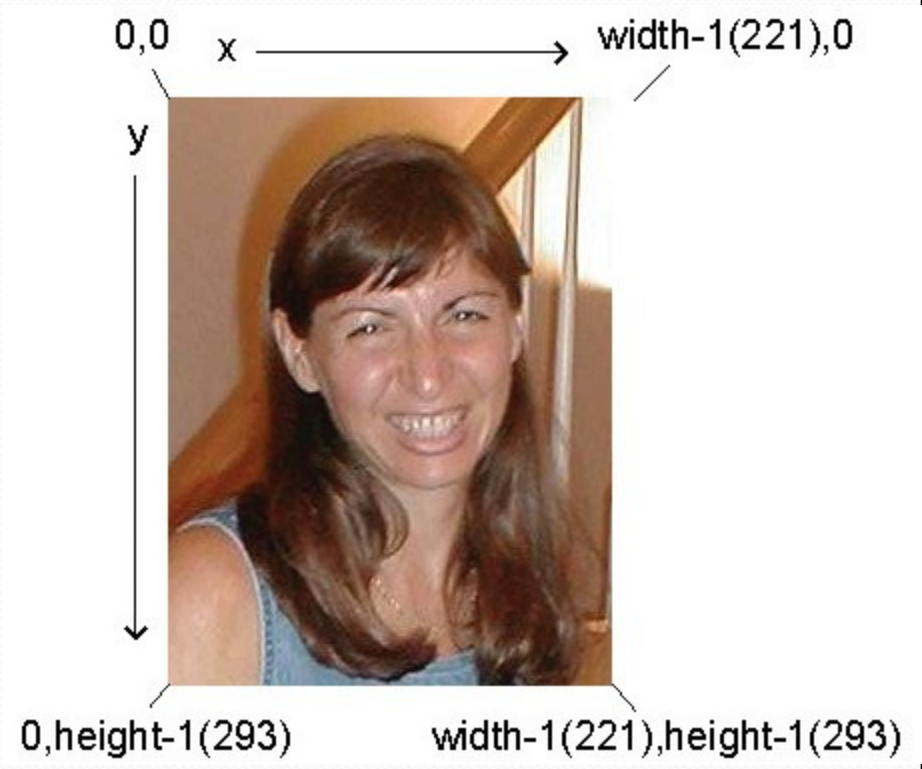
- To use nested loops for addressing elements of a matrix.
- To loop through only part of an array.
- To develop some debugging strategies—specifically, to use `print` statements to explore executing code.

Reminder: Pixels are in a matrix

- Matrices have two dimensions: A height and a width
- We can reference any element in the matrix with (x,y) or (horizontal, vertical)
  - We refer to those coordinates as *index numbers* or *indices*
- We sometimes want to know *where* a pixel is, and **getPixels** doesn't let us know that.

Pixels in a Matrix

- "Barbara.jpg" has
  - height 293 (bottommost index is 292) and
  - width 221 (rightmost index is 220)



Introducing the function range

- Range returns a sequence between its first two inputs, possibly using a third input as the increment

```
>>> print range(1,4)
[1, 2, 3]
>>> print range(-1,3)
[-1, 0, 1, 2]
>>> print range(1,10,2)
[1, 3, 5, 7, 9]
>>> print range(3)
[0,1,2]
```

Notice:

- **End value is never included.**
  - **range(0,10) ends at 9.**
- **If you leave out a start value, it's assumed to be zero.**

Side Note:
That thing in [] is a sequence

```
>>> a=[1,2,3]
>>> print a
[1, 2, 3]
>>> a = a + 4
An attempt was made to call a function with a parameter of an invalid type
>>> a = a + [4]
>>> print a
[1, 2, 3, 4]
>>> a[0]
1
```

We can assign names to sequences, print them, add items to sequences, and access individual pieces of them.

We can also use **for** loops to process each element of a sequence.

We can use range to generate index numbers

- We'll do this by working the range from 0 to the height-1, and 0 to the width-1.
  - Using the range function will make it easy to start from 0 and stop before the end value.
- But we'll need more than one loop.
  - Each for loop can only change one variable, and we need two for indexing a matrix

Working the pixels by number

- To use **range**, we'll have to use *nested loops*
  - One to walk the width, the other to walk the height
    - Be sure to watch your blocks (i.e., indentation) carefully!

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      px = getPixel(picture,x,y)
      value = getRed(px)
      setRed(px,value*1.1)
```

What's going on here?

The first time through the first loop, x is the name for 0.

We'll be processing the first column of pixels in the picture.

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      px = getPixel(picture,x,y)
      value = getRed(px)
      setRed(px,value*1.1)
```

Now, the inner loop

Next, we set y to 0. We're now going to process each of the pixels in the first column.

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      px = getPixel(picture,x,y)
      value = getRed(px)
      setRed(px,value*1.1)
```

Process a pixel

With x = 0 and y = 0, we get the leftmost pixel and increase its red by 10%

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      px = getPixel(picture,x,y)
      value = getRed(px)
      setRed(px,value*1.1)
```

Next pixel

Next we set y to 1 (next value in the sequence *range(0,getHeight(picture))*

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      px = getPixel(picture,x,y)
      value = getRed(px)
      setRed(px,value*1.1)
```

Process pixel (0,1)

x is still 0, and now y is 1, so increase the red for pixel (0,1)

```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      px = getPixel(picture,x,y)
      value = getRed(px)
      setRed(px,value*1.1)
```

We continue along this way, with y taking on every value from 0 to the height of the picture (minus 1).

Finally, next column

Now that we're done with the loop for y, we get back to the FOR loop for x.
x takes on the value 1, and we go back to the y loop to process all the pixels in the column x=1.

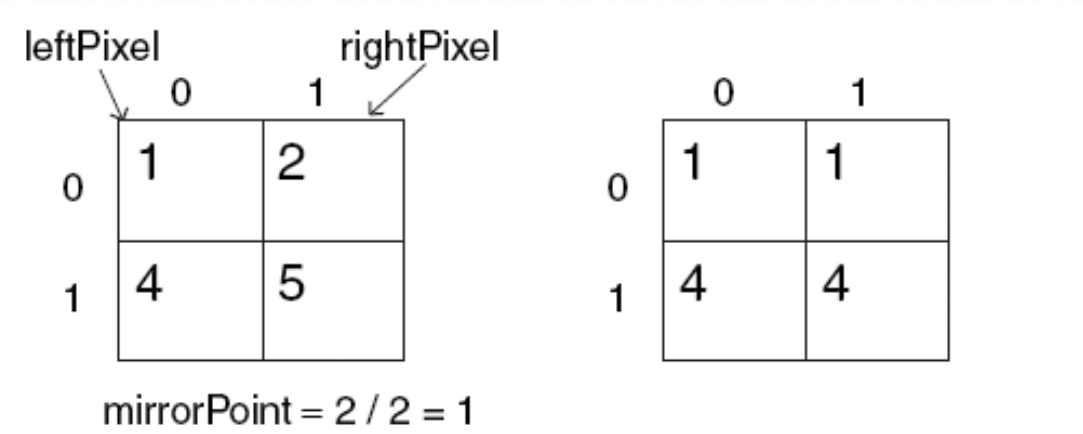
```
def increaseRed2(picture):
  for x in range(0,getWidth(picture)):
    for y in range(0,getHeight(picture)):
      px = getPixel(picture,x,y)
      value = getRed(px)
      setRed(px,value*1.1)
```

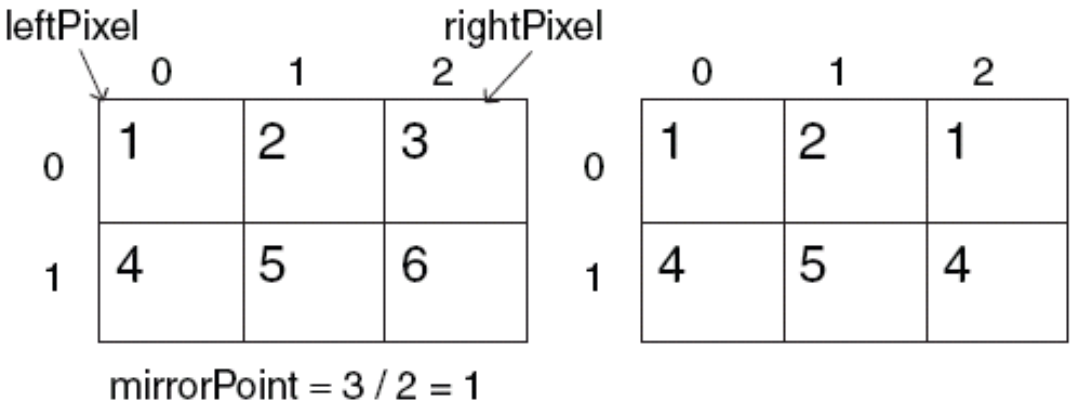
What can you do if you know where the pixels are? One answer: Mirroring

- Imagine a mirror horizontally across the picture, or vertically
- What would we see?
- How do generate that digitally?
  - We simply *copy* the colors of pixels from one place to another

Work it out with matrices

- mirrorPoint is halfway across: getWidth(picture)/2





If left pixel is at (x,y), right pixel is at (width-x-1,y)

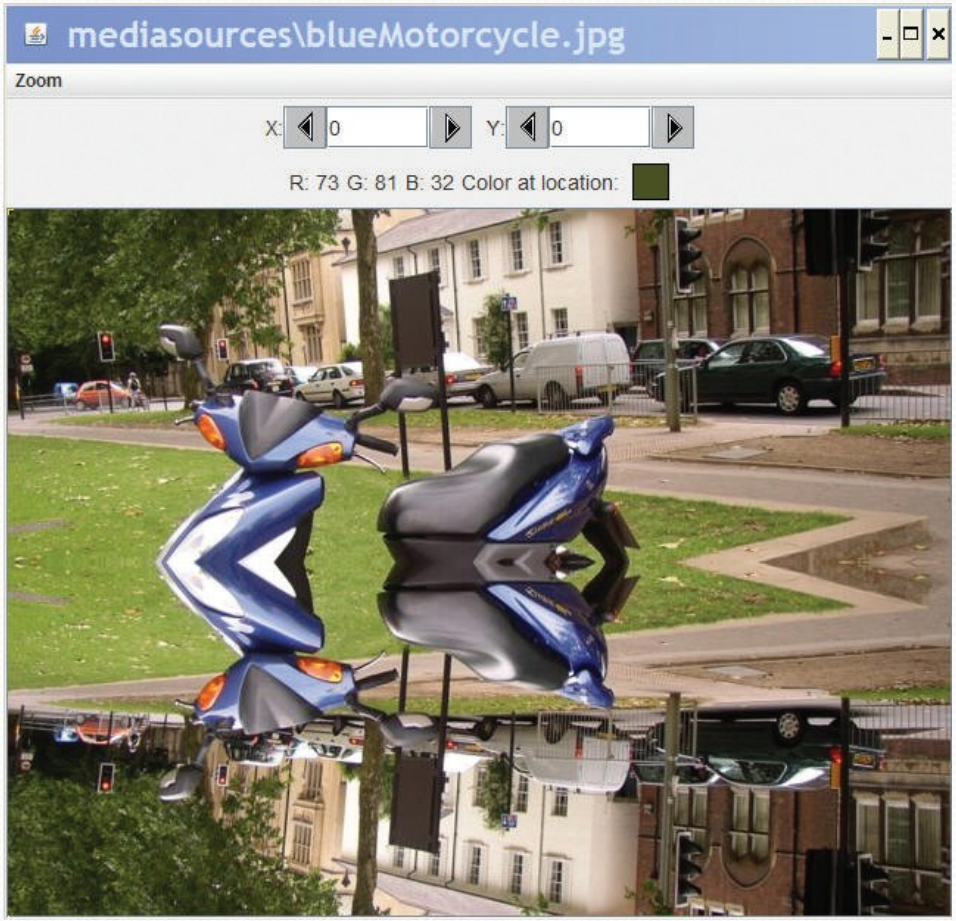
Recipe for mirroring


```
def mirrorVertical(source):
  mirrorPoint = getWidth(source) / 2
  width = getWidth(source)
  for y in range(0,getHeight(source)):
    for x in range(0,mirrorPoint):
      leftPixel = getPixel(source,x,y)
      rightPixel = getPixel(source,width - x - 1,y)
      color = getColor(leftPixel)
      setColor(rightPixel,color)
```

Can we do it with a horizontal mirror?

```
def mirrorHorizontal(source):
  mirrorPoint = getHeight(source) / 2
  height = getHeight(source)
  for x in range(0,getWidth(source)):
    for y in range(0,mirrorPoint):
      topPixel = getPixel(source,x,y)
      bottomPixel = getPixel(source,x,height - y - 1)
      color = getColor(topPixel)
      setColor(bottomPixel,color)
```

Of course!


mediasources\blueMotorcycle.jpg
Zoom
X: 0
Y: 0
R: 73 G: 81 B: 32 Color at location:

What if we wanted to copy bottom to top?

- Very simple: Swap the **order of pixels** in the bottom lines

```
def mirrorBotTop(source):
  mirrorPoint = getHeight(source) / 2
  height = getHeight(source)
  for x in range(0,getWidth(source)):
    for y in range(0,mirrorPoint):
      topPixel = getPixel(source,x,y)
      bottomPixel = getPixel(source,x,height - y - 1)
      color = getColor(bottomPixel)
      setColor(topPixel,color)
```

Mirroring bottom to top

Doing something useful with mirroring

- Mirroring can be used to create interesting effects, but it can also be used to create realistic effects.
- Consider this image from a trip to Athens, Greece.
  - Can we "repair" the temple by mirroring the complete part onto the broken part?

Figuring out where to mirror

- Use MediaTools to find the mirror point and the range that we want to copy

Writing functions for specific files… generally

- The function to mirror the temple needs to work for one and only one file.
- But we still don't want to write out the whole path.
  - setMediaPath() allows us to pick a directory where our media will be stored.
  - getMediaPath(filename) will *generate* the entire path for us to the filename *in the media directory*
  - *THIS ONLY WORKS WHEN WE'RE ACCESSING FILES IN THE MEDIA DIRECTORY AND WHERE WE HAVE SET THE PATH FIRST!*

Some Utility Functions

- If you *know* the name of the file, searching for it with **pickAFile()** feels tedious
- You can set and get a media *folder* (*path*) for remembering a place where your media will be coming from (or going to)
  - **setMediaPath()** lets you pick a file in your media folder
  - **getMediaPath(basefilename)** lets you generate a complete filename out of only the last part

Example

```
>>> setMediaPath()
New media folder: C:\Documents and Settings\Mark
Guzdial\My Documents\mediasources\
>>> getMediaPath("barbara.jpg")
'C:\\Documents and Settings\\Mark Guzdial\\My
Documents\\mediasources\\barbara.jpg'
>>> barb=makePicture(getMediaPath("barbara.jpg"))
```

Program to mirror the temple

```
def mirrorTemple():
  source = makePicture(getMediaPath("temple.jpg"))
  mirrorPoint = 276
  for x in range(13,mirrorPoint):
    for y in range(27,97):
      pleft = getPixel(source,x,y)
      pright = getPixel(source,mirrorPoint + mirrorPoint - 1 - x,y)
      setColor(pright,getColor(pleft))
  show(source)
  return source
```

Did it really work?

- It clearly did the mirroring, but that doesn't create a 100% realistic image.
- Check out the shadows: Which direction is the sun coming from?

Understanding the Temple Fix

- What is the very first transfer of pixels from and to? Which (x,y) pixel from? Which (x,y) pixel to?
- What is second?
- How many pixels get copied?

Adding print statements to see what's happening

```
def mirrorTemple():
  source = makePicture(getMediaPath("temple.jpg"))
  mirrorPoint = 276
  for x in range(13,mirrorPoint):
    for y in range(27,97):
      print "Copying color from",x,y, " to ",mirrorPoint + mirrorPoint - 1 - x, y
      pleft = getPixel(source,x,y)
      pright = getPixel(source,mirrorPoint + mirrorPoint - 1 - x,y)
      setColor(pright,getColor(pleft))
  show(source)
  return source
```

First pixels are either side of the mirrorpoint, then moving down

```
>>> p2=mirrorTemple()
Copying color from 13 27
   to  538 27
Copying color from 13 28
   to  538 28
Copying color from 13 29
   to  538 29
```

Counting pixels

```
def mirrorTemple():
  source = makePicture(getMediaPath("temple.jpg"))
  mirrorPoint = 276
  count = 0
  for x in range(13,mirrorPoint):
    for y in range(27,97):
      pleft = getPixel(source,x,y)
      pright = getPixel(source,mirrorPoint + mirrorPoint - 1 - x,y)
      setColor(pright,getColor(pleft))
      count = count + 1
  show(source)
  print "We copied",count,"pixels"
  return source
```

Counting pixels

```
>>> p2=mirrorTemple()
We copied 18410 pixels
```

- Where did that come from?
  - How many rows? Y goes from 27 to 97
    - = 70 rows of pixels
  - How many columns? X goes from 13 to 276
    - = 263 columns of pixels
  - 70 * 263 = 18410

Moving pixels *across* pictures

- We've seen using index variables to track the pixel position we're working with in a picture.
- We can copy *between* pictures, if we keep track of:
  - The *source* index variables
    - Where we're getting the pixels *from*
  - The *target* index variables
    - Where we're putting the pixels *at*
- (Not really copying the pixels: *Replicating* their color.)

What can you do then?

- What can you do when copying from one picture to another?
  - Collages: Copy *several* pictures onto one
  - Cropping: You don't have to take the *whole* picture
  - Scaling: Make a picture smaller, or larger when copying it

Blank files in mediasources

- getMediaPath("7inX95in.jpg") gives you a JPEG canvas which prints out as 7x9.5 inches
  - Letter-sized page with 1 inch margins
- getMediaPath("640x480.jpg") gives a JPEG canvas at a common size: 640 pixels across by 480 pixels high

Copying pixels

- In general, what we want to do is to keep track of a sourceX and sourceY, and a targetX and targetY.
  - We *increment* (add to them) in pairs
    - sourceX and targetX get incremented together
    - sourceY and targetY get incremented together
  - The tricky parts are:
    - Setting values *inside* the body of loops
    - Incrementing at the *bottom* of loops

Copying Barb to a canvas

```
def copyBarb():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 0
  for sourceX in range(0,getWidth(barb)):
    targetY = 0
    for sourceY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

Comments

- Python ignores from “#” through the rest of the line
- If you start a line with “#”, the whole line is ignored
- Why do we want lines to be *ignored?*
  - To be able to leave notes to ourselves or someone else about how the program works

Walking through the copying function

- First, get the source (barb) and target (canvas) files and pictures as names we can use later.

```
def copyBarb():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
 targetX = 0
 for sourceX in range(0,getWidth(barb)):
   targetY = 0
   for sourceY in range(0,getHeight(barb)):
     color = getColor(getPixel(barb,sourceX,sourceY))
     setColor(getPixel(canvas,targetX,targetY), color)
     targetY = targetY + 1
   targetX = targetX + 1
 show(barb)
 show(canvas)
 return canvas
```

The actual copy

- We get the color of the pixel at sourceX and sourceY
- We set (copy) the color to the pixel in the target picture at targetX and targetY

```
def copyBarb():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 0
  for sourceX in range(0,getWidth(barb)):
    targetY = 0
    for sourceY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

Setting up the copy loop

- targetX gets set to 0 at the beginning
- sourceX will range across the width of the source picture
- *INSIDE* the loop, we set targetY to 0
  - *Inside* because we want it to start at 0 each time we do a new X
- sourceY will range from 0 to one less height of source

```
def copyBarb():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 0
  for sourceX in range(0,getWidth(barb)):
    targetY = 0
    for sourceY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

Ending the loop

- Just before we end the sourceY loop, we increment targetY
  - **It's now set up for the next time through the loop**
  - **It's set correctly for the next value of sourceY**
- Just before we end the sourceX loop, we increment the targetX
  - **Note carefully the indentation to figure out which goes with which loop**

```
def copyBarb():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 0
  for sourceX in range(0,getWidth(barb)):
    targetY = 0
    for sourceY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

What's this naming something as itself?

- **targetX = targetX + 1**
- This isn't really naming something as itself
  - targetX + 1 is *evaluated*
    - It will result in the number after targetX
  - targetX = then sets the value of targetX
- The result is that targetX gets incremented by 1

Ending the copy function

- At the very end, we show the source and target
- And return the modified target.

```
def copyBarb():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 0
  for sourceX in range(0,getWidth(barb)):
    targetY = 0
    for sourceY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

Works either way

```
def copyBarb2():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  sourceX = 0
  for targetX in range(0,getWidth(barb)):
    sourceY = 0
    for targetY in range(0,getHeight(barb)):
      color =
     getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      sourceY = sourceY + 1
    sourceX = sourceX + 1
  show(barb)
  show(canvas)
  return canvas
```

As long as we increment sourceX and targetX together, and sourceY and targetY together, it doesn't matter which is in the for **loop and which is incremented via expression**

Transformation = Small changes in copying

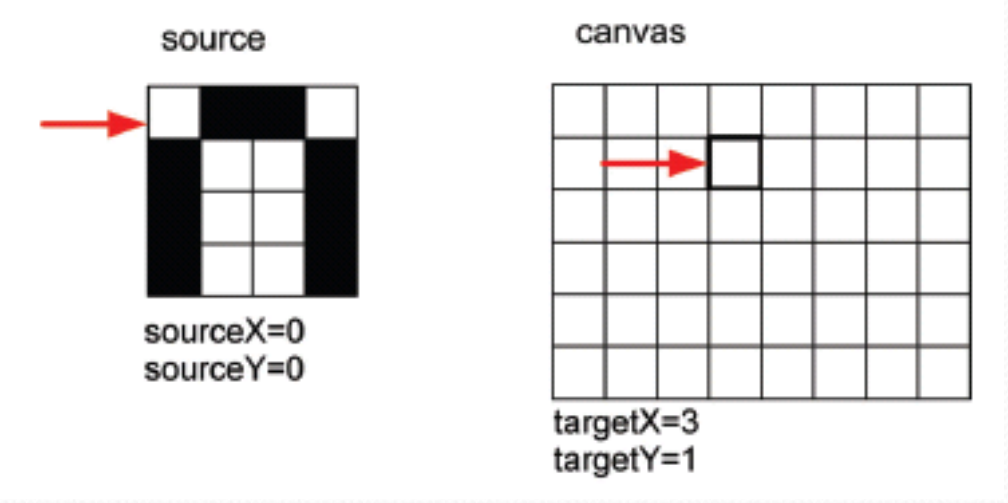
- Making relatively small changes in this basic copying program can make a variety of transformations.
  - Change the targetX and targetY, and you copy wherever you want
  - **Cropping**: Change the sourceX and sourceY range, and you copy only part of the program.
  - **Rotating**: Swap targetX and targetY, and you end up copying sideways
  - **Scaling**: Change the increment on sourceX and sourceY, and you either grow or shrink the image.

Copying into the middle of the canvas

```
def copyBarbMidway():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 100
  for sourceX in range(0,getWidth(barb)):
    targetY = 100
    for sourceY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

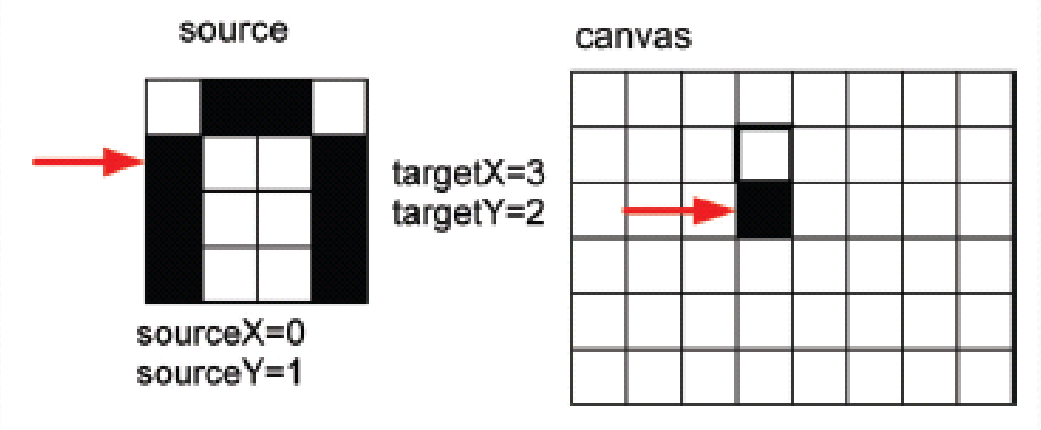
Copying: How it works

- Here's the initial setup:



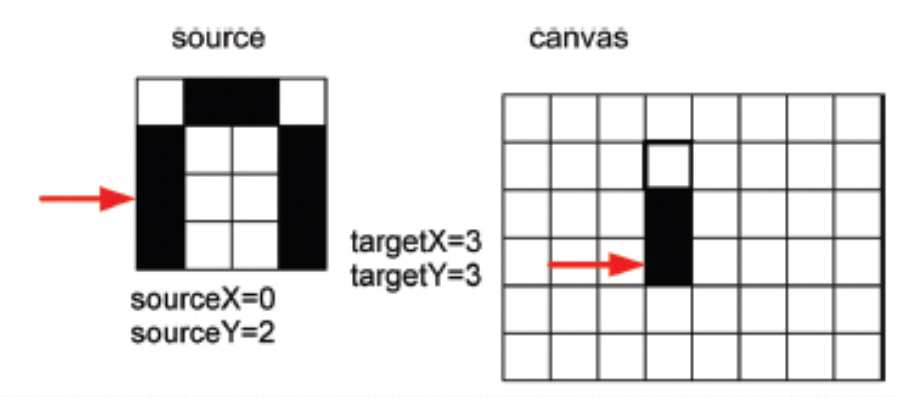
Copying: How it works 2

- After incrementing the sourceY and targetY once (whether in the **for** or via expression):



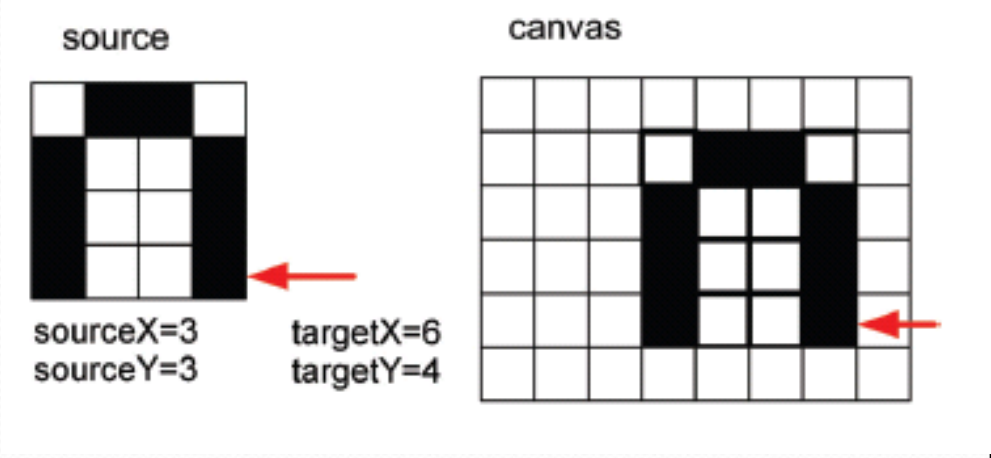
Copying: How it works 3

- After yet another increment of sourceY and targetY:
- When we finish that column, we increment sourceX and targetX, and start on the next column.



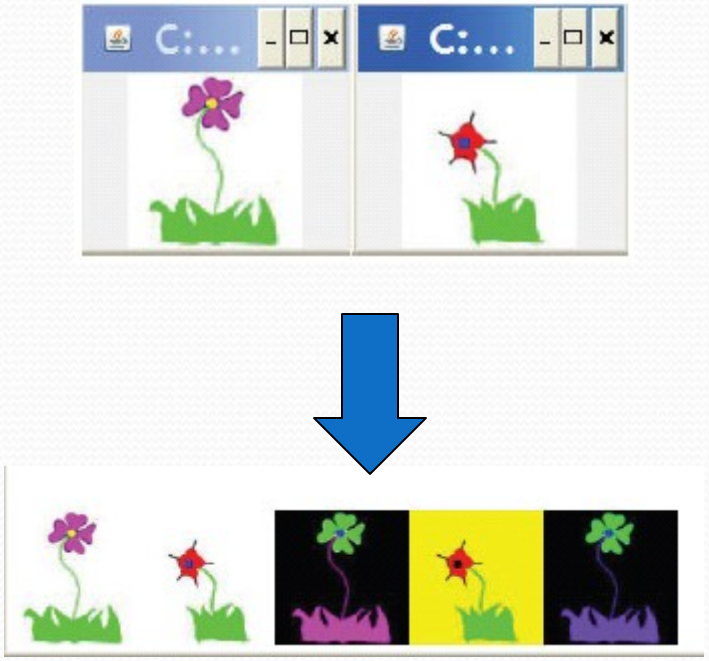
Copying: How it looks at the end

- Eventually, we copy every pixel



Making a collage

- Could we do something to the pictures we copy in?
  - Sure! Could either apply one of those functions *before* copying, or do something to the pixels *during* the copy.
- Could we copy more than one picture!
  - Of course! Make a collage!



```
def createCollage():
 flower1=makePicture(getMediaPath("flower1.jpg"))
 print flower1
 flower2=makePicture(getMediaPath("flower2.jpg"))
 print flower2
 canvas=makePicture(getMediaPath("640x480.jpg"))
 print canvas
 #First picture, at left edge
 targetX=0
 for sourceX in range(0,getWidth(flower1)):
  targetY=getHeight(canvas)-getHeight(flower1)-5
  for sourceY in range(0,getHeight(flower1)):
   px=getPixel(flower1,sourceX,sourceY)
   cx=getPixel(canvas,targetX,targetY)
   setColor(cx,getColor(px))
   targetY=targetY + 1
  targetX=targetX + 1
 #Second picture, 100 pixels over
 targetX=100
 for sourceX in range(0,getWidth(flower2)):
  targetY=getHeight(canvas)-getHeight(flower2)-5
  for sourceY in range(0,getHeight(flower2)):
   px=getPixel(flower2,sourceX,sourceY)
   cx=getPixel(canvas,targetX,targetY)
   setColor(cx,getColor(px))
   targetY=targetY + 1
  targetX=targetX + 1
```

Page 91-92 (2ed edition)

```
 #Third picture, flower1 negated
 negative(flower1)
 targetX=200
 for sourceX in range(0,getWidth(flower1)):
  targetY=getHeight(canvas)-getHeight(flower1)-5
  for sourceY in range(0,getHeight(flower1)):
   px=getPixel(flower1,sourceX,sourceY)
   cx=getPixel(canvas,targetX,targetY)
   setColor(cx,getColor(px))
   targetY=targetY + 1
  targetX=targetX + 1
 #Fourth picture, flower2 with no blue
 clearBlue(flower2)
 targetX=300
 for sourceX in range(0,getWidth(flower2)):
  targetY=getHeight(canvas)-getHeight(flower2)-5
  for sourceY in range(0,getHeight(flower2)):
   px=getPixel(flower2,sourceX,sourceY)
   cx=getPixel(canvas,targetX,targetY)
   setColor(cx,getColor(px))
   targetY=targetY + 1
  targetX=targetX + 1
 #Fifth picture, flower1, negated with decreased red
 decreaseRed(flower1)
 targetX=400
 for sourceX in range(0,getWidth(flower1)):
  targetY=getHeight(canvas)-getHeight(flower1)-5
  for sourceY in range(0,getHeight(flower1)):
   px=getPixel(flower1,sourceX,sourceY)
   cx=getPixel(canvas,targetX,targetY)
   setColor(cx,getColor(px))
   targetY=targetY + 1
  targetX=targetX + 1
 show(canvas)
 return(canvas)
```

Can we make that easier?

- The collage code is long, yet simple.
- It's the same thing over-and-over.
- We can ***generalize*** that copying loop, and with ***parameters***, use it in many places.

```
def copy(source, target, targX, targY):
  targetX = targX
  for sourceX in range(0,getWidth(source)):
    targetY = targY
    for sourceY in
     range(0,getHeight(source)):
      px=getPixel(source,sourceX,sourceY)
      tx=getPixel(target,targetX,targetY)
      setColor(tx,getColor(px))
      targetY=targetY + 1
    targetX=targetX + 1
```

Exact same collage!

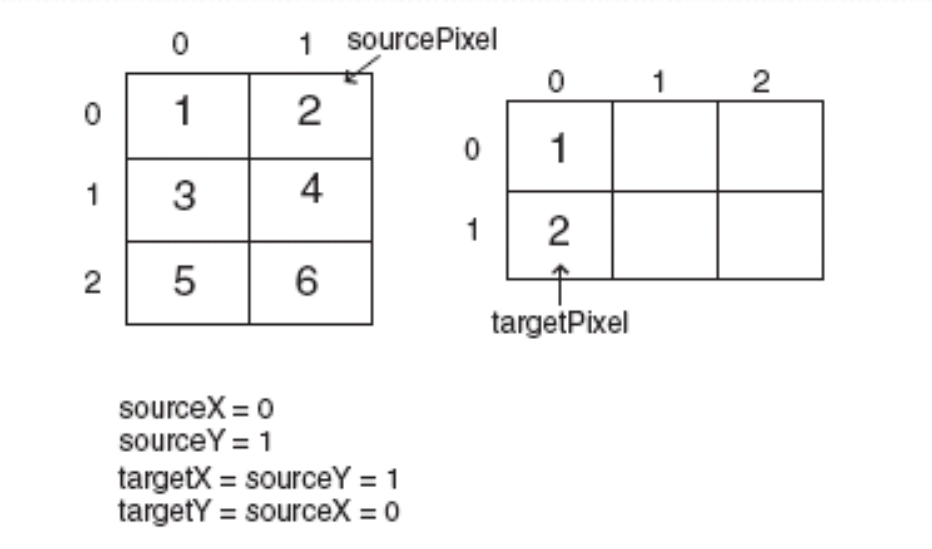
```
def createCollage2():

    flower1=makePicture(getMediaPath("flower1.jpg"))
    print flower1

    flower2=makePicture(getMediaPath("flower2.jpg"))
    print flower2

    canvas=makePicture(getMediaPath("640x480.jpg"))
    print canvas
     #First picture, at left edge
    copy(flower1,canvas,0,getHeight(canvas)-getHeight(flower1)-5)
    #Second picture, 100 pixels over

    copy(flower2,canvas,100,getHeight(canvas)-getHeight(flower2)-5)

    #Third picture, flower1 negated
    negative(flower1)

    copy(flower1,canvas,200,getHeight(canvas)-getHeight(flower1)-5)
     #Fourth picture, flower2 with no blue
    clearBlue(flower2)

    copy(flower2,canvas,300,getHeight(canvas)-getHeight(flower2)-5)
     #Fifth picture, flower1, negated with decreased red
    decreaseRed(flower1)

    copy(flower1,canvas,400,getHeight(canvas)-getHeight(flower2)-5)
    return canvas
```

Rotating the copy

```
def copyBarbSideways():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 0
  for sourceX in range(0,getWidth(barb)):
    targetY = 0
    for sourceY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetY,targetX), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

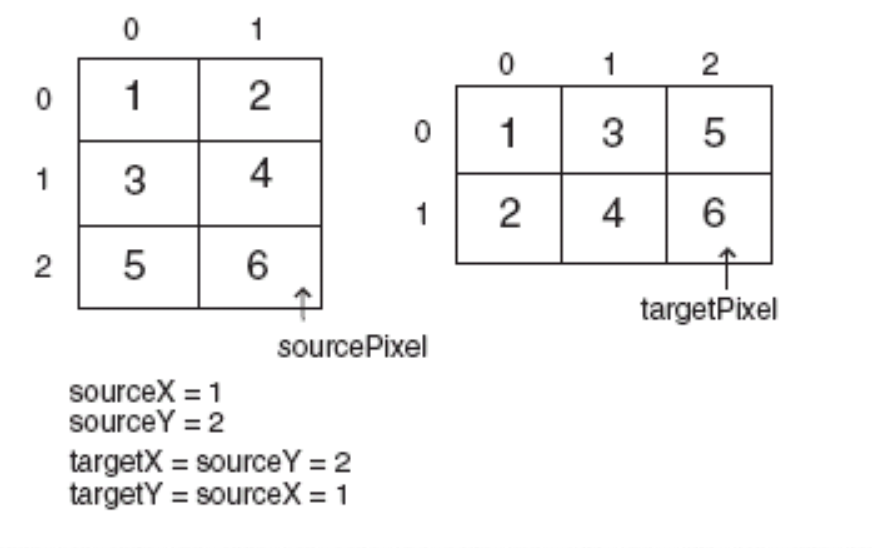
Rotating: How it works

- We increment the same, but we use **targetX** for the ***Y coordinate*** and **targetY** for the ***X coordinate***



Rotate: How it ends

- Same amount of increment, even same values in the variables, but a different result.



Doing a *real* rotation

```
def rotateBarbSideways():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 0
  width = getWidth(barb)
  for sourceX in range(0,getWidth(barb)):
    targetY = 0
    for sourceY in range(0,getHeight(barb)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetY,width - targetX
     - 1), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

Cropping: Just the face

```
def copyBarbsFace():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  targetX = 100
  for sourceX in range(45,200):
    targetY = 100
    for sourceY in range(25,200):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      targetY = targetY + 1
    targetX = targetX + 1
  show(barb)
  show(canvas)
  return canvas
```

Cropping, another way

```
def copyBarbsFace2():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  sourceX = 45
  for targetX in range(100,100+(200-45)):
    sourceY = 25
    for targetY in range(100,100+(200-25)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      sourceY = sourceY + 1
    sourceX = sourceX + 1
  show(barb)
  show(canvas)
  return canvas
```

Scaling

- Scaling a picture (smaller or larger) has to do with *sampling* the source picture differently
  - When we just copy, we *sample* every pixel
  - If we want a smaller copy, we skip some pixels
    - We *sample* fewer pixels
  - If we want a larger copy, we duplicate some pixels
    - We *over-sample* some pixels

Scaling the picture down

```
def copyBarbsFaceSmaller():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  sourceX = 45
  for targetX in range(100,100+((200-45)/2)):
    sourceY = 25
    for targetY in range(100,100+((200-25)/2)):
      color = getColor(getPixel(barb,sourceX,sourceY))
      setColor(getPixel(canvas,targetX,targetY), color)
      sourceY = sourceY + 2
    sourceX = sourceX + 2
  show(barb)
  show(canvas)
  return canvas
```

Scaling Up: Growing the picture

- To grow a picture, we simply duplicate some pixels
- We do this by incrementing by 0.5, but only use the integer part.

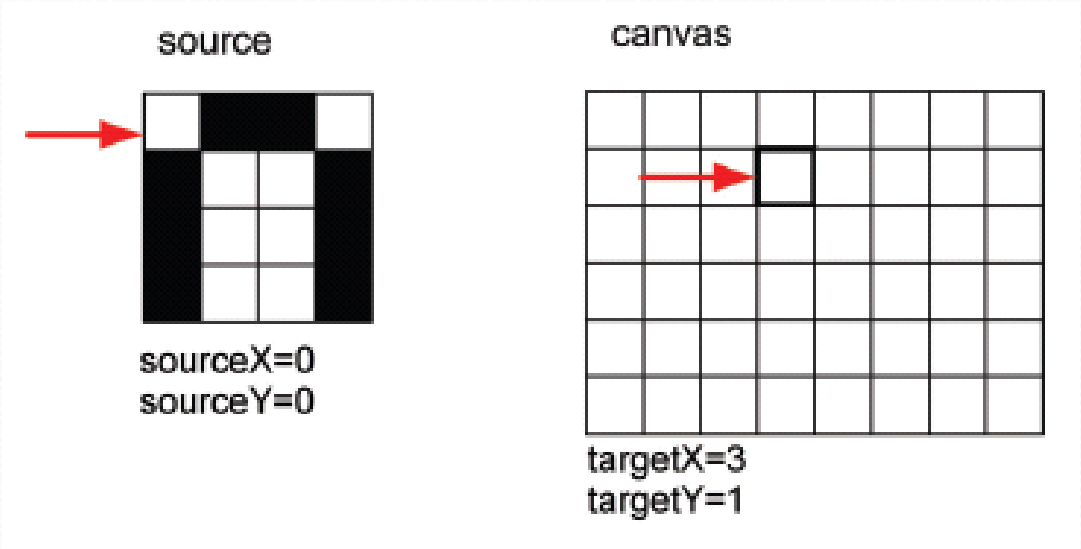
```
>>> print int(1)
1
>>> print int(1.5)
1
>>> print int(2)
2
>>> print int(2.5)
2
```

Scaling the picture up

```
def copyBarbsFaceLarger():
  # Set up the source and target pictures
  barbf=getMediaPath("barbara.jpg")
  barb = makePicture(barbf)
  canvasf = getMediaPath("7inX95in.jpg")
  canvas = makePicture(canvasf)
  # Now, do the actual copying
  sourceX = 45
  for targetX in range(100,100+((200-45)*2)):
    sourceY = 25
    for targetY in range(100,100+((200-25)*2)):
      color = getColor(getPixel(barb,int(sourceX),int(sourceY)))
      setColor(getPixel(canvas,targetX,targetY), color)
      sourceY = sourceY + 0.5
    sourceX = sourceX + 0.5
  show(barb)
  show(canvas)
  return canvas
```

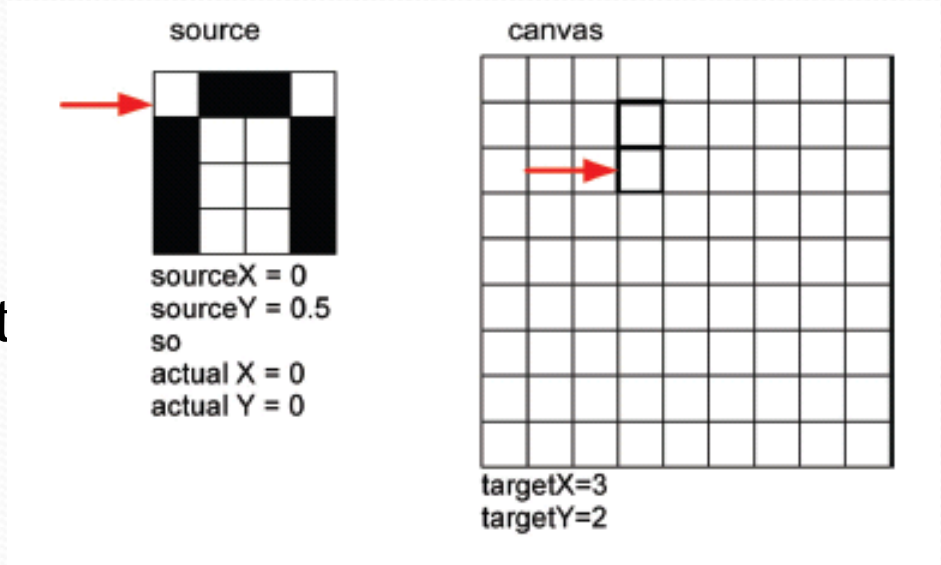
Scaling up: How it works

- Same basic setup as copying and rotating:



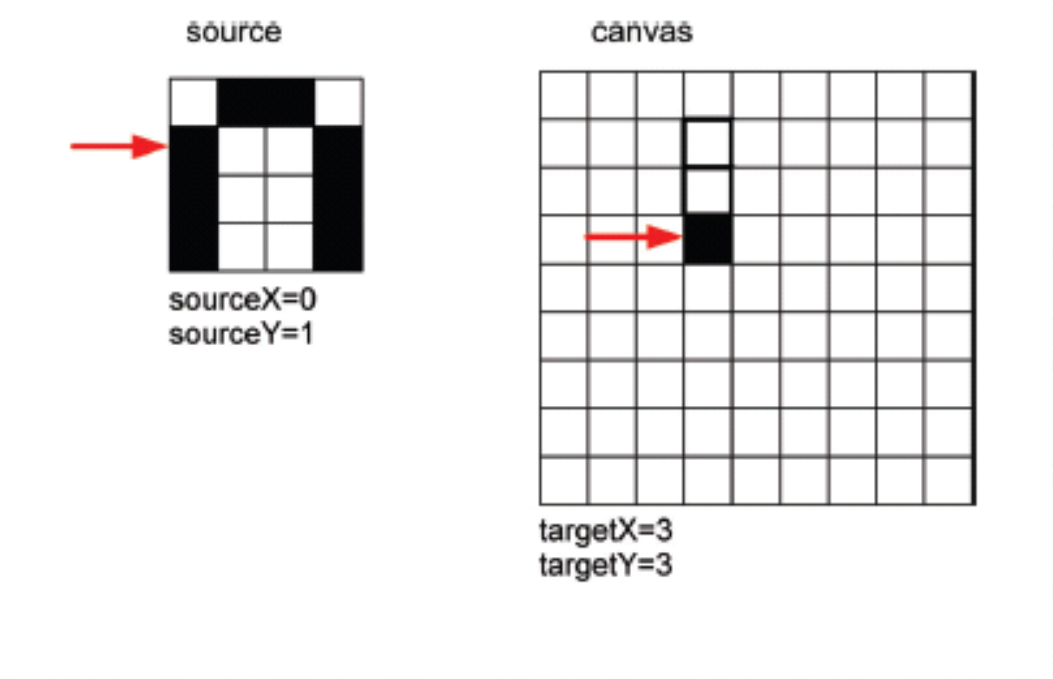
Scaling up: How it works 2

- But as we increment by *only* 0.5, and we use the **int()** function, we end up taking every pixel *twice*.
- Here, the blank pixel at (0,0) in the source gets copied twice onto the canvas.



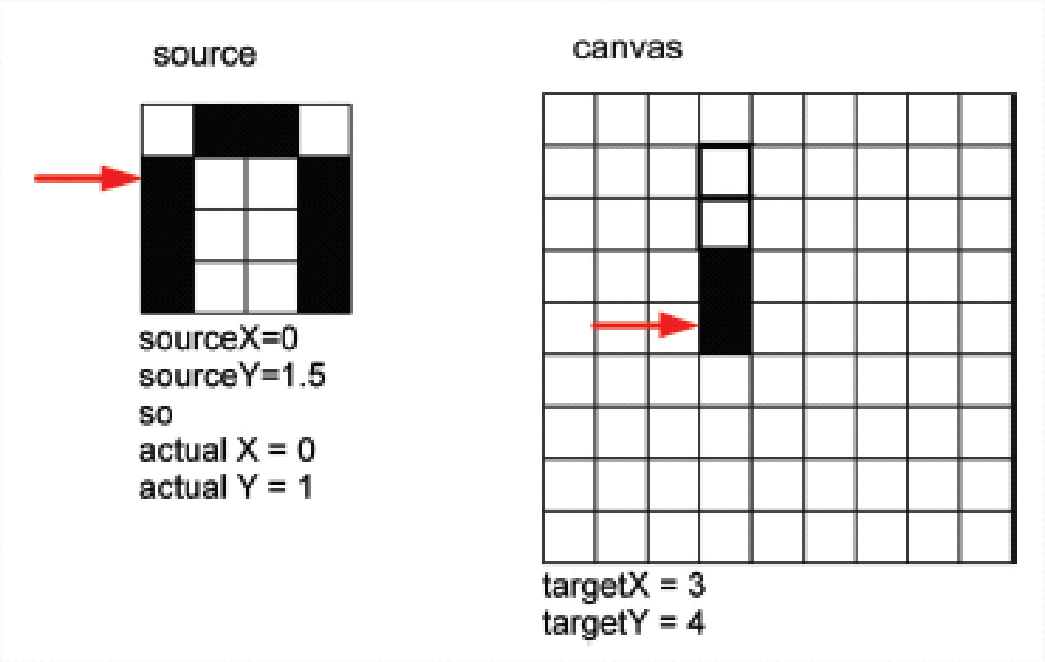
Scaling up: How it works 3

- Black pixels gets copied once…



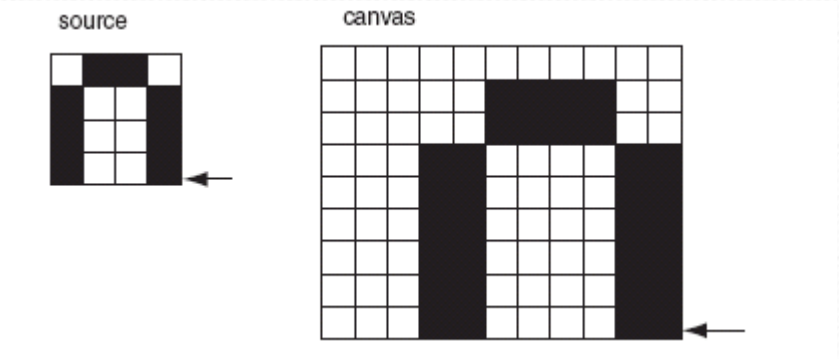
Scaling up: How it works 4

- And twice…



Scaling up: How it ends up

- We end up in the same place in the source, but twice as much in the target.
- Notice the degradation:
  - Gaps that weren't there previously
  - Curves would get "choppy": Pixelated



What to do?

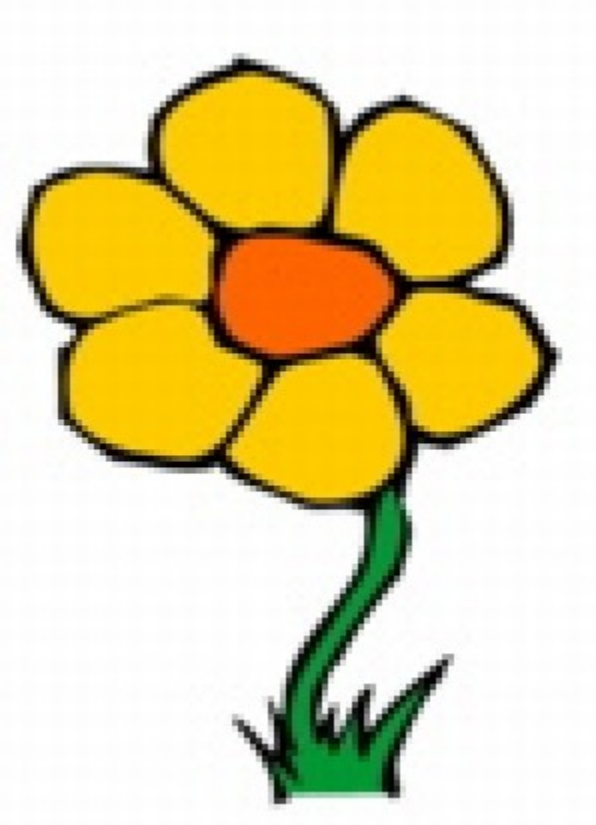
- How do we clear up the degradation of scaling up?
- Variety of techniques, but mostly following the same basic idea:
  - Use the pixels around to figure out what color a new pixel *should* be, then somehow (e.g., by averaging) compute the right color.
  - Different techniques look at different pixels and compute different averages in different ways.

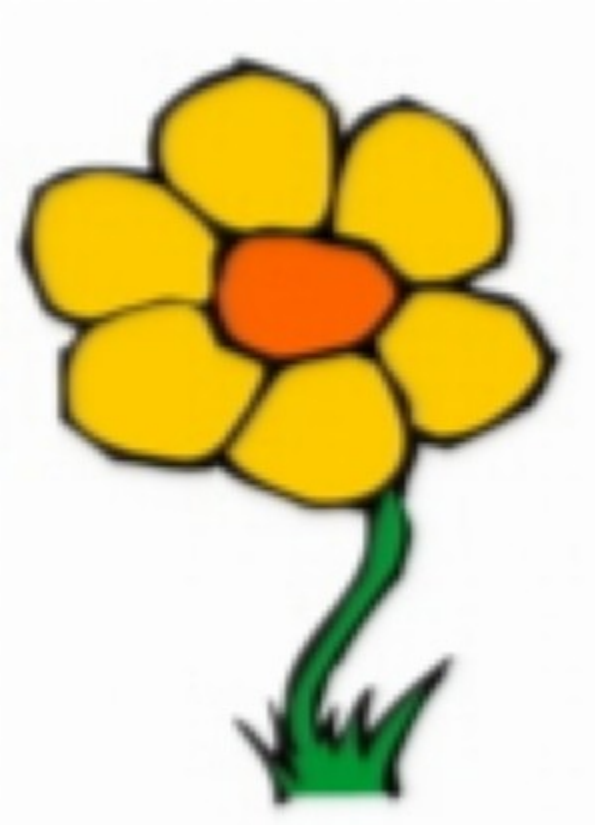
A blurring recipe

```
def blur(pic,size):
  for pixel in getPixels(pic):
    currentX = getX(pixel)
    currentY = getY(pixel)
    r = 0
    g = 0
    b = 0
    count = 0
    for x in range(currentX - size,currentX + size):
      for y in range(currentY - size, currentY + size):
        if(x<0) or (y<0) or (x >= getWidth(pic)) or (y >=getHeight(pic)):
          pass  # Skip if we go off the edge
        else:
          r = r + getRed(getPixel(pic,x,y))
          g = g + getGreen(getPixel(pic,x,y))
          b = b + getBlue(getPixel(pic,x,y))
          count = count + 1
    newColor = makeColor(r/count,g/count,b/count)
    setColor(pixel,newColor)
```

We'll see pass **and** else **later, but you can probably get a sense here of what's going on.**

Blurring out the pixelation





Things to try:

- Can you come up with general copy, rotate, copy, and scale functions?
  - Take input pictures and parameters
  - Return the canvas the correct transformation applied
- Also think about generalizing the transformations:
  - Scaling up and down by non-integer amounts
  - Rotating by something other than 90 degree increments