



COMP2720: Automating Tools for New Media

Scaling pictures up, and more
on posterising



Learning objectives of this lecture

- How to scale picture up
- Problems with scaling up: pixelation
- Improving posterisation
- Make posterisation more general using lists
- Extreme posterising

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
```

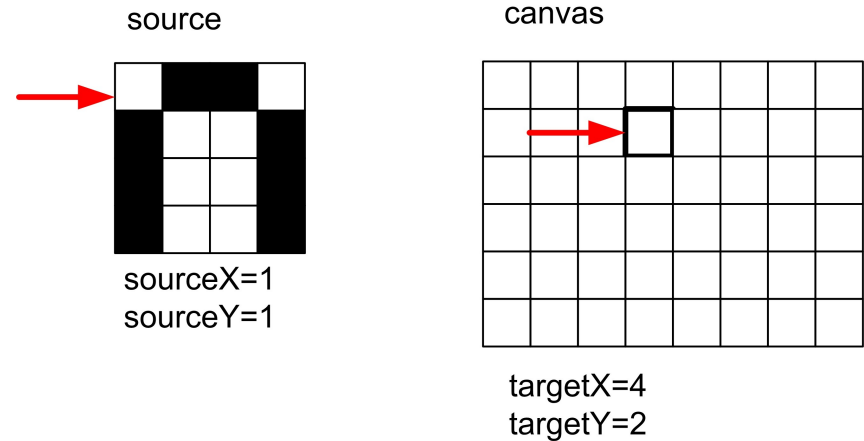
Scaling a picture up

```
def copyBarbsFaceLarger():
    # Set up the source and target pictures
    barbf=getMediaPath('barbara.jpg')
    barb = makePicture(barbf)
    canvas = makeEmptyPicture(500, 600)
    # 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
```



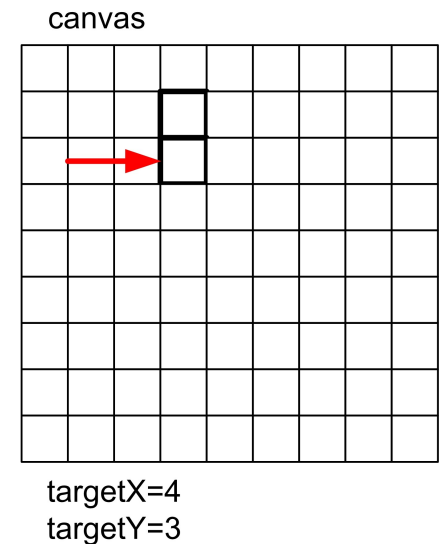
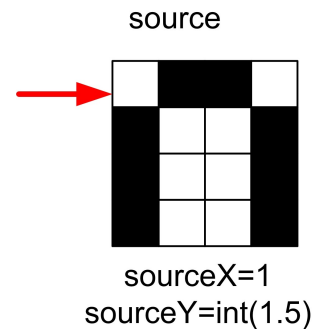
Scaling up: How it works

- Same basic setup as copying and rotating:



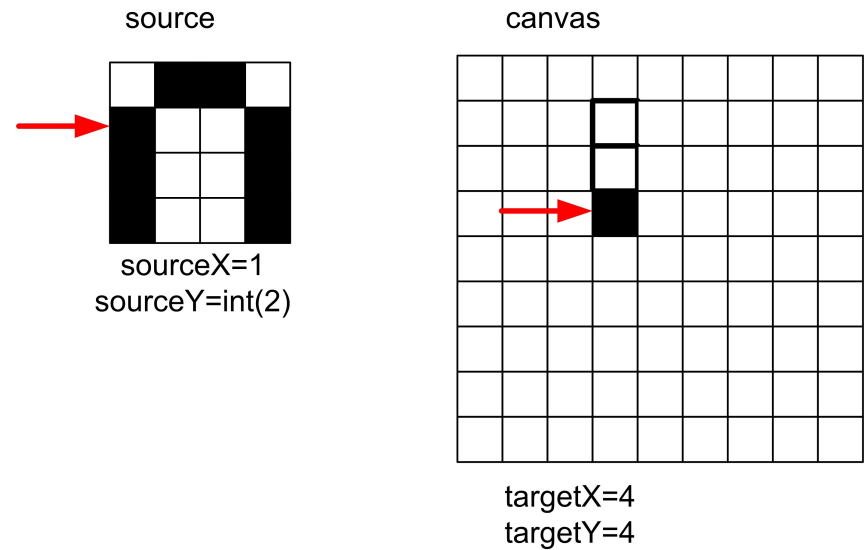
Scaling up: How it works 2

- 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 (1,1) in the source gets copied twice onto the canvas.



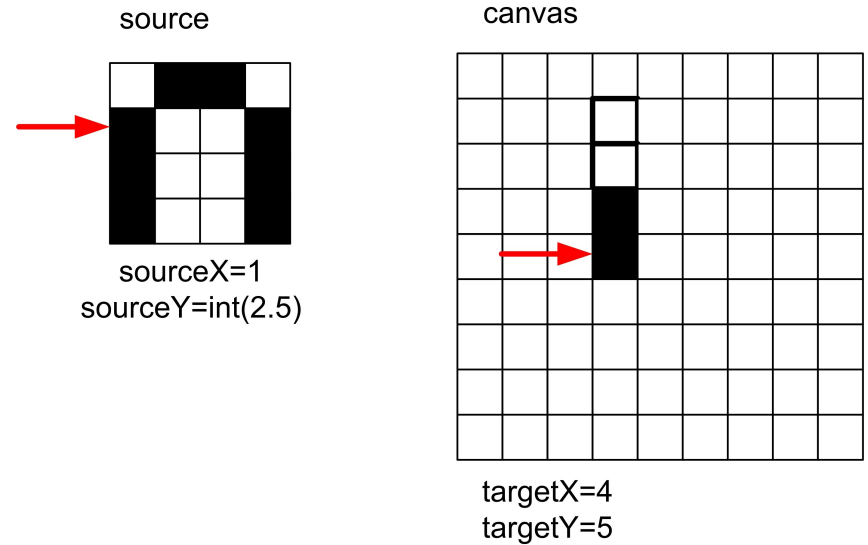
Scaling up: How it works 3

- Black pixels gets copied once...



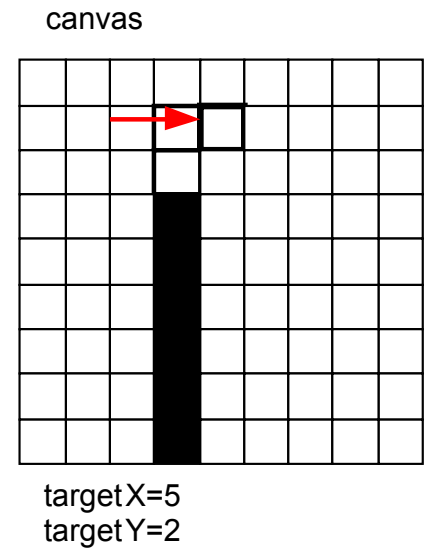
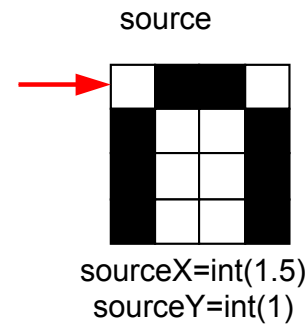
Scaling up: How it works 4

- And twice...



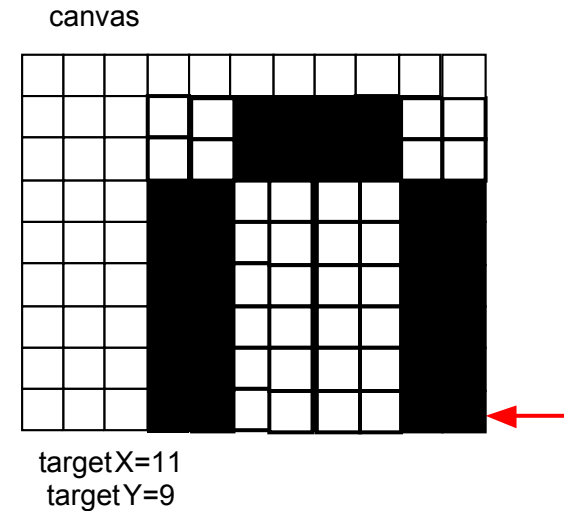
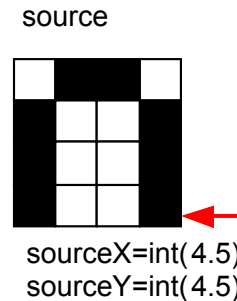
Scaling up: How it works 5

- The next “column” (x) in the target is the same “column” (x) in the source.



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:
 - Curves get “choppy”:
Pixelated



Things to try:

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

Recall *posterising*

- Posterising is about reducing the range of colors in a picture.
- For example, for all red values between 64 and 129, make it 95.
 - That reduces $(129-64)$ 65 possible red values to just 1.

Posterising example code

```
def posterise(pic):
```

```
    # Loop through the pixels
```

```
    for p in getPixels(pic):
```

```
        #get the RGB values
```

```
        red = getRed(p)
```

```
        green = getGreen(p)
```

```
        blue = getBlue(p)
```

```
    # Check and set red values
```

```
    if (red < 64):
```

```
        setRed(p, 31)
```

```
    if (red > 63 and red < 128):
```

```
        setRed(p, 95)
```

```
    if (red > 127 and red < 192):
```

```
        setRed(p, 159)
```

```
    if (red > 191 and red < 256):
```

```
        setRed(p, 223)
```

```
    # Check and set green values
```

```
    if (green < 64):
```

```
        setGreen(p, 31)
```

```
    if (green > 63 and green < 128):
```

```
        setGreen(p, 95)
```

```
    if (green > 127 and green < 192):
```

```
        setGreen(p, 159)
```

```
    if (green > 191 and green < 256):
```

```
        setGreen(p, 223)
```

```
    # Check and set blue values
```

```
    if (blue < 64):
```

```
        setBlue(p, 31)
```

```
    if (blue > 63 and blue < 128):
```

```
        setBlue(p, 95)
```

```
    if (blue > 127 and blue < 192):
```

```
        setBlue(p, 159)
```

```
    if (blue > 191 and blue < 256):
```

```
        setBlue(p, 223)
```

Before and After





Wasn't that painful to type in?

- That's a long, tedious, boring program to type in.
- Could we make it easier?
- What if we used a separate function to figure out the right value from a set of ranges, then just reused that function?
 - It would save us repeatedly typing all those if's!

How we can pick from 3 values?

```
def pickFrom3Ranges(input,lowerbound,upperbound,lowvalue,midvalue,uppervalue):  
    if (input <= lowerbound):  
        return lowvalue  
    if (input > lowerbound) and (input <= upperbound):  
        return midvalue  
    if (input > upperbound):  
        return uppervalue
```

Compare these:

```
# Check and set red values
```

```
if (red <= 64):
```

```
    setRed(p, 31)
```

```
if (red > 64 and red <= 128):
```

```
    setRed(p, 95)
```

```
if (red >= 128):
```

```
    setRed(p, 223)
```

```
# Check and set red values
```

```
newvalue = pickFrom3Ranges(red,64,128,31,95,223)
```

```
setRed(p,newvalue)
```

But...we had four ranges

- In our posterise function, we actually had *four* ranges.
- **EXERCISE:** Write `pickFrom4Ranges()` and rewrite `posterise()` using that.

```
# Check and set red values
if (red < 64):
    setRed(p, 31)
if (red > 63 and red < 128):
    setRed(p, 95)
if (red > 127 and red < 192):
    setRed(p, 159)
if (red > 191 and red < 256):
    setRed(p, 223)
```



Could we write this to handle *any* number of ranges?

- How would we describe the ranges?
 - How would we say them to the computer?
 - How would we represent the ranges on the computer?
- If we can figure out how to say it, we can make the computer to do it.



Suggestion: Use lists

- Represent the start and end of each range, and the return value for that range.
- Python lists: [1, 2, 3, 4]

Trying the list representation

```
# Check and set red values
if (red < 64):
    setRed(p, 31)
if (red > 63 and red < 128):
    setRed(p, 95)
if (red > 127 and red < 192):
    setRed(p, 159)
if (red > 191 and red < 256):
    setRed(p, 223)
```

- A list of lists (one for each of the four ranges)

```
[[0,63,31], [64,127,95], [128,191,159], [192,255,223]]
```

Handling any range

```
def pickFromRanges(input, ranges):  
    for r in ranges:  
        if (input >= r[0]) and (input <= r[1]):  
            return r[2]  
    # If we get here, something weird happened  
    return 0
```

The full new posterise code

```
def posterise2(pic):
    # Loop through the pixels
    for p in getPixels(pic):
        # Get the RGB values
        red = getRed(p)
        green = getGreen(p)
        blue = getBlue(p)

        # Check and set red values
        newred =
            pickFromRanges(red,[[0,63,31],[64,127,95],[128,191,159],
            ],[192,255,223]])
        setRed(p, newred)

        # Check and set green values
        newgreen =
            pickFromRanges(green,[[0,63,31],[64,127,95],[128,191,159],
            ],[192,255,223]])
        setGreen(p, newgreen)

        # Check and set blue values
        newblue =
            pickFromRanges(blue,[[0,63,31],[64,127,95],[128,191,159],
            ],[192,255,223]])
        setBlue(p, newblue)
```

```
def pickFromRanges(input,ranges):
    for r in ranges:
        if (input >= r[0]) and (input <= r[1]):
            return r[2]
    # If we get here, something weird happened
    return 0
```

**Even better: Define ranges list!
(avoid repetition)**

```
ranges = [[0,63,31], [64,127,95], \
          [128,191,159], [192,255,223]]
```

Extreme posterising!

- What if we don't have a *range*, per se, but only two specific values?
- Let's compute a luminance value, by averaging red, green, and blue.
- If it's more light than dark, make it *completely* light (white).
- If it's more dark than light, make it *completely* dark (black).

Two-levels of greyscale posterising

```
def greyposterise(pic):  
    for p in getPixels(pic):  
        r = getRed(p)  
        g = getGreen(p)  
        b = getBlue(p)  
        luminance = (r+g+b)/3  
        if (luminance < 128):  
            setColor(p,black)  
        if (luminance >= 128):  
            setColor(p,white)
```

Before and after 2-level posterising



Different example





Can we write 2-level using our range function?

- Does it make it any easier to type?
- Or is it about the same?

```
ranges = [[0,127,0], [128,255,255]]
```



Things to do:

- Start working on assignment 1... now!!