Chapter 4: Nested For Loops
Pixels in a Matrix

- Pictures are stored as a matrix
- Matrices have two dimensions: height and width
- We can reference any element in the matrix with \((x,y)\) or \((\text{horizontal, vertical})\)
  - We refer to those coordinates as **index numbers** or **indices**
- Example: “Barbara.jpg” has
  - Height: 294 pixels
    - bottommost index is 293
  - Width: 222 pixels
    - rightmost index is 221
Pixels are in a matrix

- We sometimes want to know where a pixel is
  - For example, we want to manipulate only part of the picture.
- **getPixels** doesn’t let us know where a pixel is
  - **getPixels** simply generates a linear list of all the pixels in the picture, but it does not know the original location of the pixels.
Introducing the function `range()`

- `range()` creates a sequence of numbers
- It takes two integer inputs, a starting point and an ending point
- It returns a sequence of numbers between the starting point and the ending point
  - The ending point is not included in the sequence
- There is a third optional input: the increment between the elements in the sequence
  - The default increment is 1
The function `range()`

```python
>>> 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.
- If you leave out a start value, it’s assumed to be zero.
We can use `range()` to generate index numbers for pictures

- We’ll do this by working the range from 0 to `height-1`, and the range from 0 to `width-1`.
- But we’ll need more than one loop.
- If we want to know both, the \textit{x} and \textit{y} values for a pixel, we will have to use two \texttt{for} loops:
  - one index for the width values, and
  - one index for the height values
Nested for loops

- Nesting is to put one thing inside another
- When we nest loops we write one loop inside another
- We say then, that the inner loop is nested inside the outer loop

```python
def changeRed2(picture, amount):
    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 * amount)
```

Outer loop

Inner loop
Working with nested loops

- When working with nested loops, we first enter the outer loop for the first iteration.
- Once we reach the inner loop, we must work with this loop until we have gone through all its iterations.
- Then, once we are completely done with the inner loop, we go back for a second iteration of the outer loop.
- We repeat this process until all the iterations of the outer loop are completed.
Understanding nested loops

```python
for i in range(1, 5): # [1, 2, 3, 4]
    for j in range(1, 3): # [1, 2]
        print i, j
```

• The outer loop will be iterated a total of 4 times
• The inner loop will be iterated 2 times, each time the outer loop is iterated
• Therefore, the inner loop is iterated a total of 8 times

Output:

1, 1
1, 2
2, 1
2, 2
3, 1
3, 2
4, 1
4, 2
Using nested for loops to manipulate pictures

- We will use `range()` and *nested loops*
  - One loop to walk the width, the other loop to walk the height
  - Be sure to watch your blocks (i.e., indentation) carefully!

```python
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 * 1.10)
```
Some Utility Functions

• When you want to deal with several pieces of media in the same directory 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 the media directory using a directory chooser
  • `getMediaPath(baseFilename)` lets you generate a complete path to a file out of only the file base name
    • You must use `setMediaPath()` first
    • Example: `>>>file = getMediaPath("arch.jpg")`
Blank canvas files in media sources

- `getMediaPath("7inX95in.jpg")` gives you the path to a JPEG canvas of size 7 by 9.5 inches
  - Letter-sized page with 1 inch margins
- `getMediaPath("640x480.jpg")` gives you the path to a JPEG canvas of size: 640 pixels width by 480 pixels high
- `getMediaPath("400x300.jpg")` gives you the path to a JPEG canvas of size: 400 pixels width by 300 pixels high
- `makeEmptyPicture(w, h)` creates a picture of a desired width and height (both specified in pixels)
Some useful constants

- A constant is a value that does not change during the execution of a program
- JES has some pre-defined constants for colors:

  - black
  - white
  - blue
  - red
  - green
  - gray
  - darkGray
  - ligthGray
  - yellow
  - orange
  - pink
  - magenta
  - cyan
The `return` keyword

- The `return` keyword specifies what the return value is from our function.
- This keyword also ends the function
  - Once `return` is executed, no more statements in the functions are executed
- Functions should use the keyword `return` when they produce something we want to use in the future
  - Otherwise the object created by our function will disappear when the function ends
Moving pixels *across* pictures

- 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*
- Note: We are not really copying the pixels, instead we are *replicating* their color.
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* in pairs
  - sourceX and targetX get incremented together
  - sourceY and targetY get incremented together
- You need to pay careful attention to:
  - Setting values *inside* the body of loops
  - Incrementing at the *bottom* of loops
Copy / Duplicate picture

Source

Target
The basic “copy” algorithm

```python
def copyPicture(source) :
    width = getWidth(source)
    height = getHeight(source)
    target = makeEmptyPicture(width, height)

    targetX = 0
    for sourceX in range (0, width) :
        targetY = 0 ;
        for sourceY in range (0, height) :
            color = getColor(getPixel(source, sourceX, sourceY))
            setColor (getPixel(target, targetX, targetY), color)
            targetY = targetY +1
        targetX = targetX + 1
    return target
```
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
  - Rotating: flip the image across a diagonal
  - Mirroring: reflect the whole picture or part of it along one of the edges
Transformation = Small changes in copying

- Making relatively small changes in the basic copying algorithm can make a variety of transformations.
  - Change the targetX and targetY initial values, and you copy wherever you want
  - **Cropping**: Change the sourceX and sourceY range, and you copy only part of the picture.
  - **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.
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 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.

- `int()` is a function that takes a number as input and it returns the integer part of the number, throwing away anything after the decimal point

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