Image Processing in Python

- Slides at:
  
We will program in JES

- JES: Jython Environment for Students
- A simple *editor* (for entering in our *programs* or *recipes*): the *program area*
- A *command* area for entering in commands for Python to execute.
JES - Jython Environment for Students

Editor or Program Area

```python
def pickAndShow():
    myFile = pickAFile();
    myPict = makePicture(myFile);
    show(myPict);
```

Command Area

```text
>>> 3 + 4
7
>>> 1 / 3
0
>>> 2 * 6
12
>>> print "Hi"
Hi
```
Python understands *commands*

- We can name data with `=`
- We can print values, expressions, anything with `print`
Using JES

>>> print 34 + 56
90

>>> print 34.1/46.5
0.7333333333333334

>>> print 22 * 33
726

>>> print 14 - 15
-1

>>> print "Hello"
Hello

>>> print "Hello" + "Mark"
HelloMark
Command Area Editing

- Up/down arrows walk through *command history*
- You can edit the line at the bottom
  - and then hit Return/Enter
  - that makes that last line execute
Demonstrating JES for files and sounds

```python
>>> print pickAFile()
/Users/guzdial/mediasources/barbara.jpg
>>> print makePicture(pickAFile())
Picture, filename /Users/guzdial/mediasources/barbara.jpg height 294 width 222
>>> print pickAFile()
/Users/guzdial/mediasources/hello.wav
>>> print makeSound(pickAFile())
Sound of length 54757
>>> print play(makeSound(pickAFile()))
None
>>> myfilename = pickAFile()
>>> print myfilename
/Users/guzdial/mediasources/barbara.jpg
>>> mypicture = makePicture(myfilename)
>>> print mypicture
Picture, filename /Users/guzdial/mediasources/barbara.jpg height 294 width 222
>>> show(mypicture)
```
Writing a recipe:

- To make a function, use the command `def`.
- Then, the name of the function, and the names of the input values between parentheses (“(input1)”)
- End the line with a colon (“:”)
- The body of the recipe is indented (Hint: Use three spaces)
  - That’s called a block
Making functions the easy way

- Get something working by typing commands in the command window (bottom half of JES)
- Enter the `def` command in the editing window (top part of JES)
- Copy-paste the right commands up into the recipe
Image Processing

Goals:

- Give you the basic understanding of image processing, including psychophysics of sight,
- Identify some interesting examples to use
Light Perception

- Color is continuous
  - Visible light is in the wavelengths between 370 and 730 nanometers
    - That’s 0.00000037 and 0.00000073 meters
- But we *perceive* light with color sensors that peak around 425 nm (blue), 550 nm (green), and 560 nm (red).
  - Our brain figures out which color is which by figuring out how much of each kind of sensor is responding
  - One implication: We perceive two kinds of “orange” — one that’s *spectral* and one that’s red+yellow (hits our color sensors just right)
  - Dogs and other simpler animals have only two kinds of sensors
    - They do see color. Just *less* color.
Luminance vs. Color

- We perceive borders of things, motion, depth via **luminance**
  - Luminance is *not* the amount of light, but our *perception* of the amount of light.
  - We see blue as “darker” than red, even if same amount of light.
- Much of our luminance perception is based on comparison to backgrounds, not raw values.

Luminance is actually *color blind*. Completely different part of the brain.
Digitizing pictures

- We digitize pictures into lots of little dots
- Enough dots and it looks like a continuous whole to our eye
  - Our eye has limited resolution
  - Our background/depth *acuity* is particularly low
- Each picture element is referred to as a *pixel*
- Pixels are *picture elements*
  - Each pixel object knows its *color*
  - It also knows where it is in its *picture*
Encoding color

- Each pixel encodes color at that position in the picture
- Lots of encodings for color
  - Printers use CMYK: Cyan, Magenta, Yellow, and black.
  - Others use HSB for Hue, Saturation, and Brightness (also called HSV for Hue, Saturation, and Value)
- We’ll use the most common for computers
  - RGB: Red, Green, Blue
Encoding Color: RGB

- In RGB, each color has three component colors:
  - Amount of redness
  - Amount of greenness
  - Amount of blueness
- Each does appear as a separate dot on most devices, but our eye blends them.
- In most computer-based models of RGB, a single byte (8 bits) is used for each
  - So a complete RGB color is 24 bits, 8 bits of each
Each component color (red, green, and blue) is encoded as a single byte.

Colors go from \((0,0,0)\) to \((255,255,255)\):
- If all three components are the same, the color is in greyscale:
  - \((50,50,50)\) at \((2,2)\)
  - \((0,0,0)\) (at position \((1,2)\) in example) is black
  - \((255,255,255)\) is white
Basic Picture Functions

- `makePicture(filename)` creates and returns a picture object, from the JPEG file at the filename
- `show(picture)` displays a picture in a window
- We’ll learn functions for manipulating pictures later, like `getColor`, `setColor`, and `repaint`
Writing a recipe: Making our own functions

- To make a function, use the command `def`
- Then, the name of the function, and the names of the input values between parentheses (“(input1)”)
- End the line with a colon (“:”)
- The **body** of the recipe is indented (Hint: Use two spaces)
- Your function does **NOT** exist for JES until you **load** it
Use a loop!
Our first picture recipe

def decreaseRed(picture):
    for p in getPixels(picture):
        value = getRed(p)
        setRed(p, value * 0.5)

Used like this:
>>> file = "Users/guzdial/mediasources/barbara.jpg"
>>> picture = makePicture(file)
>>> show(picture)
>>> decreaseRed(picture)
>>> repaint(picture)
Use a loop!

Our first picture recipe

```python
def decreaseRed(picture):
    for p in getPixels(picture):
        value = getRed(p)
        setRed(p, value * 0.5)
```

Used like this:
```python
>>> file = "'/Users/guzdial/mediasources/katie.jpg"
>>> picture = makePicture(file)
>>> show(picture)
>>> decreaseRed(picture)
>>> repaint(picture)
```
def decreaseRed(pict):
    for p in getPixels(pict):
        value = getRed(p)
        setRed(p, value * 0.5)

• Recipe: To decrease the red
• Ingredients: One picture, name it pict
• Step 1: Get all the pixels of pict. For each pixel p in the pixels...
• Step 2: Get the value of the red of pixel p, and set it to 50% of its original value
How does increaseRed differ from decreaseRed?

Well, it does increase rather than decrease red, but other than that...

- It takes the same parameter input
- It can also work for any picture
  - It’s a specification of a process that it will work for any picture
  - There’s nothing specific to any particular picture here.

Practical programs = parameterized processes
def clearBlue(picture):
    for p in getPixels(picture):
        setBlue(p, 0)

Again, this will work for any picture.

Try stepping through this one yourself!
Can we combine these? Why not!

- How do we turn this beach scene into a sunset?
- What happens at sunset?
  - At first, I tried increasing the red, but that made things like red specks in the sand REALLY prominent.
    - That can’t be how it really works
  - New Theory: As the sun sets, less blue and green is visible, which makes things look more red.
A Sunset-generation Function

def makeSunset(picture):
    for p in getPixels(picture):
        value = getBlue(p)
        setBlue(p, value * 0.7)
        value = getGreen(p)
        setGreen(p, value * 0.7)
Creating a negative

- Let’s think it through
  - R, G, B go from 0 to 255
  - Let’s say Red is 10. That’s very light red.
    - What’s the opposite? LOTS of Red!
    - The negative of that would be 245: 255 - 10
  - So, for each pixel, if we negate each color component in creating a new color, we negate the whole picture.
def negative(picture):
    for px in getPixels(picture):
        red = getRed(px)
        green = getGreen(px)
        blue = getBlue(px)
        negColor = makeColor( 255-red, 255-green, 255-blue)
        setColor(px, negColor)
Original, negative, double negative

(This gives us a quick way to test our function: Call it twice and see if the result is equivalent to the original)
Converting to grayscale

- We know that if red = green = blue, we get gray
  - But what value do we set all three to?
- What we need is a value representing the darkness of the color, the *luminance*
- There are many ways, but one way that works reasonably well is dirt simple—simply take the average:

\[
\frac{\text{red} + \text{green} + \text{blue}}{3}
\]
Converting to grayscale

```python
def grayscale(picture):
    for p in getPixels(picture):
        sum = getRed(p) + getGreen(p) + getBlue(p)
        intensity = sum / 3
        setColor(p, makeColor(intensity, intensity, intensity))
```

You’ll play with variations of this at the “play” period.
Building a better grayscale

- We’ll *weigh* red, green, and blue based on how light we perceive them to be, based on laboratory experiments.

```python
def grayScaleNew(picture):
    for px in getPixels(picture):
        newRed = getRed(px) * 0.299
        newGreen = getGreen(px) * 0.587
        newBlue = getBlue(px) * 0.114
        luminance = newRed + newGreen + newBlue
        setColor(px, makeColor(luminance, luminance, luminance, luminance))
```
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 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 * 1.1)
```
Replacing colors in a range

```python
def turnRedInRange():
    brown = makeColor(57,16,8)
    file=r"C:\Documents and Settings\Mark Guzdial\My Documents\mediasources\barbara.jpg"
    picture=makePicture(file)
    for x in range(70,168):
        for y in range(56,190):
            px=getPixel(picture,x,y)
            color = getColor(px)
            if distance(color,brown)<50.0:
                redness=getRed(px)*1.5
                setRed(px,redness)
    show(picture)
    return(picture)
```

Get the range using MediaTools
Removing “Red Eye”

- When the flash of the camera catches the eye just right (especially with light colored eyes), we get bounce back from the back of the retina.
- This results in “red eye”
- We can replace the “red” with a color of our choosing.
- First, we figure out where the eyes are (x,y) using MediaTools
Removing Red Eye

def removeRedEye(pic, startX, startY, endX, endY, replacementColor):
    red = makeColor(255, 0, 0)
    for x in range(startX, endX):
        for y in range(startY, endY):
            currentPixel = getPixel(pic, x, y)
            if (distance(red, getColor(currentPixel)) < 165):
                setColor(currentPixel, replacementColor)

What we’re doing here:

• Within the rectangle of pixels (startX, startY) to (endX, endY)
• Find pixels close to red, then replace them with a new color

Why use getPixels? Because we don’t want to replace her red dress!
“Fixing” it: Changing red to black

`removeRedEye(jenny, 109, 91, 202, 107, makeColor(0,0,0))`

- Jenny’s eyes are actually not black—could fix that
- Eye are also not mono-color
  - A better function would handle *gradations* of red and replace with *gradations* of the right eye color
If you know where the pixels are: 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
Mirroring a picture: Two ways

- Slicing a picture down the middle and sticking a mirror on the slice
- One Approach: Do it by using a loop to measure a *difference*
  - The index variable is actually measuring distance from the mirror point
  - Then reference to either side of the mirror point using the difference
Mirroring a Picture

- Second Approach: Work out the formula for rightside x given leftside x.
  - Answer: width-x-1
Recipe for mirroring

def mirrorVertical(source):
    mirrorpoint = getWidth(source)/2
    width = getWidth(source)
    for y in range(0,getHeight(source)):
        for xOffset in range(0,mirrorpoint):
            leftPixel = getPixel(source, x, y)
            rightPixel = getPixel(source, width-x-1,y)
            c = getColor(leftPixel)
            setColor(rightPixel,c)
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)
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 that I took on 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
Program to mirror the temple

def mirrorTemple():
    source = makePicture(getMediaPath("temple.jpg"))
    mirrorpoint = 276
    lengthToCopy = mirrorpoint - 13
    for x in range(1,lengthToCopy):
        for y in range(27,97):
            p = getPixel(source,mirrorpoint-x,y)
            p2 = getPixel(source,mirrorpoint+x,y)
            setColor(p2,getColor(p))
    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?
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
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
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(getWidth(barb)):
        targetY = 100
        for sourceY in range(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):
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(getWidth(flower1)):
        targetY = getHeight(canvas) - getHeight(flower1) - 5
        for sourceY in range(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(getWidth(flower2)):
        targetY = getHeight(canvas) - getHeight(flower2) - 5
        for sourceY in range(getHeight(flower2)):
            px = getPixel(flower2, sourceX, sourceY)
            cx = getPixel(canvas, targetX, targetY)
            setColor(cx, getColor(px))
            targetY = targetY + 1
            targetX = targetX + 1
    # Third picture, flower1 negated
    negative(flower1)
    targetX = 200
    for sourceX in range(getWidth(flower1)):
        targetY = getHeight(canvas) - getHeight(flower1) - 5
        for sourceY in range(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(getWidth(flower2)):
        targetY = getHeight(canvas) - getHeight(flower2) - 5
        for sourceY in range(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(getWidth(flower1)):
        targetY = getHeight(canvas) - getHeight(flower1) - 5
        for sourceY in range(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)
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
Again, swapping the loop works fine

```python
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
```

We can use `targetX` and `targetY` as the `for` loop index variables, and everything works the same.
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.

```python
>>> print int(1)
1
>>> print int(1.5)
1
>>> print int(2)
2
>>> print int(2.5)
2
```
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:
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...

![Diagram showing scaling up process with source and canvas areas, sourceX=0, sourceY=1.5, targetX=3, targetY=4, actual X=0, actual Y=1]
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
def playWithCopy(inPic):
    canvas = makeEmptyPicture(1024,768)
    srcX = 0
    targetX = 0
    while (srcX < getWidth(inPic)):
        srcY = 0
        targetY = 0
        while (srcY < getHeight(inPic)):
            srcPixel = getPixel(inPic,srcX,srcY)
            targetPixel = getPixel(canvas,targetX,targetY)
            setColor(targetPixel,getColor(srcPixel))
            srcY = srcY + 1
            targetY = targetY + 1
        srcX = srcX + 1
        targetX = targetX + 1
    show(canvas)
    return(canvas)
Time to Play!

1. Posterize.py has two kinds of posterize functions. Try them both, and play with the constants.
2. Grayscale uses the red channel as the luminance value. Try red, green, blue, and the average. What's the difference?
3. playWithCopy(inputPicture) returns canvas
   - What happens if you increment src by 2 while increment target by 1?
   - Vice versa?
   - What if you swap X and Y in setting the target?
   - Change starting constants