CS 1124 Media Computation Lecture 2.2

Steve Harrison

September 3, 2008

Much of programming is about naming

- We name our data
  - *Data*: The "numbers" we manipulate
  - We call our names for data *variables*
- We name our recipes
- Quality of names determined much as in Philosophy or Math
  - Enough words to describe what you need to describe
  - Understandable

Our programs work with a variety of *names*

- You will name your *functions*
  - **Just like functions you knew in math, like sine and gcd (Greatest Common Divisor)**
- You will name your *data* (*variables*)
- You will name the data that your functions work on
  - **Inputs, like the 90 in sine(90)**
- Key: Names inside a function only have meaning while the function is being executed by the computer. (You'll see what we mean.)

Types: Naming our Encodings

- We even name our encodings
  - **Sometimes referred to as types**
- Some programming languages are *strongly typed*
  - **A name has to be declared to have a type, before any data is associated with it**
  - **Python is NOT strongly typed**
- Some *types* are:
  - **integers**
  - **text**
  - **real numbers**
  - **pixels**

Names for things that are not in memory

- A common name that you'll deal with is a *file name*
  - **The program that deals with those is called the operating system, like Windows, MacOS, Linux**
- A file is a collection of bytes, with a name, that resides on some external medium, like a *hard disk*.
  - **Think of it as a whole bunch of space where you can put your bytes**
- Files are typed, typically with three letter *extensions*
  - **.jpg files are JPEG (pictures), .wav are WAV (sounds)**

Names can be (nearly) whatever we want

- Must start with a letter (but can *contain* numerals)
- Can't contain spaces
  - **myPicture is okay but my Picture is not**
- Be careful not to use command names as your own names
  - **print = 1 won't work**
  - **(Avoid names that appear in the editor pane of JES highlighted in blue or purple)**
- *Case matters*
  - **MyPicture is not the same as myPicture or mypicture**
- Sensible names are sensible
  - **E.g. myPicture is a good name for a picture, but not for a picture file.**
  - **x could be a good name for an x-coordinate in a picture, but probably not for anything else**

JES Functions

- A bunch of functions are pre-defined in JES for sound and picture manipulations
  - **pickAFile()**
  - **makePicture()**
  - **makeSound()**
  - **show()**
  - **play()**
- Some of these functions accept *input* values

```
theFile = pickAFile()
pic = makePicture(theFile)
```

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**

Sound Functions

- **makeSound**(filename) creates and returns a sound object, from the WAV file at the filename
- **play**(sound) makes the sound play
  - **but doesn't wait until it's done**
  - **blockingPlay(sound) waits for the sound to finish**
- We'll learn more later like **getSample** and **setSample**

COMPLETELY THE SAME:
Values, names for those values, functions that return those values

```
>>> file=pickAFile()
>>> print file
C:\Documents and Settings\Mark Guzdial\My Documents
   \mediasources\barbara.jpg
>>> show(makePicture(file))
>>> show(makePicture(r"C:\Documents and Settings\Mark
   Guzdial\My Documents\mediasources\barbara.jpg"))
>>> show(makePicture(pickAFile()))
```

Put r in front of Windows filenames: r"C:\mediasources\pic.jpg"

Names are “scoped”

```
def pickAndShow():
  myfile = pickAFile()
  mypicture = makePicture(myfile)
  show(mypicture)
```

Names are "scoped"

```
def pickAndShow():
  myfile = pickAFile()
  mypicture = makePicture(myfile)
  show(mypicture)
```

Bug alert!!!

myfile and **mysound**, inside pickAndPlay(), are *completely different* from the same names in the command area.

Side effects vs. returning a result

```
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)
```

Side effects vs. returning a result

```
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)
```

- What would happen if we typed ?
  - `>>> show(negative(picture))`

Side effects vs. returning a result

```
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)
```

- What would happen if we typed ?
  - **>>> show(negative(picture))**
- What would we need to do make it show the negative picture?
  - **return the thing we want to be the result**
  - **in this case the result should be the parameter called “picture**

Side effects vs. returning a result

```
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)
```

- What would happen if we typed ?
  - **>>> show(negative(picture))**
- What would we need to do make it show the negative picture?
  - **return the thing we want to be the result**
  - **in this case the result should be the parameter called “picture**
- Lets write that....

Side effect

```
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)
```

```
>>> myPicture = makePicture(file)
>>> negative(myPicture)
>>> show(myPicture)
```

Returning a result

```
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)
```

```
>>> myPicture = makePicture(file)
>>> negPicture = negative(myPicture)
>>> show(negPicture)
```

Returning a result

```
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)

  return (picture)
```

```
>>> myPicture = makePicture(file)
>>> negPicture = negative(myPicture)
>>> show(negPicture)
```

Returning from a function

- At the end, we **return** the picture
- Why are we using **return**?
  - **If we didn't return it, we couldn't get at it in the command area**
- So we can't give the results of a function a name unless we **return** it.
  - **We use the returned value by giving it a name, that is by assigning it to a variable. For example, negPicture = negative(myPicture)**

```
negColor = makeColor( 255-red, 255-green, 255-blue)
setColor(px, negColor)

return(picture)
```

Returning a result (more)

```
def negative(file):
  picture = makePicture(file)
  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)
```

```
>>> negPicture = negative(file)
>>> show(negPicture)
```

Returning a result (more)

```
def negative(file):
   picture = makePicture(file)
   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)
    return (picture)
```

```
>>> negPicture = negative(file)
>>> show(negPicture)
```

Returning a result (more)

- In this case we must use **return**
- Why?
  - **Because we created "picture" inside the function and variable names are scoped -- they only work INSIDE the function**

```
setColor(px, negColor)

return(picture)
```

Lets do a simple name-space exercise

- Thanks for volunteering ....

Lets do a simple name-space exercise

```
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)
```

```
>>> file = pickAFile()
>>> picture = makePicture(file)
>>> negative(picture)
>>> show(picture)
```

Lets do a simple name-space exercise

```
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)
```

```
>>> file = pickAFile()
>>> myPicture = makePicture(file)
>>> negPicture = negative(myPicture)
>>> show(negPicture)
```

Lets do a simple name-space exercise

```
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)

    return (picture)
```

```
>>> file = pickAFile()
>>> myPicture = makePicture(file)
>>> negPicture = negative(myPicture)
>>> show(negPicture)
```

Lets do a simple name-space exercise

- Thanks for volunteering ....

"Hard-coding" for a specific sound or picture

```
def playSound():
  myfile = "FILENAME"
  mysound = makeSound(myfile)
  play(mysound)

def showPicture():
  myfile = "FILENAME"
  mypict = makePicture(myfile)
  show(mypict)
```

You can always replace data (a *string* of characters, a number…. whatever) with a name (*variable*) that holds that data

…. or vice versa.

Q: This works, but can you see its disadvantage?

Functions with inputs are more general-purpose

```
def playNamed(myfile):
  mysound = makeSound(myfile)
  play(mysound)


def showNamed(myfile):
  mypict = makePicture(myfile)
  show(mypict)
```

Q: What functions do you need?

Q: What (if any) should be their input(s)?

A: In general, have enough functions to do what you want, easily, understandably, and in the fewest commands

(i.e. by using more generic, less specific functions)

But these are *only* questions of style

What can go wrong?

- Did you use the *exact* same names (case, spelling)?
- All the lines in the block must be *indented*, and *indented the **same** amount.*
- Variables in the command area don't exist in your functions, and variables in your functions don't exist in the command area.
- The computer can't read your mind.
  - **It will only do exactly what you tell it to do.**

Programming is a craft

- You don't learn to write, paint, or knit by attending knitting lectures and watching others knit.
  - **You learn to knit by doing it.**
- Programming is much the same.
  - **You have to try it, make many mistakes, learn how to control the computer, learn how to think in Python.**
- *The HW and group project programs that you have to write in this class aren't enough!*
  - **Do programming on your own!**

Review: Converting to grayscale

$$\frac{(red+green+blue)}{3}$$

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

Review: 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:

$$\frac{(red+green+blue)}{3}$$

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

Why can't we get back again?

- Converting to grayscale is different from computing a negative.
  - **A negative transformation retains information.**
- With grayscale, we've lost information
  - **We no longer know what the ratios are between the reds, the greens, and the blues**
  - **We no longer know any particular value.**

Media compressions are one kind of transformation.
Some are **lossless** (like negative);
Others are **lossy** (like grayscale)

Why can't we get back again?

- Converting to grayscale is different from computing a negative.
  - **A negative transformation retains information.**
- *With grayscale, we've lost information*
  - **We no longer know what the ratios are between the reds, the greens, and the blues**
  - **We no longer know any particular value.**

Media compressions are one kind of transformation.
Some are **lossless** (like negative);
Others are **lossy** (like grayscale)

But that's not really the best grayscale

- In reality, we don't perceive red, green, and blue as *equal* in their amount of luminance: How bright (or non-bright) something is.
  - **We tend to see blue as "darker" and red as "brighter"**
  - **Even if, physically, the same amount of light is coming off of each**
- Photoshop's grayscale is very nice: Very similar to the way that our eye sees it
  - **B&W TV's are also pretty good**

Building a better grayscale

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

```
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))
```

Let's try making Barbara a redhead!

- We could just try increasing the redness, but as we've seen, that has problems.
  - **Overriding some red spots**
  - **And that's more than just her hair**
- If only we could increase the redness *only* of the brown areas of Barb's head…

…./MediaSources/barbara.jpg

Treating pixels differently

- We can use the **if** statement to treat some pixels differently.
- For example, color replacement: Turning Barbara into a redhead
  - **Use the MediaTools to find the RGB values for the brown of Barbara's hair**
  - **Then look for pixels that are close to that color (within a threshold), and increase by 50% the redness in those**

Find the RGB of the brown color

How “close” are two colors?

- Sometimes you need to find the *distance* between two colors, e.g., when deciding if something is a “close enough” match
- How do we measure distance?
  - **Pretend it’s Cartesian coordinate system**
  - **Distance between two points:**

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Distance between two colors:
  - **This is a case where the figure of speech “distance between colors” actually is a mathematical function!**

$$\sqrt{(red_1 - red_2)^2 + (green_1 - green_2)^2 + (blue_1 - blue_2)^2}$$

distance(color1, color2)

- It does the distance calculation:

$$\sqrt{(red_1 - red_2)^2 + (green_1 - green_2)^2 + (blue_1 - blue_2)^2}$$

```
def distance( color1, color2 ):
    redDiff = getRed(color1) - getRed(color2)
    greenDiff = getGreen(color1) - getGreen(color2)
    blueDiff = getBlue(color1) - getBlue( color2)
    colorDistance = sqrt((redDiff*redDiff)+(greenDiff*greenDiff)+(blueDiff*blueDiff))
    return colorDistance
```

Making Barb a redhead

```
def turnRed(file ):
  brown = makeColor(57, 16, 8)
  picture = makePicture(file)
  for px in getPixels(picture):
     color = getColor(px)
     if distance(color, brown) < 50.0:
        redness = getRed(px) * 1.5
        setRed(px, redness)
  show(picture)
  return(picture)
```

Original:

Digital makeover:

Talking through the program slowly

- The brown is the brownness that figured out from MediaTools
- The file is where the picture of Barbara is on the computer
- We need the picture to work with

```
def turnRed( file ):
  brown = makeColor(57, 16, 8)
  picture = makePicture(file)
  for px in getPixels(picture):
    color = getColor(px)
    if distance(color, brown) < 50.0:
      redness = getRed(px) * 1.5
      setRed(px, redness)
  show(picture)
  return(picture)
```

Walking through the for loop

- Now, for each pixel **px** in the picture, we
  - Get the color
  - See if it's within a distance of 50 from the brown we want to make more red
  - If so, increase the redness by 50%

```
def turnRed( file ):
 brown = makeColor(57, 16, 8)
 picture = makePicture(file)
 for px in getPixels(picture):
  color = getColor(px)
  if distance(color, brown) < 50.0:
    redness=getRed(px) * 1.5
   setRed(px, redness)
 show(picture)
 return(picture)
```

How an if works

- **if** is the command name
- Next comes an expression: Some kind of true or false comparison
- Then a colon
- Then the body of the if—the things that will happen if the expression is true is a **block**

```
if distance(color, brown) < 50.0:
    redness = getRed(px)*1.5
    blueness = getBlue(px)
    greenness = getGreen(px)
```

Expressions

Bug alert!

= means "assign the results to this variable" (and does NOT work with "if")

== means "are they equal?"

- Can test equality with ==
- Can also test <, >, >=, <=, <> (not equals)
- In general, 0 is false, 1 is true
  - **So you can have a function return a "true" or "false" value.**

Returning from a function

- At the end, we **show** and **return** the picture
- Why are we using **return**?
  - **Because the picture is created within the function**
  - **If we didn't return it, we couldn't get at it in the command area**
- We could **print** the result, but we'd more likely assign it a name

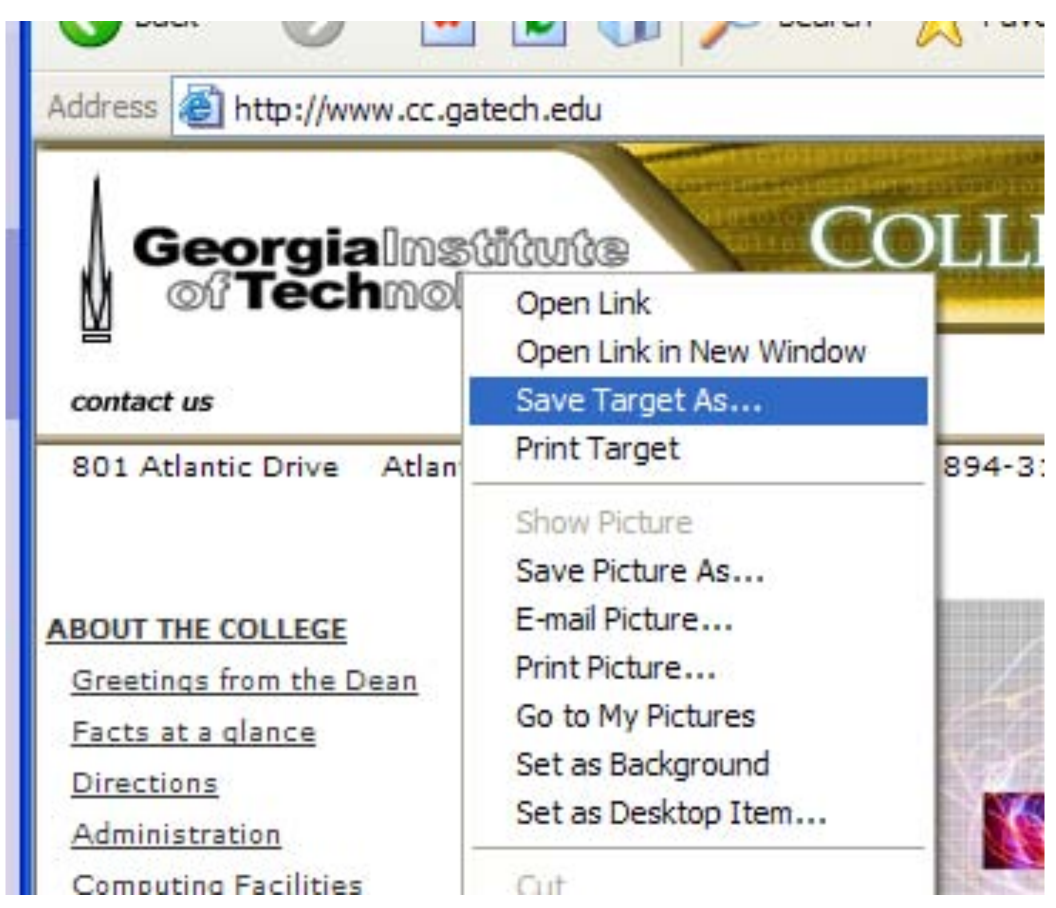
```
if distance(color, brown) < 50.0:
    redness = getRed(px) * 1.5
    setRed(px, redness)
show(picture)
return(picture)
```

Things to change

- Lower the threshold to get more pixels
  - **But if it's too low, you start messing with the wood behind her**
- Increase the amount of redness
  - **But if you go too high, you can go beyond the range of valid color intensities (i.e. more than 255)**

Grabbing media from the Web

- Right-click (Windows) or Control-Click (Mac)
- Save Target As…
- Can *only* do JPEG images (.jpe, .jpg, .jpeg)



Most images on the Internet are copyright. You can download and use them for your use *only* without permission.

Lots of ideas today ....

- Names
- return from a function
- Side-effects
- Programming as craft
- Grayscale
  - **why it looses information**
  - **better looking grayscale**
- color “distance”
- the “if” statement

Questions?

Project 2

- Specification - FIVE variations of Lane Stadium:
  - **reduce red by 50%**
  - **reduce blue by 40%**
  - **reduce green by 30%**
  - **makeSunset (page 62)**
  - **posterize(page 105)**

- Lagniappe ("A Little Bit Extra") - variation 6
  - **do any of the above for 1/2 of the picture. There are many ways to define "1/2 of the picture". (Think about it…)**
  - **# tell us what you did so we will know!**
- Details on moodle

chci Center for Human Computer Interaction
Virginia Polytechnic Institute and State University

On September 5, 2008

OPEN HOUSE!

CENTER FOR HUMAN COMPUTER INTERACTION

- **Come meet our CHCI faculty and students.**
- **See demonstrations of ongoing projects and find out how you can participate.**
- **Come to view our resources: labs, equipment.**
- **Join us for refreshments, information and FUN!**
  - **Opening welcome at 4pm in #1110 KW II**
  - **Research Demonstrations**
  - **Refreshments at 5pm**

Coming Attractions

- For Friday
  - **Project 1 due @ 2:00**
  - **start on Project 2**
  - **shortened lab**
  - **HCI Center Open House @ 4:00 PM**
- For Monday
  - **Read Chapter 4 (through at least 4.3)**
  - **Do Quiz 4 (due 10:00 am)**
- Next Friday
  - **Project 2 Due**