Introduction to Programming with JES

Titus Winters & Josef Spjut

October 6, 2005

1 Introduction

First off, welcome to UCR, and congratulations on becoming a Computer Engineering major. Excellent choice.

One of the major skills you'll be learning as a Computer Engineer is the art of programming. Since this is very likely to be a skill that you haven't been exposed to before, we'd like to take this opportunity to give you a head start. Tonight, we are hoping to present you with most of the high-level concepts from an introductory programming course all in the space of 3 hours. Sound difficult? It won't be. If anything, this should be a lot of fun.

1.1 JES

Tonight we are using JES, a cutting-edge tool for teaching introductory programming, and we are going to use it to do some fun stuff with graphics. JES[1] is a free implementation of the Python[2] programming language, and also includes some nice tools for playing with media files like graphics and sounds. We are only going to focus on graphics tonight.

To get started **figure out how to open a terminal in the default config** and type *jes*, and then hit Enter. It may come up with a box asking for your registration information, you should just hit *Cancel* if that happens.

JES has two main ways of writing code (we call instructions for the computer "source code" or just "code" for short). If you are just trying stuff out, trying to see what things do, you can use the interactive window. The interactive window is the black box at the bottom of the JES window.

If you are writing code that you want to keep, like the code you'll turn in for the Freshman Programming Contest, then you want to write it in the upper window. The upper window lets you store your code to a file, or load it from a file later on. A good rule of thumb is that if you are writing a function (which we'll get to in Section 5), you should write it in this upper window. Otherwise, just use the interactive window.

The rest of this handout will be organized into sections. Each section will have a few simple examples of new commands and tools, followed by an activity for you to complete. If you finish before everybody else, go see who needs help. You'll learn a **lot** by helping others, and you may make some new friends.

[1] http://coweb.cc.gatech.edu/mediaComp-plan/94

[2] http://www.python.org

2 Variables & Arithmetic

Traditionally, the first program that you should write in any language is what we call the "Hello World" program, a simple program that prints out the friendly message "Hello World!" In the Python programming language[3] this is very easy. Type in the interactive window (the bottom window)[4]:

```
print "Hello World!"
```

When you hit Enter, JES goes off to interpret what commands you gave it. In this case, it interprets your command as an instruction to print[5] the message in quotes. In Python, a bit of text in quotes is called a "string". You can do things with strings other than just print them. For example, in Python you can add two strings together (concatenation) using the "+" sign. Look at this and guess what it does before you run it:

```
print "Hello" + "World!"
```

Did you guess right? Exactly right? Notice that there is a little bit of uglyness here. Fix it before moving on.

All better now? Good. Moving right along. What if we want to put together two strings and save the results for later? Instead of printing the results to the screen, we can just *assign* the result of mashing those strings together to a *variable*, like so:

```
x = "Hello " + "World!"
print x
```

Of course, if all we have to work with is text, we're gonna be pretty limited. Python also supports numeric calculations:

```
print 5 * 10
print 3 / 2
print 3.0 / 2
print 10 + 5
print 32 % 3
print 5 % 2
print 3 - 1
total = 5 + 3
print total
```

So we can do all sorts of math. **N**OTE: notice that Python rounds down if you divide whole numbers, but does exact division if you give it at least one number with a decimal.[6] Can you tell what the % operator does? (It is related to division.)

[3] Remember, JES is an implementation of Python, kinda like you are a speaker of English

[4] For any code written in like the following, we really encourage you to type it in and make it work before continuing on.

[5] To the screen, not to the printer

[6] Decimal numbers are called *floating point* numbers.

Variables are also useful as place holders. Can you guess what the following code does before you type it in?

```
total = 5 + 3
total = total * 4
print total
```

2.1 Task

1. Write up a series of at least 10 steps, in English, of simple arithmetic. For example, "Start with 4, multiply by 10, add 7, subtract 3, divide by 11, etc."
2. What is the correct answer for your steps? Write it down and keep it to yourself.
3. Find a partner
4. Trade steps with your partner.
5. Code up your steps in JES
6. Do your answers match? Figure out who was right: your code or your partner's math.

3 Images & Colors

OK, so you're good with arithmetic and variables now. Let's move on to something more interesting: pictures.

First: how do we make a new picture? In JES you can use the function *makePicture*.

```
myPicture = makePicture("/home/csgrads/titus/jesPic01.jpg")
show(myPicture)
```

- The *makePicture* function takes in a string that is the path to a picture, and returns a new picture. You can then store that picture in a variable, just like storing a string or storing the result of some arithmetic.
- The *show* function takes a picture and displays it on the screen.

Since you may not always know the path to your pictures, you can also use the *pickAFile* function to make a new window appear to let the user choose a file:

```
fileName = pickAFile()
myPicture = makePicture(fileName)
show(myPicture)
```

- The *pickAFile* function takes no parameters[7] (but you do have to give it the ()'s!) and returns a string. Since it returns a string and makePicture is expecting a string, we can actually make this even easier:

```
myPicture = makePicture(pickAFile())
show(myPicture)
```

Also, if you don't want to have a picture from a file, you can just use *makeEmptyPicture*[8].

```
# Creates an empty picture 200 pixels wide by 100 pixels high
myEmptyPicture = makeEmptyPicture(200, 100)
```

- The *makeEmptyPicture* function takes two parameters: the width and height of the new picture to make. It returns a picture.

3.1 Task

1. Find a picture online using your web browser
2. Trade pictures with your partner either by giving them the address of the picture, or emailing it to them
3. Use *makePicture* to create a new picture variable from that picture and store it
4. Call *show* on that picture

[7] Parameters are what goes inside the parenthesis after a function. If there are more than one parameter, they are separated by commas.

[8] Also, any line in Python that starts with a # will be ignored. These are called comments, and are used to communicate with a human reader rather than the computer

4 Drawing on Images

So we can display images, which is great, but how about actually manipulating them? Well, lets start by making a red dot on a blank picture.

```
# Make a new picture
newPic = makeEmptyPicture(200, 100)

# Get a pixel from the middle of it
pixel = getPixel(newPic, 100, 50)

# Make that pixel red
setColor(pixel, red)

# Display the result
show(newPic)
```

This is nice, but drawing a single pixel at a time is gonna be a little tedious. We can also draw lines, rectangles, ovals, and text.

```
# Make a new picture
newPic = makeEmptyPicture(200, 100)

# Make a white line running from upper-left to lower-right
newPic.addLine(white, 0, 0, 200, 100)

# Make a blue 30x30 square whose upper-left corner is at 10, 20
newPic.addRect(blue, 10, 20, 30, 30)

# Make a red oval centered at 80, 80
newPic.addOval(red, 80, 80, 20, 10)

# Make a new color of mostly red and green, with a little blue
pukeColor = makeColor(196, 181, 84)

# Print a friendly message in that new color
newPic.addText(pukeColor, 30, 80, "Hello UCR!")

# Display the results
show(newPic)

# Save the results to a new file
writePictureTo(newPic, "test.jpg")
```

Quite a few new functions here:

- The function *addLine* takes a color and four coordinates x1, y1, x2, y2, and draws a straight line in that color. Note that you call this in a slightly different way than the other functions we've seen so far. Don't worry about the difference for now.
- The function *addRect* takes a color and four coordinates x1, y1, x2, y2, and draws a rectangle with (x1, y1) and (x2, y2) as two opposite corners.
- The function *addOval* takes a color and four integers x, y, width, height. It draws an oval whose left edge touches x, whose top edge touches y, and with the given width and height. To make a circle, just make width and height equal.
- The function *addText* takes a color, two coordinates x and y, and a string. It then writes that string starting at those coordinates in that color.
- The function *makeColor* takes three integers red, green, and blue, and returns a new color. This way you can make your own colors if you know their red, green, and blue intensities! (0, 0, 0) is black, (255, 255, 255) is white, and (255, 0, 0) is red. (Note: you can also pick a color using the *pickAColor()* function.)
- The function *writePictureTo* takes a picture and a filename and writes the picture to that file. Be sure to include the extension ".jpg" so that other programs can recognize the file type.

4.1 Task

1. Write your name in blue in the lower-left corner of the picture your partner chose in the previous section
2. Add a blue circle to the *center* of the picture. Think ahead: what do the x and y coordinates for *addOval* mean?
3. Save your picture as "task4.jpg"

5 Loops

What if we want to do something to every pixel in a picture? For example, the following code switches the red with the green, the green with the blue, and the blue with the red, in every pixel in a picture.

```
# This is how we make our own new commands (called "functions"), using
# the "def" keyword.  Now if we call colorSwap(newPic), it'll make
# "pic" another name for newPic, and do everything listed in this
# new function.

# NOTE: Indentation (spaces at the beginning of the line) are
# important here!
def colorSwap(pic):
  # Do the following to every pixel in the picture
  for pixel in getPixels(pic):
    # Get the color of the pixel
    color = pixel.getColor()

    # Get the red, green, and blue values of that color
    r, g, b = color.getRGB()

    # Make a new color that has swapped (r, g, b) for (g, b, r)
    newColor = makeColor(g, b, r)

    # Make the pixel the new color
    pixel.setColor(newColor)

newPic = makePicture(pickAFile())
colorSwap(newPic)
```

Type that in and test it out yourself. Do you see what it is doing?

5.1 Task

1. Load the image from the previous section as a new variable
2. Write a function that "inverts" the image: for each pixel, invert red, green, and blue. That is, if the red is 255, it goes to 0, and if it is 0 it goes to 255[9]. It will look a lot like the sample function.

[9]You should be able to calculate the new value for each component with a single subtraction: what is the formula that takes 0 to 255 and 255 to 0?

6 Conditionals

So far, we have only written things that will *always* do the same actions. What if we want to have a particular action happen only *some* of the time? For this, we use a *conditional*[10].

```
a = 5
if a == 5:
  print ‘‘A was 5!’’
else:
  print ‘‘A was not 5!’’
```

Mostly simple. But one important note: if you are asking if something is equal, you use ==, not just =. Remember that. So how about a graphical example:

```
# This makes any pixel that is mostly green into black
def makeGreenBlack(pic):
  # Same as before: for every pixel, do this
  for pixel in getPixels(pic):
    # Get the color of the pixel
    color = pixel.getColor()
    r, g, b = color.getRGB()

    # If there is more green than red and blue together, then make
    # it black
    if g > r + b:
      pixel.setColor(black)
```

6.1 Task

This is the hardest task of the night. If you can do this, you're a champion programmer (and if you finish, you just have to go help your classmates with your amazing skills!)

1. Pick a color: red, green, blue
2. Write a function that takes two parameters (pic and background). For each pixel, check if it is the color you picked. If so, find its x and y coordinates using *getX* and *getY*. Then set its color to the color of the pixel from *background* with the same coordinates.
3. This is very similar to early blue-screen techniques from major Hollywood movies!

[10]That is, there is a *condition*, and if that condition is true, then we do the thing

7 Freshman Programming Contest

Now that you've finished with our introduction, you're ready to start thinking about the Freshman Programming Contest. This year the contest is only open to you, the Computer Engineering majors. Rather than giving you a closed problem with no creativity behind it, the contest this year is actually an artistic one: you are to take all of the picture manipulation skills you've gained here and write a Python program to create a collage.

You'll want to hunt around online to find some source images, and then write a Python program to put them together artistically into one image and display that image. You can (and *should*) write some functions to manipulate those images before pasting them together. What you write is totally up to you. We'll be looking for two major things here: artistic merit, and technical merit.

7.1 Submission & Judging

We will email you instructions on submitting your collage programs a few days before the contest ends. After you submit them, we will examine the entries and make sure everything is in order (your file works, and your pictures all came through) the night of October 9th, and let you know if there are any problems. (So check your email before bed that night.) The next day we'll give your submissions to our judging panel, and then from 5-6 pm on October 10th we will all meet to demonstrate your submissions and see the winners.

More information will be sent out via email as the submission date draws nearer.

7.2 Hints & Suggestions

Here are some functions that you might want to implement when writing your collage program. Don't think of this as a complete list, this is just to get you started thinking about what is possible. If you come up with an idea that isn't here, try it out!

- Resize - Given an image, return a new image that is bigger or smaller by some amount. Easy to do with integers (like twice as big or half as big), a little harder with floats.
- Paste - Given a large image and a small one, copy the pixels from the smaller one into the larger one starting at some coordinate.
- Rotate - Given an image, return a new image that is rotated. Rotations by 90° increments are pretty easy, you'll need some trigonometry to do anything else.
- Alter color - You could write a collection of functions that take all the green out of an image (set the green part of each pixel to 0), or double the red, or cut the blue in half, etc.[11]

7.3 More Information and Help

If you have questions, *ask for help.* Your friends and classmates are one of your best resources, but they may be stumped too. If that happens, send email to *titus@cs.ucr.edu* and *sjosef@cs.ucr.edu* and we'll figure out your problem as quick as we can.

[11] If you are really clever you might even be able to write a single function that can replace any of the above.

If you are stumped on something, or just want to learn more, there are a lot of resources for Python available for free online. Here are some of the better ones.

1. Python tutorial - *http://www.python.org/doc/tut/* - The standard tutorial for learning Python. This won't teach you the graphical bits we've been talking about, but it will teach you more things like looping, doing math, using variables, etc.

2. JES Media Source - *http://www.cs.ucr.edu/ titus/media.txt* - The Python source for JES's media functions. There are quite a few things that we didn't introduce, you might be able to find some good stuff by poking around here.

3. Dive Into Python - *http://diveintopython.org* - A more advanced Python tutorial, primarily for people that have programmed in other languages before.