CSC104 Chapter 5 and Chapter 8 Lecture

Tim Capes

October 25, 2011

Course Work

- Finalized Assignment 2 is up.

Course Work

- Finalized Assignment 2 is up.
- Practice Midterm and solutions are posted.

Course Work

- Finalized Assignment 2 is up.
- Practice Midterm and solutions are posted.
- Midterm next week; Assignment Due in Two weeks

This weeks tutorial

- Sepia Tones

This weeks tutorial

- Sepia Tones
- Echoing a sound

Using if statements

- if condition : code block

Using if statements

- if condition : code block
- elif condition: code block

Using if statements

- if condition : code block
- elif condition: code block
- else code block

Using if statements

- if condition : code block
- elif condition: code block
- else code block
- example in lecture code

Targeted Colour Reduction

- Designing a colour replacement.

Targeted Colour Reduction

- Designing a colour replacement.
- Most typical example is red-eye removal which we will go through in detail

Reduce Red Eye: Line 1

- def removeRedEye(pic,startX,startY,endX,endY, replacementColor)

Reduce Red Eye: Line 1

- def removeRedEye(pic,startX,startY,endX,endY, replacementColor)
- 6 inputs: a picture, the box (4 inputs), and the color to replace with.

Reduce Red Eye: Line 1

- def removeRedEye(pic,startX,startY,endX,endY, replacementColor)
- 6 inputs: a picture, the box (4 inputs), and the color to replace with.
- Why do we apply the algorithm to only a small part of the picture?

Reduce Red Eye: Line 1

- def removeRedEye(pic,startX,startY,endX,endY, replacementColor)
- 6 inputs: a picture, the box (4 inputs), and the color to replace with.
- Why do we apply the algorithm to only a small part of the picture?
- Being close to red in an eye is a reasonable context for changing the red.
- Being close to red on clothing probably wouldn't work as well.

Reduce Red Eye: Line 2

- red = makeColor(255,0,0)

Reduce Red Eye: Line 2

- red = makeColor(255,0,0)
- We build a pure red colour. We will use this later for comparisons.

Reduce Red Eye: Line 3

- for x in range(startX,endX):

Reduce Red Eye: Line 3

- for x in range(startX,endX):
- Sets up our loop over the x-coordinate for values in the box.

Reduce Red Eye: Line 4

- for y in range(startY,endY):

Reduce Red Eye: Line 4

- for y in range(startY,endY):
- Sets up our loop over the y-coordinate for values in the box.

Reduce Red Eye: Line 5

- currentPixel = getPixel(pic,x,y)

Reduce Red Eye: Line 5

- currentPixel = getPixel(pic,x,y)
- Fetch the pixel at the current co-ordinates selected based on the loops.

Reduce Red Eye: Line 6 part 1

- distance(red,getColor(currentPixel))

Reduce Red Eye: Line 6 part 1

- distance(red,getColor(currentPixel))
- How far is our earlier defined red colour from the colour of our current pixel?

Reduce Red Eye: Line 6 part 1

- distance(red,getColor(currentPixel))
- How far is our earlier defined red colour from the colour of our current pixel?
- Distance returns that value and we have to decide what to do with it.

Reduce Red Eye: Line 6 part 2

- distance (red, getColor(currentPixel)) $<$ 165

Reduce Red Eye: Line 6 part 2

- distance (red, getColor(currentPixel)) < 165
- We will compare it to the number 165. Why 165? This comparison is either true or false and this makes up the conditional part of the if statement.

Reduce Red Eye: Line 6 part 2

- distance (red, getColor(currentPixel)) < 165
- We will compare it to the number 165. Why 165? This comparison is either true or false and this makes up the conditional part of the if statement.
- We find a value that works by trial and error. Too high and we'll select non-red colours. Too low and we won't get all the red.

Reduce Red Eye: Line 6 part 2

- distance (red, getColor(currentPixel)) < 165
- We will compare it to the number 165. Why 165? This comparison is either true or false and this makes up the conditional part of the if statement.
- We find a value that works by trial and error. Too high and we'll select non-red colours. Too low and we won't get all the red.
- Later we will run examples.

Reduce Red Eye: Line 6 part 3

- if distance(red,getColor(currentPixel))$<$ 165:

Reduce Red Eye: Line 6 part 3

- if distance(red,getColor(currentPixel))$<$ 165:
- We tell the computer when this condition holds we want to execute specific code and when it doesn't we don't.

Reduce Red Eye: Line 6 part 3

- if distance(red,getColor(currentPixel))$<$ 165:
- We tell the computer when this condition holds we want to execute specific code and when it doesn't we don't.
- What specific code? The block following the : determined by indentation

Reducing Red Eye: Inside the if statement, Line 7

- setColor(currentPixel, replacementColor)

Reducing Red Eye: Inside the if statement, Line 7

- setColor(currentPixel, replacementColor)
- We are inside the if block so we are close enough to red.

Reducing Red Eye: Inside the if statement, Line 7

- setColor(currentPixel, replacementColor)
- We are inside the if block so we are close enough to red.
- If the color is close enough to red use the replacement color.

The absence of a line 8

- There is no line 8

The absence of a line 8

- There is no line 8
- Why do we not need to return anything?

The absence of a line 8

- There is no line 8
- Why do we not need to return anything?
- This function works by side-effects; the original picture is changed.

Running the function

- Once we have a hypothesis about how it works.

Running the function

- Once we have a hypothesis about how it works.
- Run it to see if we are correct.

Running the function

- Once we have a hypothesis about how it works.
- Run it to see if we are correct.
- Also experiment and fit parameters (change values of distance)

Possible Variations 1

- How many inputs should this function really have?

Possible Variations 1

- How many inputs should this function really have?
- There is a lot of merit to adding one (which)?

Possible Variations 1

- How many inputs should this function really have?
- There is a lot of merit to adding one (which)?
- 165 should be a variable not a constant, allow the user to test red eye by repeatedly calling the function with different values.

Possible Variations 2

- Alternatively can use fewer inputs (5) (which 5)?

Possible Variations 2

- Alternatively can use fewer inputs (5) (which 5)?
- Hard-code the replacement color rather than expecting the user to provide it.

So why 6 inputs?

- If we think the user should specify parameters we should have 7; If not we should have 5.

So why 6 inputs?

- If we think the user should specify parameters we should have 7; If not we should have 5.
- So why do we have 6 is it reasonable design to want the user to specify only half of the parameters?

So why 6 inputs?

- If we think the user should specify parameters we should have 7; If not we should have 5.
- So why do we have 6 is it reasonable design to want the user to specify only half of the parameters?
- Perhaps. There is a case to be made that distance is something the program is responsible for, while replacementColor is something the user should specify.

So why 6 inputs?

- If we think the user should specify parameters we should have 7; If not we should have 5.
- So why do we have 6 is it reasonable design to want the user to specify only half of the parameters?
- Perhaps. There is a case to be made that distance is something the program is responsible for, while replacementColor is something the user should specify.
- Might have a function based on AI called calculateBestDistance(picture,startX,startY,endX,endY,red) to determine what value to use for changing the picture. Could call this function in the location where you need the distance.

Analyzing complicated blocks of code

1. Figure out how to break each line into individual pieces to understand.

Analyzing complicated blocks of code

1. Figure out how to break each line into individual pieces to understand.
2. Work from the smallest pieces outwards

Analyzing complicated blocks of code

1. Figure out how to break each line into individual pieces to understand.
2. Work from the smallest pieces outwards
3. Understand what the line is doing.

Analyzing complicated blocks of code

1. Figure out how to break each line into individual pieces to understand.
2. Work from the smallest pieces outwards
3. Understand what the line is doing.
4. For loops: think about it first in the context of a single iteration.

Analyzing complicated blocks of code

1. Figure out how to break each line into individual pieces to understand.
2. Work from the smallest pieces outwards
3. Understand what the line is doing.
4. For loops: think about it first in the context of a single iteration.
5. For loops: build up to multiple iterations and then what it does for the whole loop.

Exercises

1. Rewrite this function as a 7 input function.

Exercises

1. Rewrite this function as a 7 input function.
2. Rewrite this function as a 5 input function. Assume computeBestDistance(pic,startX,startY,endX,endY) is computed for you.

Exercises

1. Rewrite this function as a 7 input function.
2. Rewrite this function as a 5 input function. Assume computeBestDistance(pic,startX,startY,endX,endY) is computed for you.
3. Write a specification for the 7 input version

Sound Examples in Detail (8.4)

- We will be covering frequency shifts from 8.4

Sound Examples in Detail (8.4)

- We will be covering frequency shifts from 8.4
- First doubling, then halving

doubleFrequency: Line 1 Change

- def double(source):

doubleFrequency: Line 1 Change

- def double(source):
- Replace with def doubleFrequency(source):

doubleFrequency: Line 1 Change

- def double(source):
- Replace with def doubleFrequency(source):
- What are we doubling? -¿ Be clear

doubleFrequency: Line 1 Details

- def doubleFrequency(source):

doubleFrequency: Line 1 Details

- def doubleFrequency(source):
- Only input is a source sound.

doubleFrequency: Line 2 Details

- len = getLength(source)/2 + 1

doubleFrequency: Line 2 Details

- len = getLength(source)/2 + 1
- recall getLength(source)/2 rounds down. Why do we want to add 1?

doubleFrequency: Line 2 Details

- len = getLength(source)/2 + 1
- recall getLength(source)/2 rounds down. Why do we want to add 1?
- Consider a 3 sample sound: We will later want to sample 0 and 2 but 3/2 is 1. More later.

doubleFrequency: Line 3 Details

- target = makeEmptySound(len)

doubleFrequency: Line 3 Details

- target = makeEmptySound(len)
- We make the target sound with length len.

doubleFrequency: Line 4 Details

- targetIndex = 0

doubleFrequency: Line 4 Details

- targetIndex = 0
- Initialize the targetIndex in preparation to have different source and target indices.

doubleFrequency: Line 5 part 1

- range(0, getLength(source), 2):

doubleFrequency: Line 5 part 1

- range(0, getLength(source), 2):
- Note range has 3 parameters.

doubleFrequency: Line 5 part 1

- range(0, getLength(source), 2):
- Note range has 3 parameters.
- The 3rd parameter is the rate at which we move through indices

doubleFrequency: Line 5 part 2

- for sourceIndex in range(0,getLength(source),2):

doubleFrequency: Line 5 part 2

- for sourceIndex in range(0,getLength(source),2):
- Make note of the fact we are increasing our source by 2 each iteration. Ignoring all samples with odd indices.

doubleFrequency: Line 5 part 2

- for sourceIndex in range(0,getLength(source),2):
- Make note of the fact we are increasing our source by 2 each iteration. Ignoring all samples with odd indices.
- As predicted earlier we have a sourceIndex to go with our targetIndex

doubleFrequency: Line 6

- sourceValue = getSampleValueAt(source,sourceIndex)

doubleFrequency: Line 6

- sourceValue = getSampleValueAt(source,sourceIndex)
- We use getSampleValueAt because we have a numerical index rather than a sample.

doubleFrequency: Line 6

- sourceValue = getSampleValueAt(source,sourceIndex)
- We use getSampleValueAt because we have a numerical index rather than a sample.
- sourceValue is a number containing a sample

doubleFrequency: Line 7

- setSampleValueAt(target, targetIndex, sourceValue)

doubleFrequency: Line 7

- setSampleValueAt(target, targetIndex, sourceValue)
- We use setSampleValueAt because we have a numerical index rather than a sample.

doubleFrequency: Line 7

- setSampleValueAt(target, targetIndex, sourceValue)
- We use setSampleValueAt because we have a numerical index rather than a sample.
- We use targetIndex in the target. The value we copy over is the sourceValue. Fairly standard loop stuff.

doubleFrequency: Line 8

- targetIndex = targetIndex + 1

doubleFrequency: Line 8

- targetIndex = targetIndex + 1
- A standard index increment, nothing unusual here.

doubleFrequency: Line 8

- targetIndex = targetIndex + 1
- A standard index increment, nothing unusual here.
- Wait! There is something important to notice. Take stock of all the loop indices

doubleFrequency: Line 8 to Line 9 interlude

- targetIndex starts at 0 and goes up by 1's.

doubleFrequency: Line 8 to Line 9 interlude

- targetIndex starts at 0 and goes up by 1's.
- sourceIndex starts at 0 and goes by 2's.

doubleFrequency: Line 8 to Line 9 interlude

- targetIndex starts at 0 and goes up by 1's.
- sourceIndex starts at 0 and goes by 2's.
- This loop will copy every second sample from the source to the target.

doubleFrequency: Line 9

- play(target)

doubleFrequency: Line 9

- play(target)
- Plays the sound target

doubleFrequency: Line 9

- play(target)
- Plays the sound target
- What does the target look like? Do we have any special insight?

doubleFrequency: Line 10

- return target

doubleFrequency: Line 10

- return target
- returns the target sound.

doubleFrequency: Take stock of whole function

- Why does the target have double frequency?

doubleFrequency: Take stock of whole function

- Why does the target have double frequency?
- Could we have done this function in a natural way by side effects?

doubleFrequency: Take stock of whole function

- Why does the target have double frequency?
- Could we have done this function in a natural way by side effects?
- No. Target is half the length so want a new sound.

doubleFrequency: Take stock of whole function

- Why does the target have double frequency?
- Could we have done this function in a natural way by side effects?
- No. Target is half the length so want a new sound.
- Unnaturally we could take the target as input and modify it but requiring the user to build the target sound outside is a bad idea.

halveFrequency

- We next analyze halving the frequency. As before we change the function name.

halveFrequency: line 1

- def halveFrequency(source):

halveFrequency: line 1

- def halveFrequency(source):
- takes a source sound as input

halveFrequency: line 2 part 1

- getLength(source) $*2$

halveFrequency: line 2 part 1

- getLength(source) $*2$
- Does doubling the length of the sound make sense?

halveFrequency: line 2 part 1

- getLength(source) $*2$
- Does doubling the length of the sound make sense?
- Yes, each cycle needs to be twice the length. Same number of cycles.

halveFrequency: line 2 part 2

- target = makeEmptySound(getLength(source)*2)

halveFrequency: line 2 part 2

- target $=$ makeEmptySound(getLength(source)$*$2)
- So we use this length to make a sound of the correct length.

halveFrequency:line 3

- sourceIndex = 0

halveFrequency:line 3

- sourceIndex = 0
- Our loop will run over the target, so we should be careful since this is opposite to what we usually do.

halveFrequency:line 4

- for targetIndex in range(0,getLength(target)):

halveFrequency:line 4

- for targetIndex in range(0,getLength(target)):
- The loop runs over the target from 0 to end by 1's

halveFrequency:line 5

- value = getSampleValueAt(source, int(sourceIndex))

halveFrequency:line 5

- value = getSampleValueAt(source, int(sourceIndex))
- So we use get sampleValueAt because we have indices

halveFrequency:line 5

- value = getSampleValueAt(source, int(sourceIndex))
- So we use get sampleValueAt because we have indices
- int! We will likely be incrementing the source by a non-integer value later.

halveFrequency:line 6

- setSampleValueAt(target, targetIndex, value)

halveFrequency:line 6

- setSampleValueAt(target, targetIndex, value)
- A standard target assingment using the targetIndex and At

halveFrequency: line 7

- sourceIndex = sourceIndex + 0.5

halveFrequency: line 7

- sourceIndex = sourceIndex + 0.5
- Our predicted non-integer increment.

halveFrequency: line 7 to line 8 interlude

- What does this loop do?

halveFrequency: line 7 to line 8 interlude

- What does this loop do?
- Copy sample 0 to 0, Copy sample 0 to 1, Copy sample 1 to 2, Copy sample 1 to 3, etc.

halveFrequency: line 8

- play(target)

halveFrequency: line 8

- play(target)
- plays the target. We will analyze the target after.

halveFrequency: line 9

- return target

halveFrequency: line 9

- return target
- returns the target sound

halveFrequency: line 9

- return target
- returns the target sound
- Why does the target have half the frequency?

halveFrequency: line 9

- return target
- returns the target sound
- Why does the target have half the frequency?
- It takes twice as long per cycle.

Demo time

- Have a hypothesis about what our function will do time to run it.