Quantitative and Mathematical Reasoning
Course Proposal Form
For consideration of courses proposed to satisfy the QMR requirement of the
Union College Core Components Curriculum

Instructions
This cover sheet explains the basis on which the CCC Board will evaluate the attached
application form. You’ll find it helpful to read this page in conjunction with the samples of completed forms
available at
http://cs.union.edu/~barrv/CCC/QMR_samples.html
While your course might be quite different in content from the ones sampled, the completed forms should
give you a better sense of the sort of rationale required for approval of a QMR course.

Basis of Evaluation
In the CCC Program that was approved by the AAC in March, 2005, the description
of the Quantitative and Mathematical Reasoning requirement includes this passage:

Courses will be evaluated by the CCC Board to determine if they satisfy the following criteria. Definition:
The primary focus of the course should normally include at least three of the following goals for its students:

(1) to be able to reason quantitatively (to be numerate)
(2) to develop an appreciation for the elegance of mathematics and the beauty and utility of
math as a language for expressing certain types of problems and their solutions
(3) to be able to think logically, solve problems that require multiple steps, and understand
and adapt non-trivial algorithms
(4) to develop the ability to deal with symbolic and abstract representations.

The attached form asks for:
(1) A course description and list of core topics that will be covered in all sections of the course.
(2) A list of three goals (from the above four) addressed by the course.
(3) A description of some sample problems (typically, 2 or 3) taken from the course. Accompany each
problem with an analysis of how the solution requires the student to engage in the type of thinking
described in one or more of the goals, followed by the solution itself. Together, your samples should
address the goals claimed in Part B. Across the aggregate of the sample problems you may claim that
all four goals are addressed, but a strong case for three is preferable to a weak case for all four.

What is part C about? It is the view of the CCC Board that for a QMR course to meet any of the four goals
above, it should require that students solve problems that necessitate the appropriate form of thinking. If a
proposal is for an already existing course, then sample problems should be chosen from ones that have
actually been assigned to students as homework, or that have appeared on tests or quizzes. If a proposal is
for a new course, then sample problems should be ones that realistically will be assigned.

Courses with QMR content may include, appropriately, a significant number of routine “drill” problems. We
understand that the samples you submit will not be of this character, and so may not constitute a
representative sample of all problems assigned or tested. At the same time, we expect them to be
representative of a significant fraction of problems given to your students. We may ask you to elaborate on
this point.

When you explain the ways in which the solution embodies one of the four goals, please appeal as directly
as possible to the specific language of that goal. When you present the solution itself, feel free to use the
same technical terms that you would employ in explaining that solution to a student. In other words, strive to
be technical where appropriate, without bludgeoning us unnecessarily with jargon. And don’t assume that
we’re total idiots! The sample problems and analyses at http://cs.union.edu/~barrv/CCC/QMR_samples.html
should give you a better handle on what it is that we are expecting. Feel free to attach a final sheet on which
you add anything additional about this course that you did not address in the above sections.
DIRECTIONS: Please fill in all information indicated below. Please attach a description of 2-3 sample problems. Each problem should be accompanied by analysis of how the solution requires the students to engage in the type of thinking described in one or more of the goals, followed by the solution itself. The samples should exemplify all the QMR goals addressed by the course.

Department: Computer Science
Date: April 4, 2008

Course number and title:
CSC-107 Creative Computing: Introduction to Computer Science

Course description, core topics which will be covered in all sections of this course:
An introduction to computer science and programming for students who never thought they could successfully tell the computer to do cool things. First we will explore the magic and limitations of computing through a number of physical exercises. Then students will use the Python programming language to do things such as robot simulations, image manipulation, sound splicing, animations, HTML generation and automated reading of web pages.

List each QMR goal addressed by the course, with a brief (one paragraph) explanation of how the course addresses that goal.

(1) to be able to reason quantitatively (to be numerate) – in this course we deal with binary, octal, and hexadecimal arithmetic. Students also have to work with numbers of pixels in an image, numeric location of sound samples

(3) to be able to think logically, solve problems that require multiple steps, and understand and adapt non-trivial algorithms – this is the core of the course since we teach programming. Every assignment involves solving a multi-step problem and then expressing the resulting algorithm in a language the computer can execute.

(4) to develop the ability to deal with symbolic and abstract representations – by definition programming involves the ability to deal with symbolic and abstract representations. First of all, the students are working with images and sound files which are represented as bits within the computer. Second, they are using languages which are essentially abstract representations of instructions that tell the computer what operations to carry out.
Course:
CSC-107 Creative Computing

Problem:
Write a function named hw4. This function will be given a picture as input when it is called, and then mixes and shifts its colors. For each pixel:

- Make the red component 50% of whatever the blue component was plus 50% of whatever the green component was.
- Make the blue component whatever the green component was.
- Make the green component 75% of whatever the red component was plus 25% of whatever the blue component was.

Analysis of solution and its relationship to QMR goals:
Students have to be able to reason quantitatively in order to form the correct equations and determine whether their program is functioning correctly. They have to be able to think logically in order to construct the proper sequence of steps. They are inherently dealing with symbolic representations since they are manipulating a digital expression of the color components in an image.

Solution Method:
```python
def hw4(picture):
    for px in getPixels(picture):
        oldred = getRed(px)
        oldgreen = getGreen(px)
        oldblue = getBlue(px)

        newred = oldblue * .5 + oldgreen * .5
        newblue = oldgreen
        newgreen = oldred * .75 + oldblue * .25

        newcolor = makeColor(newred, newblue, newgreen)
        setColor(px, newcolor)
```
Course:
CSC-107 Creative Computing

Problem:

Part 1 Write a function called changeRed that takes as input a picture and an amount to increase or decrease the red by. The amount will be a number between -.99 and .99

- changeRed(pict, -.10) should decrease the amount of red in the picture by 10%
- changeRed(pict, .30) should increase the amount of red in the picture by 30%

Part 2 Make a copy of your changeRed code. Then, using changeRed as a starting point, write the function changeColor that takes as input a picture, an amount by which to increase or decrease the color, and a number that specifies the color to change.

Analysis of solution and its relationship to QMR goals :
In addition the QMR goals satisfied by the first problem, this problem extends the students' logical thinking to also involve decision making. In addition, it also requires that they take their initial solution and extend that into the solution of a more complex problem, in the same way that many of the assignments extend work that has been done in class.

Solution Method:
# this function will change the red values of pixels in
# picture by the specified percentage (given as a decimal in
# amount). A positive value in amount will cause an increase
# while a negative value in amount will cause a decrease

def changeRed(picture, amount):
    for px in getPixels(picture):
        # get the old red value
        oldred = getRed(px)
        # the new value is the old one plus or minus a percentage
        # of the old value
        newred = oldred + oldred * amount
        # set red to the new value
        setRed(px, newred)

    # now create similar functions to change green and blue
def changeGreen(picture, amount):
    for px in getPixels(picture):
        oldgreen = getGreen(px)
        newgreen = oldgreen + oldgreen * amount
        setGreen(px, newgreen)

def changeBlue(picture, amount):
    for px in getPixels(picture):
        oldblue = getBlue(px)
        newblue = oldblue + oldblue * amount
        setBlue(px, newblue)

# this function will increase or decrease one color, based on
# the value of whichOne: 1 -> red, 2->green, 3->blue
def changeColor(picture, amount, whichOne):
    if whichOne == 1:
        changeRed(picture, amount)
    else:
        if whichOne == 2:
            changeGreen(picture, amount)
        else:
            if whichOne == 3:
                changeBlue(picture, amount)
            else:
                if whichOne <> 0:
                    print "illegal argument passed to changeColor"
# Sample Problem

## Course:

| CSC-107 Creative Computing |

## Problem:

Write a function named `audioCollage` that will create an audio collage. You must compose together **at least two separate sounds** (from two separate files), and your result must be **at least six seconds** long. **One of the sounds must appear at least twice where it is changed in some way.** I must be able to hear your collage from the returned sound. I will run your function from the control panel as follows:

```python
>>> setMediaPath()     #setting media path to my directory
>>> play(audioCollage())   #calling your function, playing returned result
```

There are a variety of ways in which you can change sounds:

- increasing, reducing, or normalizing volume
- reversing sounds
- splicing out parts of a sound, splicing in parts of a sound
- change frequency of a sound
- blending sounds (not discussed in class, but it's in the book)
- any other sound technique discussed in the book or in class

You can use any sounds you want, from the mediaSources directory on the CD, the words in the Speech folder on the CD, other sounds that you locate.

`audioCollage()` can call other functions, which must be in your .py file!

## Analysis of solution and its relationship to QMR goals:

This problem requires a multi-step solution with decisions and iteration. It requires logical thinking, a degree of quantitative reasoning to get the sound samples properly aligned, and symbolic representation of sound as digital data.

## Solution Method:
def audioCollage():
    target=makeSound(getMediaPath("sound1.wav"))
    length(target) #prints length in seconds
    for count in range(1,5):
        if(count==1): #add sound one to Knight Rider Theme
            countInt(target, "sound2.wav", 1)
        if(count==2): #add sound one to Knight Rider Theme
            countInt(target, "sound3.wav", 100000)
        if(count==3): #add sound two to Knight Rider Theme
            splice(target, "sound3.wav",150000, 11874, 19376)
        if(count==4):
            mirrorSound(splice(target, "sound2.wav", 220000, 19307, 31116))
    return(target)

#this can take any sound and insert it into the target
def countInt(target, file, number):
    source=makeSound(getMediaPath(file))
    targetIndex=number
    for index in range(1,getLength(source)+1):
        volume=getSampleValueAt(source, index)
        setSampleValueAt(target, targetIndex, volume)
        targetIndex= targetIndex + 1
    return target

#splicing the sound (calls target, source, target location, and ranges a/b from source)
def splice(target, file, destSample,a,b):
    files = getMediaPath(file)
    source = makeSound(files)
    dest = makeSound(files)   # This will be the newly spliced sound
    for number in range(1,3):
        for srcSample in range(a, b): #where calibrate is
            setSampleValueAt(target, destSample, getSampleValueAt( source, srcSample) )
        destSample = destSample + 1
    return target

#this mirrors the sound from the beginning and puts it at the end.
def mirrorSound(sound):
    mirrorpoint=getLength(sound)/2
    for sOff in range(1,mirrorpoint-1):
        samplelater = getSampleObjectAt(sound, mirrorpoint+sOff)
        samplebefore = getSampleObjectAt(sound, mirrorpoint-sOff)
        value= getSample(samplebefore)
        setSample(samplelater,value)
    return sound

#Print length of file in seconds
def length(file):
    length = getLength(file)
    rate=getSamplingRate(file)
    sec=length/rate
    print "The sound is ", sec, " seconds long, a lot longer than 6 seconds."