CS111 EXAM 1

March 05, 2015

YOUR NAME*: **EXAM 1 SOLUTIONS**

*by writing your name above you are stating that you abided by the course policies while taking this exam

Please indicate your lecture by checking the appropriate box (so we can return your exam to you):

❑Lyn 9:50am ❑ Rhys 11:10am ❑ Rhys 1:30pm

This exam has 5 problems. Some problems have several parts. The number of points for each problem is shown in square brackets next to the problem or part. There are 100 total points on the exam.

Write all your answers on the exam itself.

The exam is open book. You may refer to your notes, and other course materials **except that you may not use another person's notes, or any materials from prior semesters of CS111. You may not access any electronic device at any time.**

Please keep in mind the following tips:

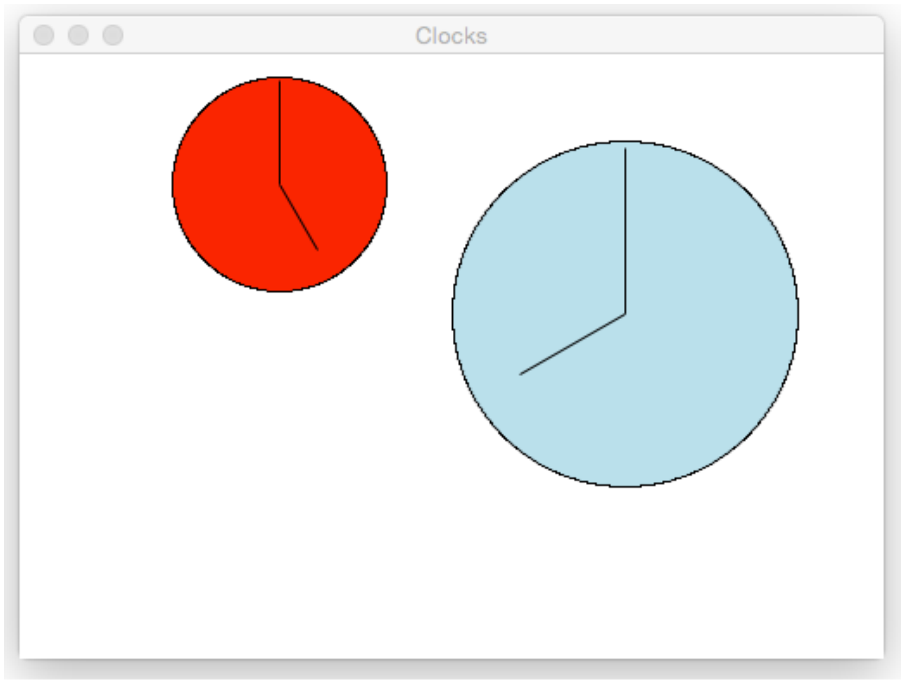
- *First skim through the entire exam.* Work first on the problems on which you feel most confident. You do not need to do the problems in the order they are presented.
- *Try to do something on every problem* so that you can receive partial credit. For programming problems, you can receive partial credit for explaining your strategy with words and pictures.
- *Allocate your time carefully.* If you are taking too long on a problem, wrap it up and move on.

The following table will be used in grading your exam:

Problem	Score
Problem 1 [21 pts]	
Problem 2 [12 pts]	
Problem 3 [16 pts]	
Problem 4 [12 pts]	
Problem 5 [39 pts]	
Total [100 pts]	

Problem 1: Capture the Pattern [21 points]

I must govern the clock, not be governed by it – Golda Meir.



Using **cs1graphics**, the canvas above can be produced with the code below:

```
from cs1graphics import *

wall = Canvas(500,350,'white','Clocks')

# Draw first clock, red, diameter 125, time 5pm, at (150,75):
clock1 = Layer()
clockface1 = Circle(125/2.0) # 125 is clockface diameter
clockface1.setFillColor('red')
clock1.add(clockface1)
minutehand1 = Path(Point(0,0), Point(0,-125*.48))
hourhand1 = Path(Point(0,0), Point(0,-125*.35))
hourhand1.rotate(5*30.0) # 5 o'clock * 30 degrees/hour
clock1.add(minutehand1)
clock1.add(hourhand1)
clock1.moveTo(150,75)
wall.add(clock1)

# Draw second clock, lightblue, diameter 200, time 8pm, at (350, 150):
clock2 = Layer()
clockface2 = Circle(200/2.0) # 200 is clockface diameter
clockface2.setFillColor('lightblue')
clock2.add(clockface2)
minutehand2 = Path(Point(0,0), Point(0,-200*.48))
hourhand2 = Path(Point(0,0), Point(0,-200*.35))
hourhand2.rotate(8*30.0) # 8 o'clock * 30 degrees/hour
clock2.add(minutehand2)
clock2.add(hourhand2)
clock2.moveTo(350,150)
wall.add(clock2)
```

Part (a) [12 points]

Capture the repeated pattern in the code above by creating a function, called **`drawClock`**, that can be used to draw clocks on the Canvas, such as those shown above. In part (b) on the next page, you will write three invocations of your **`drawClock`** function to draw the two clocks and then add a third in part (c).

Here, in part (a), you must define the **`drawClock`** function. Your **`drawClock`** function should take 6 parameters that provide the following information: the Canvas object to draw the clock on, the x-coordinate on the Canvas where the center of the clock should be placed, the y-coordinate on the Canvas where the center of the clock should be placed, the diameter of the clock, the color of the clock face, and the time the clock should display (an integer between 1 and 12).

```
def drawClock(canvas, x, y, diameter, color, hour):
    clock = Layer()
    clockface = Circle(diameter/2.0)
    clockface.setFillColor(color)
    clock.add(clockface)
    minutehand = Path(Point(0,0), Point(0,-diameter*.48))
    hourhand = Path(Point(0,0), Point(0,-diameter*.35))
    hourhand.rotate(hour*30.0)
    clock.add(minutehand)
    clock.add(hourhand)
    clock.moveTo(x,y)
    canvas.add(clock)
```

Part (b) [6 points]

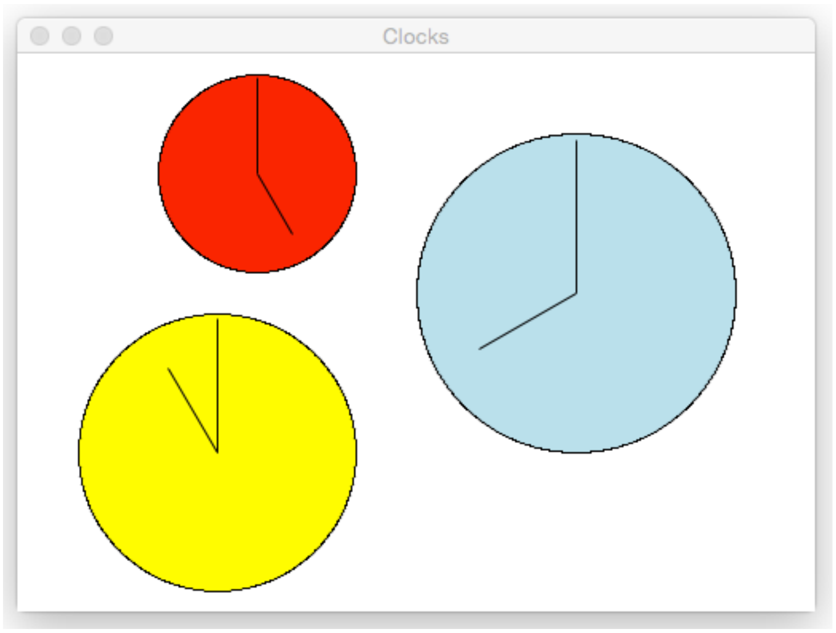
Write the two invocations of your **`drawClock`** function that will replace all of the code on page 2 except for the first two lines.

```
# Draw first clock, red, diameter 125, time 5pm, at (150,75):
drawClock(wall, 150, 75, 125, 'red', 5)

# Draw second clock, lightblue, diameter 200, time 8pm,
# at (350, 150):
drawClock(wall, 350, 150, 200, 'lightblue', 8)
```

Part (c) [3 points]

Below, write one new invocation of your **`drawClock`** function that will additionally place a yellow clock of diameter 175 showing 11pm with its center at (125, 250) as shown in this image:



Write your invocation here:

```
# Draw third clock, yellow, size 175, time 11pm, at (125, 250):
drawClock(wall, 125, 250, 175, 'yellow', 11)
```

Problem 2: Duplicate Removal [12 points]

Define a function named **`removeDuplicates`** that takes a list as its parameter and returns a new list consisting of all elements in the original list but without any duplicate elements. For example:

```
removeDuplicates([1, 2, 2, 3, 2, 3, 1])
returns [1, 2, 3]

removeDuplicates(['john', 'george', 'paul', 'ringo', 'john',
                  'paul', 'jones', 'rhys', 'price', 'jones' ])
returns ['john', 'george', 'paul', 'ringo', 'jones', 'rhys', 'price']

removeDuplicates(range(0,10,2)+range(0,10,3))
returns [0, 2, 4, 6, 8, 3, 9]
```

The order of elements in your returned list does not matter. For example, it is fine for your function to return **`[2, 3, 1]`** for the first example above.

```
def removeDuplicates(elts):
    result = [] # result is initially an empty list
    for elt in elts: # iterate over all elements in elts
        if elt not in result: # include each element
                              # not already in result
            result.append(elt)
    return result
```

Problem 3: Understanding Conditionals [16 points]

Fill in the table below to indicate the printed output for various values of the input variable **x**.

```
x = int(raw_input("Enter an integer x: "))
answer = ''
if x >= 60:
    answer = answer + 'A'
    if x < 75:
        answer = answer + 'B'
    elif x > 90:
       answer = answer + 'C'
    else:
       answer = answer + 'D'
elif x >= 50:
    answer = answer + 'E'
else:
    answer = answer + 'F'
if x < 35:
    answer = answer + 'G'
else:
    answer = answer + 'H'
    answer = answer + 'I'
print(answer)
```

x	**printed output**
25	FG
35	FHI
45	FHI
55	EHI
65	ABHI
75	ADHI
85	ADHI
95	ADHI

Notes:

- There are two top-level `if` statements that are executed sequentially, each of which concatenates at least one letter to answer. So each answer will have at least two letters.
- The else clause of the second top-level `if` statement (when x >=35) adds two letters.
- The first test of the first top-level `if` statement (x >= 60) has a nested `if` statement that adds an extra letter.
- The number itself is never displayed by the code.

Problem 4: Random Pairs [12 points]

In the Python **random** module, there is a function named **random.choice** that takes a list and returns a randomly selected element from that list. For example, here are some sample invocations:

```
myList = [8, 2, 7, 5]
random.choice(myList)  returns 7
random.choice(myList)  returns 8
random.choice(myList)  returns 5
random.choice(myList)  returns 5
random.choice(myList)  returns 8
```

Write a function called **randomPair** that, given a list with at least two different elements, returns a pair (i.e., a tuple with two elements) of two distinct randomly selected elements from the given list.
randomPair should not modify the input list in any way. Here are some sample invocations of **randomPair**:

```
randomPair(myList) returns (5, 2)
randomPair(myList) returns (7, 8)
randomPair(myList) returns (2, 7)
randomPair(myList) returns (8, 5)
randomPair(myList) returns (7, 8)
```

Note that **randomPair(myList)** should never return (8,8) or any pair that contains two of the same element.

In your definition, employ this strategy: randomly choose the first element of the pair from the list, and then repeatedly randomly choose a second element from the list until that second element is different from the first element. You should not invoke any function other than **random.choice**

```
def randomPair1(elts):
    elt1 = random.choice(elts) # Pick first element of pair
    elt2 = random.choice(elts) # Pick second element of pair
    while elt2 == elt1: # Repick second element until it's
                        # different from first element
       elt2 = random.choice(elts)
    return (elt1, elt2)
```

Problem 5: Understanding Loops & Sequences [39 points]

Part (a) [12 points]

Define a function named **`includesNumberBetween`** that takes three parameters:

(1) a low number
(2) a high number
(3) a list of numbers

It returns **`True`** if one of the numbers in the list is between the low number and the high number (inclusive). Otherwise it returns **`False`**.

`includesNumberBetween(18, 40, [15, 7, 78, 63, 42])` returns False
`includesNumberBetween(40, 42, [15, 7, 78, 63, 42])` returns True
`includesNumberBetween(20, 100, [15, 7, 78, 63, 42])` returns True
`includesNumberBetween(65, 75, [15, 7, 78, 63, 42])` returns False
`includesNumberBetween(70, 80, [15, 7, 78, 63, 42])` returns True
`includesNumberBetween(15, 20, [15, 7, 78, 63, 42])` returns True

Define your function here:

```
def includesNumberBetween(lo, hi, nums):
    for num in nums:
        if num >= lo and num <= hi: # Check if number is in range
            return True # If number in range, exit loop
                        # (and function) immediately with True
    return False # only get here if no number in nums was in range
```

Part (b) [15 points]

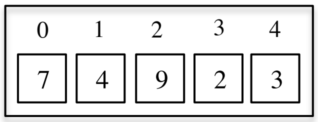
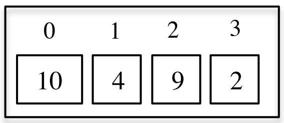
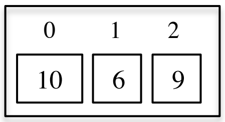
Consider the following functions **`loopy1`** and **`test_loopy1`**:

```
def loopy1 (nums):
    i = 0
    while i < (len(nums) - 2):
        nums[i] = nums[i] + nums.pop()
        i = i + 1

def test_loopy1():
    testList = [7, 4, 9, 2, 3]
    loopy1(testList)
    print(str(testList))
```

What is printed by the invocation **`test_loopy1()`**? To receive full credit, you **must** show your work, which can include iteration tables and memory diagrams.

Recall that lists are mutable. `nums[i] = …` changes the ith slot of the list `nums`, and `nums.pop()` removes the last slot from a list and returns the value of the last slot before it is removed. Below is an iteration table that shows the state variables of the iteration at the beginning of each `while` loop iteration. Note that `testList` and `nums` name the same list in memory, and that since `nums.pop()` removes a list slot, the value of `len(nums)` changes with each step of the loop (a fact missed by many students).

`i`	`len(nums)-2`	`nums[i]`	`nums/testList` (before pop)	`nums.pop()`
0	3	7		3
1	2	4		2
2	1	*not evaluated*		*not evaluated*

The result printed by `test_loopy1()` is **`[10, 6, 9]`**.

Many students who did not realize that the expression `len(nums) – 2` changed over time attempted to execute the `while` loop body when `i` is 2. This is problematic, because performing `nums.pop()` in this step removes the slot with index 2, and then the slot assignment `nums[2] = …` fails with an index out of bound error because there is no longer a slot with index 2 that can be assigned.

Part (c) [12 points]

Below is a function **`loopy2`**:

```
def loopy2 (width, height, string):
    for y in range(height):
        line = ''
        for x in range(width):
            line = line + string[(y*width + x)%len(string)]
        print(line)
```

What is printed by the invocation **`loopy2(5,3,'code')`**?

In this program:

- `height` is 3 and `range(3)` is the list `[0,1,2]`
- `width` is 5 and `range(5)` is the list `[0,1,2,3,4]`
- `string` is `'code'` and `len('code')` is 4

The following iteration table shows the steps in the nested loop. Although somewhat complicated, it has exactly the same column and row arithmetic as the `eggCarton` function from PS5. Indeed, the purpose of this problem was to see if you really understood the `eggCarton` function.

`y`	`x`	`line` (before assignment)	`y*5 + x`	`(y*5 + x)%4`	`'code'[(y*5 + x)%4]`	`line` (after assignment)
0	0	`''`	0	0	`'c'`	`'c'`
0	1	`'c'`	1	1	`'o'`	`'co'`
0	2	`'co'`	2	2	`'d'`	`'cod'`
0	3	`'cod'`	3	3	`'e'`	`'code'`
0	4	`'code'`	4	0	`'c'`	`'codec'`
`print('codec')`						
1	0	`''`	5	1	`'o'`	`'o'`
1	1	`'o'`	6	2	`'d'`	`'od'`
1	2	`'od'`	7	3	`'e'`	`'ode'`
1	3	`'ode'`	8	0	`'c'`	`'odec'`
1	4	`'odec'`	9	1	`'o'`	`'odeco'`
`print('odeco')`						
2	0	`''`	10	2	`'d'`	`'d'`
2	1	`'d'`	11	3	`'e'`	`'de'`
2	2	`'de'`	12	0	`'c'`	`'dec'`
2	3	`'dec'`	13	1	`'o'`	`'deco'`
2	4	`'deco'`	14	2	`'d'`	`'decod'`
`print('decod')`						

Note that: (1) `line` is reinitialized to the empty string `''` for each iteration of the outer loop;
(2) each iteration of the inner loop adds one character from `'code'` to the end of `line`. Since there are five such iterations, `line` has a maximal length of 5.
(3) print is called once for each iteration of the outer loop, so it is called 3 times on strings of length 5.

The final printed result has three lines of five characters each (excluding the terminating newline character):

```
codec
odeco
decod
```

This is the end of the exam.