CSSE 120 Exam 2 Topics and Sample Problems

Format: The exam will have two sections:

- **Part 1: Paper-and-Pencil.** Closed-everything, except that you may bring a one-page (back and front) "cheat sheet" with whatever you want on it. Approximately 30 points of 100.
- **Part 2: On-the-computer.** Open-everything, except that you may not communicate with anyone except your professor on this part. Resources you may use include your textbook, your notes, the Internet, homework problems, and the course web site. Approximately 70 points of 100.

Closed book topics: *All* of the closed-book portion of the exam will be drawn from the following: **For the closed-book portion of Exam 2, students should be able to:**

1. ***Trace short snippets of code*** (less than, say, 10 lines or so) **and show what gets printed.** The code might include anything from the first portion of the course (through Exam 1), but will emphasize the following topics, mostly from the second portion of the course:
    - While loops
    - Nested loops
        - When the inner loop does not depend on the outer loop variable, and also when it does.
    - Boolean operators (and, or, not)
    - Function definitions and calls, also method calls
        - Especially with parameters
        - Recall that when you return from a function, the arguments refer to the same object (or number) that they did before the function was called, but the object may have mutated (e.g. if it was a list) during the function call.
    - Dictionaries
        - Creating a dictionary, adding key/value pairs to a dictionary, modifying the value associated with a key, deleting a key/value pair, determining whether a key is in the dictionary. The two types of uses of dictionaries.
2. ***Loop*** through a list or string, ***building and returning a new list or string***.
3. Do ***counting*** and ***summing*** (examples of the ***accumulation*** pattern) over a list or other collection of values.
4. Find the ***max*** and ***min*** over a list or other collection of values.
5. Explain what a ***sentinel*** is and how it is used.
6. Write simple ***sentinel loops***, e.g. reading numbers until the user enters the null string.
7. Write short functions with ***parameters*** that ***return values***.

8. Distinguish between ***printing*** and ***returning values*** from a function. Capture returned values from a function in a variable. Return and capture multiple values from a function.
9. Apply ***Boolean operators*** (and, or, not) and relational operators (<, >, <=, >=, ==, !=)
10. Explain what ***top-down design*** and ***procedural decomposition*** are.
11. Explain what it means to use ***documented stubs*** before coding.
12. Explain any of the main ideas that your team used in your ***Robotics project***.
13. Explain the difference between the following notations:

    ```
    math.sin(5)
    circle.draw(window)
    ```

    In particular, understand the ***implicit*** and ***explicit parameters*** in the latter.
14. Understand the implications of the fact that variables are ***references*** to their values.
15. Know how and why we put:
    - ***Comments*** in our programs.
    - ***Documentation strings*** in our programs.
16. Do anything else from the closed-book topics of Exam 1.
    - But the closed book portion will emphasize the preceding topics, not the topics from Exam 1 (except those that are duplicated above).

<u>Open-everything topics, Big Ideas</u>: These are the most important ideas that we have covered since the first exam (some are repeated here from the material of the first exam). Much of the open-everything portion of the exam will be drawn from the following.

1. Apply ***loop patterns***:
    - The ***for loop*** pattern
    - The ***while loop*** pattern
        - ***Interactive loop***
        - ***Sentinel loop using impossible values*** as the sentinel
        - ***Sentinel loop using no-input*** as the sentinel
    - The ***loop-and-a-half*** pattern
        - Combined with use of no-input as a sentinel
    - The ***file loop*** pattern
        - Also, how to ***split*** words on a line into a list of words
        - Also, how to convert strings that represent numbers into numbers
    - ***Nested loops***
        - When the interior loop does not depend on the exterior loop variable
        - When the interior loop DOES depend on the exterior loop variable
    - ***Wait-until-event loop***
    - ***Saving-a-value-for-the-next-iteration*** pattern: Inside a loop:

      x = [some expression involving *previous_x*]

      …

      previous_x = *x* [this normally appears near the end of the loop]
2. Do ***top-down design*** and ***procedural decomposition***
3. ***Loop*** through a list or string, ***building and returning a new list or string***.
4. Do ***counting*** and ***summing*** (examples of the ***accumulation*** pattern) over a list or other collection of values.
5. Find the ***max*** and ***min*** over a list or other collection of values.
6. ***Use a dictionary*** in what we called dictionary use #1: the dictionary is associated with a single object and stores attributes of the object, all under the single name of the dictionary. For example, each dictionary in the exercise we did was a student. There was a list of such dictionaries, one item in the list for each student.
7. Do ***zellegraphics***, including: constructing a GraphWin; constructing circles, rectangles, lines, points, polygons, ovals and text objects. Getting the mouse click and using it. Getting data from an Entry box. Using images.

Sample closed-book problems:

1. What is the output of each of the following code fragments?

a.

```
def silly(x):
    print "start silly"
    print x
    funny(2 * x)
    goofy(x - 1)
    print "end silly"

def funny(y):
    print "start funny"
    print y
    goofy(y + 1)
    print "end funny"

def goofy(z):
    print "start goofy"
    print z
    print "end goofy"

silly(5)
```

Output:

b.

```
numRows = 4
for i in range(numRows):
    print " " * i,
    for j in range(numRows - i):
      print "*",
    print
```

Output:

c.

```
a, b = 10, 64
while a < b:
     print a, b
     a = a - 1
     b = b / 2
```

Output:

d.

```
def changeMe(x, y, z):
    x += z[1]
    y = [3, 6, 9]
    z[0], z[1] = z[1], z[0]

a = 10
b = [10, 20, 30]
c = [3, 22, 68]
changeMe(a, b, c)
print a, b, c
```

Output:

2. What gets printed by the following code:

```
circleA = Circle(Point(25, 25), 10)
circleB = Circle(Point(50, 50), 20)
circleC = Circle(Point(75, 75), 30)
circleC = circleB
circleB = circleA
circleA.move(100, 0)
circleB.move(200, 0)
print circleA.getCenter().getX()
print circleB.getCenter().getX()
print circleC.getCenter().getX()
```

Output:

3. Write a function that takes a list of numbers and two integers, and returns a list of all numbers in the given list that are between the two integers (including the endpoints). You may assume that the first of the two integers is less than or equal to the second of the two integers.
4. If you are using top-down design, you should design the contents of which function first?
5. Explain what a *sentinel* value is and give a concrete example of its use.
6. Write the definition of a Python function called *inorder* that takes three numeric parameters and returns a Boolean value indicating whether the three numbers are in increasing order. For example, `inorder(3, 6, 9)` returns `True`, and `inorder(4, 2, 8)` returns `False`. Note that `True` is NOT the same as `'True'`.

<u>Sample open-book problems:</u>

1. Threshold. Define a function `threshold(dailyGains, goal)` that behaves as follows:

   The first parameter is a list of numbers that represent daily gains in a stock's value. The second parameter is a single positive number that is a profit goal. The function should return the minimum number of days for which the stock must be owned in order to achieve a profit at least as large as the stated goal. In the case that the goal is unreachable, the function should return 0.

   For example, `threshold([5,3,8,2,9,1], 17)` should return 4 because the goal can be reached after the first 4 days (5+3+8+2).

   Another example: `threshold([5,3,8,-2,9,1], 47)` should return 0.
2. Duplicates. Write a function that allows a user to enter words (one per line), continuing until the user first enters a duplicate, as in:

   ```
   Start entering words:
   kiwi
   apple
   plum
   watermelon
   banana
   apple
   You already entered apple.
   You listed 5 distinct words.
   ```
3. Input into a list. Write a function *input_into_list* that allows the user to enter numbers. The user can enter as many numbers as she wants on a line. If she enters nothing on a line, that signals end of input. The function should return a list of all the numbers entered by the user, in the order in which they were entered.

   You may assume that the user only enters numbers, except when she enters nothing to signal end of input. That is, you don't have to deal with malformed inputs.

4. Any of the problems in the ***dictionariesUse1.py*** module of the ***22-Dictionaries*** project.
5. Any of the problems in the ***14-NestedLoops*** project.
6. Any of the problems in the ***13-LoopPatterns*** project.