CSSE 120 Exam 1 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 20 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 80 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 1, 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:
   - Variables and assignment, including parallel
   - Arithmetic expressions
   - Lists and strings, including indexing, slicing and concatenation
   - Definite loops and range expressions
   - Function definitions and calls, also method calls
   - Conditionals, relational operators

d. What gets printed by the following function calls:

```
def g(a, b):
    print "(" , a , "of" , b , ")"

def f(n):
    for i in range(1, n + 1):
        g(i, n)
        g(n, i)
        g(i, i)
    return n ** 2, n ** 3

x, y = f(3)
print x + y
```

1. Problems in boxes (like this one) are sample problems for the correspondingly numbered topic. For example, these sample problems are for topic #1 to the left.

a. What gets printed by the following arithmetic operations:

- `5.0 / 2 + 2`
- `3.0 + 1 / 2`
- `18 % 4`
- `18.0 // 4.0`
- `2 ** 4`

b. What gets printed by the following loops:

- ```
  for k in range(1, 4):
      print k, k ** 2,
      print k ** 3,
  print "Done"
  ```
- ```
  for j in range(3):
      print "Hello"
  print "Goodbye"
  ```

c. What gets printed by the following range and list expressions

- `range(2, 11, 3)`
- `""" 'hello' """`
- `"one" + "two"`
- ```
  x = [10, 20, 30, 40, 50]
  print x[2]
  ```
- ```
  y = "forget me not"
  print y[2]
  ```

e. What gets printed by the following range and list expressions:

```
quotesList = ["Looks like you've been missing a lot of work lately.",
              "I wouldn't say I've been *missing* it, Bob."]
print quotesList[1]
print quotesList[1][2]
print quotesList[1][2:4]
```

f. What gets printed by the following code that includes method calls:

```
circleA = Circle(Point(25, 25), 10)
circleB = circleA
circleA.move(15, 0)
print circleA.getCenter().getX()
print circleB.getCenter().getX()
```

g. What gets printed by the following code that includes function calls and lists:

```
def foo(x, y):
    x[0] = 100
    y = [200, 300, 400]
    y[0] = 666

x = [1, 2, 3]
y = [4, 5, 6]
foo(x, y)
print x, y
```

2. **Write range expressions** for common ranges.

2. Write range expressions that generate:

   a. [4, 5, 6, 7, 8]

   b. [10, 8, 6, 4, 2, 0, -2]

   c. [0, 1, 2, 3]

3. Understand the implications of the fact that variables are ***references*** to their values.

3. See problems f and g above. Also:

   True or false: after the function call **`foo(x)`** executes, *x* can refer to a different object than it did before the call.

   True or false: after the function call **`foo(x)`** executes, the object that *x* refers to can have different characteristics than it had before the call.

4. Know how and why we put:

   a. *Comments* in our programs.

   b. *Documentation strings* in our programs.

4. Why do we put:

   a. *Comments* in our programs?

   b. *Documentation strings* in our programs?

   What is the notation for each of the above?

5. Explain that ***objects know stuff*** (stored in *instance variables*) and ***can do stuff*** (via *methods*).

5. Objects _____ and _____.

6. Explain that Python is an ***interpreted*** language.

6. Explain "Python is an interpreted language."

Open-everything topics, Big Ideas: These are the most important ideas that we have covered.

Much of the open-everything portion of the exam will be drawn from the following.

For Exam 1, students should be able to:

1. **Write short programs and/or functions that are examples of the *input/compute/output* pattern**. Be able to:
   a. Use *input* (or *raw_input* for strings) to get input from the console
   b. Use variables to store the input and perform numeric computations. Use *integer*, *float* or *long* types. Use operators:
      `+  -  *  /  %  //  **`
   c. Use *print* to display results on the console

1. Sample problems:
a. Write a program that prompts the user for a *radius* and a *height* and inputs values for those variables. The program then prints the volume of the cylinder with that radius and height. (Hint: $V = \pi r^2 h$)
b. Write a function that … [same as above, but in a function]
c. Write a function that prompts the user for two strings and prints the first letter of the second string immediately followed by the last letter of the first string.

2. ***Define functions* that have *parameters* and (possibly) *return values*.**
   a. Explain the form of a function definition
   b. Write the *function body*, using the parameters and *local variables* as needed
   c. Return a value (possibly a *tuple*) if called for by the problem
   d. Explain what a *parameter* is. Explain its relationship to variables with the same name outside the function.
   e. Explain what a *local variable* is. Explain its relationship to variables outside the function with the same name.

2. Sample problems:
a. Write a function called *cylinderVolume* that receives two parameters: a *radius* and a *height*. The function returns the volume of the cylinder with that radius and height. (Hint: $V = \pi r^2 h$)
b. Write a function called *drawPolygon* that takes 3 parameters: a list of Points, a GraphWin and a string that represents a color. The function creates a Polygon of the given color with the given Points, and then draws that Polygon on the given GraphWin.
c. Write a function called *max_min* that takes two numbers and returns a 2-tuple: the larger of those two numbers and the smaller of those two numbers.
d. Explain, by giving a concrete example, what a *parameter* is. What a *local variable* is.

3. ***Call functions,* both ordinary functions and *methods*, and capture the returned value(s) (if any) in variable(s)**

3. Sample problems:
a. Write an expression that prints the absolute value of *x*, using the built-in function *abs*.
b. Write an expression that sets the variable *z* to the sum of the numbers in list *x* and those in list *y*, using the built-in function *sum*.
c. Write a function called *use_max_min* that has two parameters *x* and *y* and uses *max_min* (as specified in the previous problem) to print the square of the larger of *x* and *y*, followed by the cube of the smaller of *x* and *y*.

(continues on the next page)

d. Suppose you have defined the *drawPolygon* function described in the previous set of sample problems. Call this function with appropriate values, constructing objects as needed.

e. Suppose that you have three Point objects (where Point is defined in *zellegraphics*), called *p1*, *p2* and *p3*. Write an expression that prints the sum of their x-coordinates.

f. Suppose that you have a Circle object (where Circle is defined in *zellegraphics*), called *circle1*. Write a statement that sets the fill color of *circle1* to 'red'.

4. Use *definite loops* and *sequences*

a. Generate indices by using a *range* statement (used in *counted* loops)

- *range(n)* *range(m, n)* *range(m, n, d)*

b. *Iterate* through a list or string

c. Use the *accumulator pattern*, accumulating a:

- sum
- count
- product
- max
- min
- list
- string

g. Write a function called print_list2 that takes a list w and prints the following:

```
1. element 0 of the list w
2. element 1 of the list w
   . . .
```

etc. through the last element in list *w*.

h. Write a function called print_list_odd that takes a list w and prints the odd-numbered elements (i.e., elements 1, 3, 5, 7, etc – numbering starts at 0), all on a single line, separated by spaces.

i. Write a function called count_lower that takes a string and returns the number of lower-case alphabetic letters in the string.

4. Sample problems:

a. Use *range* expressions to generate the following:

- [3, 5, 7, 9, 11]
- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- [10, 9, 8, 7]

b. Write a function called *print_list* that takes a list *w* and prints the elements of list *w*, one per line:

- Using a *range* expression (and whatever else is necessary to answer this question)
- Without using a *range* expression

c. Write a function called accumulator that takes a list of numbers and returns a 4-tuple with:

- the sum of the cubes of the numbers
- a list containing the cubes of the numbers
- the number in the list whose cube is largest
- how many of the numbers are positive

d. Write a function called *backwards* that takes a string *s* and prints *s* backwards.

e. Write a function called *reverse* that takes a string *s* and returns a string containing all the characters of *s* in reverse order.

f. Write a function called *reverse* that takes a string *s* and returns a string containing all the characters of *s* in reverse order.

5. Use *conditional* statements:

- if
- else
- elif

5. Sample problems:

a. Write a function called *only_abc* that takes a string and returns *True* if the string contains only the letters a, b, or c (upper or lower case), else returns *False*.

b. Write a function that takes a number *score* and prints whether the associated letter grade is an A, B, C, D or F (in the usual 10-point scale). You may assume the number *score* is between 0 and 100.

c. Write a function that takes three numbers and returns the "middle" one: e.g. if the numbers are 100, 110 and 90, the returned value is 100.

6. Use *objects*:

a. *Construct* an object

b. Apply *methods* to the object

c. Determine what methods apply to an object

d. Explain that every object is an *instance* of some *class*.

6. Sample problems:

a. Write a statement that constructs a Point object whose x-coordinate is 100 and whose y-coordinate is 50, and assigns the constructed object to a variable called *p1*.

b. Write a function that takes a Rectangle and returns a 2-tuple with its area and perimeter.

c. Write a statement that constructs a Point object whose x-coordinate is 100 and whose y-coordinate is 50, and assigns the constructed object to a variable called *p1*.

d. Write a function that takes a Rectangle and returns a 2-tuple with its area and perimeter.

e. Write a statement that sets the fill color of the Circle called *c* to 'green'.

f. Suppose that I gave you a new module that contains methods that apply to Student objects. How would you, in Eclipse, determine what methods a Student can do?

7. Apply the above to *zellegraphics*:

a. Construct a GraphWin

b. Construct Points, Lines, Circles, Rectangles, Polygons, Text, etc.

c. Use Entry objects

d. Apply methods to the above, including *draw*, *undraw*, *getters* and *setters,* and *getMouse*

e. Do an animation (using *time.sleep*)

7. Sample problems:

a. Write a module called *circles* that displays a GraphWin and causes the following to happen 5 times: when the user clicks the mouse, the program draws a circle at the point where user clicked.

b. Same as the previous, but when the user clicks the previously-drawn circle disappears.

c. Write a module called *lines* that displays a GraphWin and causes the following to happen 5 times: when the user clicks the mouse twice, the program draws a line from the first mouse-click to the second one.

d. See the next page for a longer problem.

A longer problem:

Write a Python program (in file *house.py*) that allows the user to provide five clicks incrementally that it uses to draw a simple five-click-house in a 600 x 500 graphics window, as depicted in the figure below. Work in stages:

a. Prompt the user to click twice in the window, upper left corner first, to specify the opposite corners of the frame of the house. The program should remember these points and use them now to draw the rectangular frame of the house.

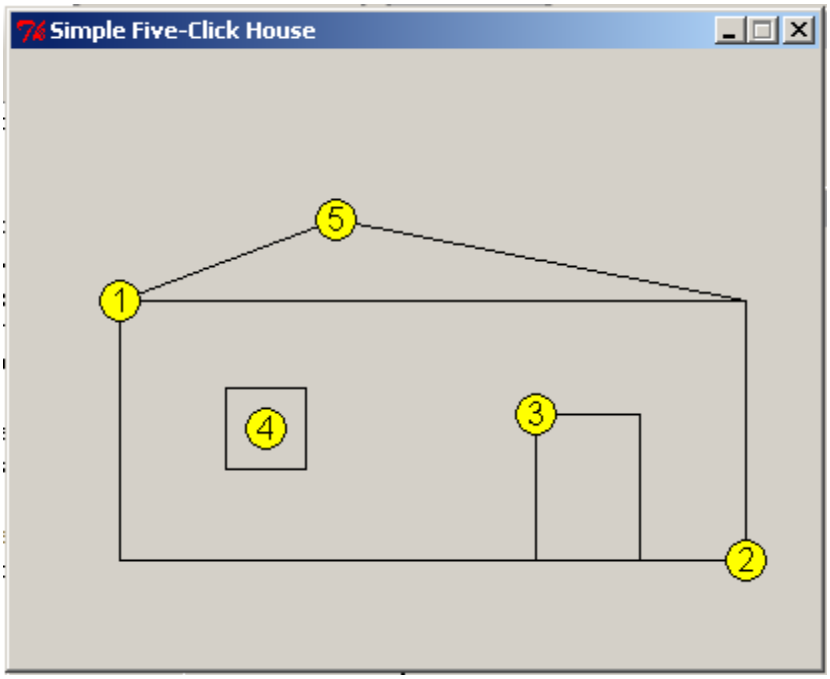
b. Prompt the user to click once on the window to specify the upper left corner of the front door. The program should draw a rectangular door that is 1/5 the width of the rectangular frame; vertically it goes from the upper left corner point to the base of the rectangular frame.

c. Prompt the user to click once to specify the center of a square window. The program should draw a square window whose side is half the height of the front door.

d. Prompt the user to click once, above the house frame, to specify the peak of the roof. The program should draw a triangular roof that extends from the point at the peak to the corners at the top edge of the house frame. [Recall zellegraphics Polygon.]

e. Animate the set of shapes so they slowly move all together off the screen.

The picture below shows where the user clicked to get the picture. Your picture should NOT include the circles at the click points.



Open-everything topics, Smaller Ideas:

These ideas, while important to developing software in Python, are not as fundamental as the ideas in the preceding Big Ideas section. You can expect to see some but not all of these ideas on the exam.

For Exam 1, students should be able to:

1. Use *eval*. Explain its relationship to *input* and *raw_input*. Explain when it is typically used.

1. Sample problems:

a. How does *eval* relate *input* and *raw_input*?

b. Given a string *s* of the following form:

```
"89.3  90  10.2  30.5"
```

(that is, the string contains numbers), generate a list that contains the numbers in the string *s* (as numbers, not as strings).

2. Use (or at least be able to read) assignment operators: = += *= /= %= //= **=

2. Sample problems:

a. How do = and == differ?

b. What does the statement `x /= 7` do?

3. Use comparison operators: == != > < >= <=

3. Sample problems:

a. What is wrong with the following code snippet?

```
if (x = 7):
```

4. Use Boolean operators: and, or, not

4. Sample problems:

a. What do the following evaluate to?

```
not (8 != 9 and 2 < 10)
not (9 != 9 and 2 < 10)
not (8 != 9 or 2 < 10)
not (9 != 9 or 2 < 10)
not ((not (7 % 3 == 2) && 4 == 7)
```

b. What is wrong with the following code snippet?

```
if (x == 7 or 9):
```

5. Use string and list operators: + *

5. What do the following print?

```
print "hello" + "bob"
print "hello" * 3
print [3, 2, 1] + [1, 2, 3]
```

6. Use *format strings*.

6. What does the following print?

```
a, b, c = 8.3482, 10, "ok"
print "x = %5.2f, y = %4d, z = %s" % (a, b, c)
```

7. Use character functions *chr* and *ord*. Explain what ASCII and Unicode are and how they are different.

7. Sample problems:

a. What is the main difference between ASCII and Unicode? When should you choose ASCII? Unicode?

b. Write a function that, given a string s, returns the sum of the character codes of the characters in the string.

Write a function that, given a list of character codes, returns the string that the list of character codes represents.

8. Explain how *variables are references* and functions are *call by object reference*. Draw *box-and-pointer diagrams*.

8. Draw a box-and-pointer diagram that depicts the effects of the following code snippet:

```
x = 10
y = x
x = 20
z = y
foo(x, y, x, z)
```

9. Do *parallel assignment* (i.e., assignment of tuples)

9. Write a function that, given a list *r* of numbers, returns the sum and the product of the numbers in *r*.

Suppose you have three lists of numbers, called *x*, *y*, and *z*. Write statements that use the above function to print the sum and product of the numbers in all three lists combined.

10. Use the built-in *int*, *float* and *round* functions. Explain the difference between int and long. Explain why we don't just use long for all whole numbers.

10. Explain why the following statement may not compute the average of the numbers in the list *w* correctly (give an example of when it fails). You may assume that the list is not empty.

```
sum(w) / len(w)
```

Then modify the above to make it correct.

What is the difference between *int* and *long*? Why don't we just use *long* for all whole numbers?

11. Define and use *optional* parameters.

11. Define a function foo that has two parameters: a required parameter and an optional parameter whose default value is 'red'.

Write two calls to the above function: one that uses the default value and one that does not do so.

12. Explain what a *documentation string* is and how it is used.

13. Write and use *comments*. Contrast their use with that of documentation strings.

12. (and 13). Why do we put:

a. *Comments* in our programs?

b. *Documentation strings* in our programs?

13. (and 12) For each of the above, what is the notation for them?

14. Explain what a *module* is and how it is used. Use *import* statements, including those with *wildcards*. Explain the danger of using wildcards.

14. Give examples of *import* statements, with wildcards and without them. Why is using wildcards in imports dangerous?

    What library do you import for the *sqrt* function?

15. Use lists operations, including:
    a. *Concatenation*
    b. *Indexing* and *slicing*
    c. The *len* function
    d. List methods, including *append*, *insert*, *count* and *sort*

    Explain what the statements "lists are mutable" but "strings are immutable" mean.

15. Give examples of:
a. The concatenation of two lists
b. Indexing in a list (positive indices and negative indices)
c. Slicing a list
d. Finding the length of a list
e. Appending an element to a list
f. Inserting an element in a list at a given position
g. Counting the number of times a given element appears in a list
h. Sorting a list
i. Mutating a list
j. A failed attempt to mutate a string, and how you "change" strings given that they are immutable

16. *Read from* and *write to files*. *Open* a file for reading or writing. Use *split* and *eval* to read entries on a line of entries. Use of *type* to determine the type of an object.

16. Write code that:
a. Opens a file for reading
b. Opens a file for writing
c. Writes a string to a file, after it has been opened
d. Reads the contents of a file, line by line, after it has been opened
e. Closes a file
f. Suppose that a file has several numbers per line (how many on each line varies from line to line, but there are only numbers in the file, nothing else). Write a function that computes the sum of all the numbers in the file. Bonus: same problem, but the file also contains strings, which should be ignored.

Answers to selected problems:

Closed-book problems:

1.