Comp 150 Exam 1 Overview.

Resources During the Exam

The exam will be closed book, no calculators or computers. You may bring notes on two sides of 8.5x11 inch paper (either both sides of one sheet, or two sheets written on single sides). Write this as you study! I test you on concepts, not memorized rote facts.

Main topics that may be on exam 1

1. The flow of central ideas and technical achievements in the history of computers from the Abacus through today's computers (DH Chapter 1)
2. The idea of an algorithm giving instructions to solve a problem.
   Python: Chapters 1-4, excluding files
3. Types of data: integer, floating point, string, list (NOT long)
4. Variables referring to current values of data
5. Assignment, print, input, and raw_input statements
6. The normal sequential order of operations
7. Looping through a sequence.
8. Basic arithmetic operations +, -, *, / and % (remainder)
9. Range function
10. Sequence operations +, *, referring to a single element with [] using positive and negative indices, slices with the [start:pastEnd] notation, the len function
11. Basic string operations find, join, lower, split, upper
12. Converting types int(<expression>) and str(<expression>)
13. String formatting with width, precision and type d, s, and f
14. The input/calculate/display-output pattern
15. Simple "for each" loops
16. Simple examples of the accumulator looping pattern, modifying results each time through a loop.

How the Python topics get used:

1. Follow fairly arbitrary code using the elements above, and show the results
2. Write a line of code translating an idea into Python, or put a few steps together.

Read the following before looking at either the problems or the solutions!

1. Study first and then look at the sample problems. The sample problems cannot give complete coverage, and if you look at them first, you are likely to study just these points first, and will not get an idea how well you are prepared in general. Look at the list at the top of the page and start by filling in any holes.
2. Do not look at the answers until you have fully studied and tried the problems and gotten *help* getting over rough spots in the problems if you need it! Looking at the answers before this time makes the problems be just a few more displayed examples, rather than an opportunity to actively learn by doing and check out where you are. The *doing* is likely to help you be able to *do* again on a test.

Zelle problems most like the exam (with solutions already on the web in the private area):

Predict the answers in Programming Exercise 1, p. 22 (approximately for part L.)
Predict the answers in Discussion problem 4, p. 48
Discussion p. 71, excluding 1c, d, f
Although I will make exam problems shorter, the ideas of Programming Exercises p. 72: 9, 11
Discussion p. 116: 1-4, excluding 3e, or noting the binary codes for letters are in alphabetical order.

New sample problems start on the next page.

Further Review Problems for Exam 1 (Solutions follow the problems.)

1. Write a Python program that prompts the user for two numbers, reads them in, and prints out the product, labeled.

2. What is printed by the Python code?
```
s = "abcdefg"
print s[2]
print s[3:5]
```

3. Given a string s, write an expression for a string that includes s repeated five times.

4. Given an odd positive integer *n*, write a Python expression that creates a list of all the odd positive numbers up through *n*. If *n* were 7, the list produced would be [1, 3, 5, 7]

5. Write a Python expression for the first half of a string *s*. If s has an odd number of characters, exclude the middle character. For example if *s* were "abcd", the result would be "ab". If *s* were "12345", the result would be "12".

6. Given a positive integer *n*, write Python code that prints out *n* rows, each with one more 'x' on it than the previous row. For instance, if *n* were 4, the following lines would be printed:
```
x
xx
xxx
xxxx
```

7. Suppose you are given a list of words, *wordList*. Write Python code that will write one line for each word. The line for a word contains the word in lower case and in upper case. For example if *wordList* is `['Jose', 'Sue', 'Ivan']`, then your code would print
```
  jose JOSE
  sue SUE
  ivan IVAN
```

8. What is printed by the Python code?
```
for z in [2, 5, 6, 8]:
    print 2*z
```

9. What is printed by the Python code?
```
x = 2.5679
y = 9.0
print "Answers %.3f and %.3f" % (x, y)
```

10. The data crunching needs of the 1890 US census were addressed by what development significant in the history of computers?

11. In modern computers, numbers are stored electronically. Earlier mechanical devices used for numerical calculation had other ways of representing numbers. List two ways.

Answers on the next page

Exam 1 Review Problem Answers

1.
```
  x = input("Enter a number:  ")         # or some such prompt
  y = input("Enter another number:  ")  # or some such prompt
  print "The product is ", x*y           # or some such label
```

2.
```
  c
  de
```

Note the indices. The slice ends before the end index:
```
0123456
abcdefg
```

3. `s*5    # or:   s+s+s+s+s`

4. `range(1, n+1, 2)` # or the end can be n+2 instead of n+1

5. `s[0:len(s)/2]`

A portion of the string is going to be a slice. Then you need expressions for the indices defining the slice.

6.
```
  for i in range(1, n+1):
      print 'x'*i
```

Alternately, use the accumulator pattern:
```
  s= 'x'
  for i in range(n):
      print s
      s = s + 'x'
```
7.
```
   for word in wordList:
       print word.lower(), word.upper()
       #or:  print string.lower(word), string.upper(word)
```

8.
```
  4
  10
  12
  16
```

9. `Answers 2.568 and 9.000`

10. A machine invented by Hollerith was used to tabulate the results, that added up data coded on punched cards.

11. Punched cards as above, the angle of gears or knobs (like in old odometers or the Difference Engine), and another choice mentioned briefly in the introduction is the position of beads in an abacus.