Comp 150 Exam 3 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 mostly want to test you on concepts, not memorized rote facts.

Main topics that may be on exam 3: Python tutorials through 4.3, CGI process, web notes through assember

1. Some Python topics from Exam 2 with more emphasis on writing than last time, including writing `while` loops.
2. Playing computer on nested loops.
3. Formatting with local variables: format parameter **locals()
4. Limited topics from cgi scripting: no HTML, but I could show you the output of my dumpcgi.cgi script from some web page form, as an indication of what is produced by a form, and show you an alternate cgi script to receive the information, and ask you what the script prints.
5. Converting between binary and decimal (both directions!) converting between binary and hexadecimal (the easy way, both directions)
6. Read Pip assembler and play computer, using all instructions except AND and CPZ. Understand the use of the accumulator and symbolic variables and labels for jumps.
7. Write assembler code without tests or jumps (do include LOD, STO, ADD, SUB, DIV, MUL, HLT) *Exclude* JMP, JMZ, AND, NOT, CPZ, CPL
8. Be able to convert individual instructions between assembler and machine language (both ways). I will post a table of op code to assembler mnemonic conversions, like 0010 MUL, so you do not have to put that in your notes.

How the Python topics get used:

1. Follow fairly arbitrary code using the elements above, and show the results. Distinguish exactly what is the output from the sequence of internal steps.
2. Write a few lines of code translating ideas into Python; put several steps together (a bit more than the last exam).

Read the following before looking at either the problems or the solutions! (Same as exam 1)

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.

New sample problems start on the next page.

Further Review Problems for Exam 3 (Solutions follow the problems.)

1. What is printed? Be careful to follow the order of execution, not the order of the text!

```
def foo(x):             #1
    return x + 3        #2

def bar(a, n):          #3
    print(a*n)          #4

print(foo(7))           #5
bar('x', 4)             #6
bar(foo(2), 6)          #7
```

2. What is printed? Remember, a definition does not do anything by itself.

```
def foobar(nums):             #1
    new = list()              #2
    for num in nums:          #3
        new.append(num+2)     #4
    return new                #5

def p(seq):                   #6
    for e in seq:             #7
      print(e, ':', end=' ')  #8

vals = foobar([2, 5, 12])     #9
p(vals)                      #10
```

3. What is printed?

```
x = 15                  #1
y = 7                   #2
while  x > 1:           #3
    print(x, y)         #4
    y -= 1              #5
    x -= y              #6
```

4. What is printed?

```
for s in ['ab', 'c']:   #1
  for n in [1, 3]:      #2
    print(s*n, end=' ') #3
```

5. Suppose a web form initially is tested with its action being my dumpcgi.cgi, and it prints the data on the left below. If the action is changed to the cgi script on the right below, then what plain text will the cgi script print back in the user's browser?

Raw CGI Data:

x: 3
y: 5
name: Joe

```
#!/usr/bin/python

import cgi

print("Content-type: text/plain\n\n")
form = cgi.FieldStorage()
x = form.getfirst("x", '10')
y = form.getfirst("y", '20')
z = form.getfirst("z", '30')

print("x: {} y: {} z: {}".format(x, y, z)
```

6. a. What ends up in the Accumulator?

```
LOD #3
STO X
ADD X
STO Y
SUB #1
MUL Y
HLT
```

Write Pip Assembler for the following. Assume as in the parsing applet that memory locations W, X, Y, Z, T1, T2, T3, and T4 are available:

b. W = 5*X - Y

c. X = 2 - X*Y

7. Convert the PIP machine code to assembler

```
00010100 00000111
00000101 10000000
00001111 00000000
```

8. Convert the PIP Assembler to Machine code

```
LOD 129
MUL #2
HLT
```

9. Play computer with the program below, completing the log at the right, showing the machine state after each instruction. To save time, you may choose to show only those values that change at each line. To be consistent with the simulator display I show columns for both the current and next IP addresses, but you only need to fill in the current IP address (left column) of the instruction just executed. The initial values are shown.

```
IP-->  ACCUM X Y
--  0      0 0 0
 0  2      5 0 0
```

```
Address | Assembler code
      0 |      LOD #5
      2 |      STO X
      4 |      LOD #3
      6 |      STO Y
      8 | LA:  JMZ LB
     10 |      SUB #1
     12 |      STO Y
     14 |      MUL #2
     16 |      ADD X
     18 |      STO X
     20 |      LOD Y
     22 |      JMP LA
     24 | LB:  HLT
```

10. Convert the regular decimal number 29 to binary.

11. Convert the binary numeral 1101101_2 to our normal base 10.

12. Convert the hexadecimal numeral A15E to binary

13. Convert the binary numeral 1010111101_2 to hexadecimal.

14. Complete the definition of the function `prob`.

```
def prob(x, y):
    '''Return x if x > y, and 0 otherwise.'''
```

15. Complete the definition of the function upDown.

```
def upDown(s):
    '''Prints out s in upper and lower case.  Examples:
    upDown('Hello') prints:  HELLOhello
    upDown('3 cheers!') prints:  3 CHEERS!3 cheers!
```

16. Use your upDown function from the previous problem to print the following (write just one possible solution):

```
SAMPLEsample
EXAMexam
```

17. Use a for-loop and your upDown function from above to print the following. Any case mixture is okay in your list:

```
SAMPLEsample
EXAMexam
HIhi
LOlo
```

18. Write interactive code that prompts the user for a word, and then calls upDown with that word, stopping (and printing nothing more) after QUIT is entered. The session could be the following (with user typing shown in ***boldface italics***):

```
Enter a word: Sample
SAMPLEsample
Enter a word: exam
EXAMexam
Enter a word: QUIT
```

19. Complete the definition of the function numbersBelow.

```
def doublesBelow(n, tooBig):
    '''Keep printing and doubling n, as long as the result is less
    than tooBig.  For example, doublesBelow(5, 25) would print
    5 10 20                                                    '''
```

20. Complete the definition of the function clicksBelow. Hint: the `getY` method returns the y-coordinate of a Point.

```
def clicksBelow(win, maxY):
    '''As long as the user clicks the mouse in the GraphWin win
    at a Point with y coordinate less than maxY, draw the Point
    and add it to a list.  Return the list of mouse clicks that
    are low enough.  For example, if maxY is 100, and the user
    clicks the mouse at points with y coordinate 5, 50, 25, and 200,
    return a list of the first three points.  (No prompting required)'''
```

21. Complete the definition below without using any if statements. Use boolean operations only.

```
def xor(A, B):
    '''return the exclusive OR of boolean values A and B: true when
    A or B is true, but not both.  Examples:
        xor(False, True) returns True
        xor(False, False) returns False
        xor(True, True) returns False.    '''
```

Answers on the next page

Exam 3 Review Problem Answers

1. 10 Execution starts at line 5 -- after the definitions!
   xxxx
   30

step by step – does not show the spaces and newlines, not a complete substitute for the final answer!

Line	comment
5	go to foo with parameter 7
1	x is 7
2	return 7+3 = 10
5	print 10
6	go to bar with parameters 'x' and 4
3	a is 'x' and n is 4
4	'x'*4 is xxxx - print it
6	done with line 6
7	inside first - go to foo with parameter 2
1	x is 2
2	return 2+3 = 5
1	go to bar with parameters 5 and 6
3	a is 5 and n is 6
4	5*6 is 30 - print it
7	done with line 7

2. `4 : 7 : 14 :` Execution starts at line 9 -- after the definitions!

step by step – does not show the spaces and newlines, not a complete substitute for the final answer!

Line	num	new	comment
9			go to foobar with parameter [2, 5, 12]
	num	new	local variable headings
1			nums is [2, 5, 12]
2		[]	
3	2		first in list
4		[4]	2+2=4, append to new
3	5		next in list
4		[4,7]	5+2=7, append to new
3	12		next (and last) in list
4		[4,7,14]	12+2=14, append to new
5			return [4, 7, 14]
9			make vals be [4, 7, 14]
10			send [4, 7, 14] to p
	e		local variable heading
6			seq is [4, 7, 14]
7	4		first in list
8			print `4 :` (stay on same line)
7	7		next in list
8			print `7 :` (stay on same line)
7	14		next (and last) in list
8			print `14 :` (stay on same line)

3. 15 7
   9 6
   4 5

step by step – does not show the spaces and newlines, not a complete substitute for the final answer!

line	x	y	comment
1	15		
2		7	
3			15>1 true: loop
4			print 15 7
5	6		7-1 = 6
6	9		15 - 6 = 9
3			9>1 true: loop
4			print 9 6
5	5		6-1 = 5
6	4		9-5 = 4
3			4>1 true: loop
4			print 4 5
5	4		5-1 = 4
6	0		4-4 = 0
3			0>1 false: skip loop

4. `ab ababab c ccc`

The first time through the loop in line 1, s is 'ab', so when the body in lines 2-3 is run, 'ab' is repeated once and three times. The next time through the loop of line 1, s is 'c' and when lines 2-3 are executed, 'c' is repeated once and three times.

step by step – does not show the spaces and newlines, not a complete substitute for the final answer!

line	s	n	comment
1	ab		
2		1	
3			print ab
2		3	
3			print ababab
1	c		
2		1	
3			print c
2		3	
3			print ccc

5. x: 3 y: 5 z: 30 Use form data for x and y. Use the default for z since z is not in the form data.

6. 30

IP-->	ACC	X	Y
--	0	0	0
0	2	0	0
2	4	3	0
4	6	3	0
6	8	3	6
8	10	3	6
10	12	3	6
12	--	3	6

6b.
```
LOD X
MUL 5
SUB Y
STO W
```

c.
```
LOD X
MUL Y
STO T1
LOD #2
SUB T1
STO X
```

7.
```
LOD #7
STO 128
HLT
```

8. 00000100 10000001
   00010010 00000010
   00001111 00000000

9. You will definitely not get anything this long on a real exam!

```
     IP-->  ACCUM   X   Y
----------------------
      -- 0       0   0   0
      0  2       5   0   0
      2  4       5   5   0
      4  6       3   5   0
      6  8       3   5   3
      8  10      3   5   3
      10 12      2   5   3
      12 14      2   5   2
      14 16      4   5   2
      16 18      9   5   2
      18 20      9   9   2
      20 22      2   9   2
      22 8       2   9   2
      8  10      2   9   2
      10 12      1   9   2
      12 14      1   9   1
      14 16      2   9   1
      16 18      11  9   1
      18 20      11  11  1
      20 22      1   11  1
      22 8       1   11  1
      8  10      1   11  1
      10 12      0   11  1
      12 14      0   11  0
      14 16      0   11  0
      16 18      11  11  0
      18 20      11  11  0
      20 22      0   11  0
      22 8       0   11  0
      8  24      0   11  0
      24 --      0   11  0
```

By the way, this roughly corresponds to the Python

```
X = 5
Y = 3
while Y != 0:
   Y = Y - 1
      X = X + 2*Y
# so X ends up as 5 + 2*2 + 1*2 +
0*2 = 11
```

10. 11101

29/2 = 14 R 1
14/2 = 7 R 0
7/2 = 3 R 1
3/2 = 1 R 1
1/2 = 0 R 1 stop when *quotient* is 0, read remainders up

11. 109 = 64 + 32 + 8 + 4 + 1

64 32 16 8 4 2 1 powers
 1 1 0 1 1 0 1 digits

12. 1010000101011110

```
       A     1     5     E
    1010 0001 0101 1110
```

13. 2BD

8421 8421 8421 powers in each group
0010 1011 1101 pad on LEFT to a multiple of 4 bits
 2   B   D
 2 is first sum, already a hex digit
      8+2+1=11 decimal, hex digit B,
           8+4+1 = 13 decimal, or the hex digit D

14.
```
def prob(x, y):
    if x > y:
        return x
    return 0
```

15.
```
def upDown(s):
    print(s.upper()+s.lower())
```

16.
```
upDown('sample')  # any mixture of cases OK
upDown('exam')    # any mixture of cases OK
```

17.
```
# any mixture of cases OK
for word in ['sample', 'exam', 'hi', 'lo']:
    upDown(word)
```

18.
```
word = input('Enter a word: ')
while word != 'QUIT':
    upDown(word)
    word = input('Enter a word: ')
```

19.
```
while n < tooBig:
    print(n, end=' ')  # all on same line
    n = 2*n
```

20.
```
new = list()
pt = win.getMouse()
while pt.getY() < maxY:
    pt.draw(win)
    new.append(pt)
    pt = win.getMouse()
return new
```

21.
```
    #direct translation
    #       A or B  but  not  both
    return (A or B) and not(A and B)
#or
    return A and not B or B and not A #both true
cases

# actually Python has an operator for this:  A ^ B
```