1. Write the complete truth table for the boolean function XYZ + XYZ.
2. Given the following truth table, perform the minterm expansion for the function f. Don’t simplify!
3. x y f(x,y)
4. -------------
5. 0 0 1
6. 0 1 0
7. 1 0 1
8. 1 1 0
9. Design the circuit for the function from the previous problem.
10. Add the following two binary numbers:
11. 1 1 0 1
12. 1 0 1 1 1
13. ----------
14. Multiply the following two binary numbers:
15. 1 0 1 1
16. 1 1 1 0 1
17. ----------
18. Design a truth table, minterm formula, and circuit that will implement a 2-bit greater-than function. Your function should take 4 bits of input, x1, x0, y1 and y0, and produce a true output if and only if the two-bit number x1x0 is greater than the two-bit number y1y0.
19. Write a Hmmm assembly-language program that gets two positive integers from the user, subtracts the second integer from the first, and writes out the square of the result.
20. Write a Hmmm assembly-language program that uses a loop to allow the user to enter one or more integers. When the user enters a zero, the program should stop looping and write out two counts: how many of the numbers were positive, and how many were negative. Don’t count the zero.
21. Write a Hmmm assembly language program that reads a positive integer n from the user and calls a separate function n times. The function should simply write out the number 111. You must use call and jumpr instructions to implement your function.
22. Write a Python function years\_needed that takes three inputs:
    * principal, which is the initial amount of money deposited in an interest-bearing account
    * rate, which is the annual interest rate in decimal form
    * target, which is the final value that the investor wants to reach

The function should use a loop to determine the number of years of compounded annual interest that are needed for the investment to reach or exceed the specified target. *Note:*After each year, the new principal is computed as

principal = principal \* (1 + rate)

1. Write a Python function count\_vowels(s) that counts and returns the number of vowels in a string. Use a loop.
2. Write a Python function stars(n) where n is a positive integer. It should print n lines of stars with 1 star on first line, 2 stars on second line, and so forth. For example, stars(4)should print
3. \*
4. \*\*
5. \*\*\*
6. \*\*\*\*

Use nested loops. You are *not* allowed to use the \* operator (\*). You should use print('\*', end= '') to print a single asterisk at a time while remaining on the same line, and an empty print() to go down to the next line.

1. Write a function all\_perfect(lst) that takes a list of numbers lst and returns True if all of the numbers are 100 and False otherwise. Use a loop.
2. Write a function index\_nearest(n, lst) that takes a number n and a list of numbers lst and returns the *index* of the element in lst whose value is closest to n. Use one or more loops.
3. A two-part problem:
   * First, write a function num\_appearances(substring, s) that returns the number of appearances of the specified substring (which you may assume is of length 2) in the string s. *Hint:* Use an index-based loop.
   * Next, write a function most\_common\_pair(s) that returns the two-character string in s that appears most often as a substring within s. For example,most\_common\_pair('alphabetical') should return al. Ties may be broken arbitrarily. *Hint:* Use num\_appearances as a helper function. In addition, most\_common\_pair will need its own index-based loop.
4. *Optional challenge:* Write a function [longest\_dna(s)](http://codingbat.com/prob/p248425?parent=/home/kzhao@bu.edu) that takes a string s and returns the longest substring that consists only of the characters ‘A’, ‘C’, ‘G’, and ‘T’. For example,
5. longest\_dna('ACCGXXCXXGTTACTGGGCXTTGT')

should return 'GTTACTGGGC'. Ties may be broken arbitrarily.

1. What is printed by the following Python program?
2. def loopy(x, y):
3. print('starting loopy:', x, y)
4. while x < y:
5. x += 1
6. y -= 2
7. print('after loop:', x, y)
8. return x
9. x = 1
10. y = 8
11. y = loopy(x, y)
12. print('after first call:', x, y)
13. loopy(y, x)
14. print('after second call:', x, y)

*Hint:* Use two different tables – one for the global scope and one for loopy – to keep track of the values of the variables.

1. Draw one or more memory diagram like the ones we have used in lecture to illustrate the execution of the following Python program:
2. a = [1, 2, 3, 4]
3. b = a
4. a[3] = 5
5. b[1] = 7
6. print('a is', a)
7. print('b is', b)

In addition, write a few sentences that refer to your diagram(s) and that explain the result of the program.

1. Draw memory diagrams that demonstrate why we get different results from the following two Python programs:
2. ### Program 1 ###
3. def foo(a):
4. a = 2 \* a
5. return
6. b = [1, 2, 3]
7. for i in range(len(b)):
8. foo(b[i])
9. print('b is', b)
10. ### Program 2 ###
11. def bar(lst, i):
12. lst[i] = 2 \* lst[i]
13. return
14. b = [1, 2, 3]
15. for i in range(len(b)):
16. bar(b, i)

print('b is', b)