**COS 120 - Introduction to Computational Problem Solving**

**Lab 03 - Selection Structures**

**Complete the following lab problems by Tuesday evening at 11:55pm and submit your lab03.py file through the Moodle link.**

**L03-01**. Given the code below, what will happen when ***demoCalls()*** is run? (Predict exactly what you expect to see appear in the IDLE environment)  
**from math import \***

**def demoCalls():**

**sqrt(16)**

**print(sqrt(16))**

**x=sqrt(16)**

**print(x)**

**y=sqrt(16) \* sqrt(25)**

**print(y)**

Copy and paste the code above into an IDLE **code** window. Run it with a call to the function. Did you get what you expected? Why or why not?

Now, enter the code **below** exactly as it appears (just cut and paste it) and run **testmySQRT**. Shouldn't you get pretty much the same results as the code above? Why don't you get similar results? Why do "none" and "none" print? What is the meaning of the error message?

Repair mySQRT so that it returns the square root value it estimates. Run testmySQRT again. Now are the results similar to the code above? So what does the return statement do?

**def mySQRT(n,iters):**

**X=1**

**for i in range(iters):**

**X=1/2\*(X+n/X)**

**def testmySQRT():**

**mySQRT(16,100)**

**print(mySQRT(16,100))**

**x= mySQRT(16,100)**

**print(x)**

**y= mySQRT(16,100) \* mySQRT (25,100)**

**print(y)**

Modify your new version of mySQRT again so that it prints rather than returns the square root value. Run testmySQRT again. What happens? Why?

***Why is it better to return the value the function calculates than to directly print it from the function? Which way provides the most flexible/useful abstraction? Repair the mySQRT function so that it returns the estimated square root value.***

**L03-02)** Write a loop that will print exactly three times:

in the loop

in the loop

in the loop

Now rewrite it **5 additional ways** that also produce the same results (think of the different forms of the range statement, and iterations other than 0 to 3 by 1's or 1 to 4 by 1's or any other iterations by ones. Make some use of negative numbers in at least one or two of your implementations, and use no more than 2 list implementations).

**L03-03)** Write a function (call it **schoolDaze**) that will accept an integer as a parameter. The parameter is the age of some person. Output a message telling in what stage of schooling you think the person is, based on the age passed to the function. Implement this using **LINEAR (SEQUENTIAL) IFs, NOT NESTED IF** selection statements. Be sure to handle invalid argument values (< 1 or > 18) in the parameter by outputting an error message and then exiting the function. Use the following categories:

1-3 Nursery

4-5 Preschool

6-11 Elementary

12-13 Middle School

14-18 High School

**L03-04)** Write the function specified in problem L03-03, but this time **use NESTED IF** selection statements to implement the program. Will the user of the function see any difference? Tell your TA two benefits of using nested selections in a program instead of linear selections.

**L03-05)** Write the function specified in problem L03-03, but this time **use** **ELIF** selection statements to implement the program. Will the user of the function see any difference?

**L03-06)** Write a function **def PRS(p1,p2):** that plays paper, rock and scissors and accepts two parameters (the players' moves). Pass it two arguments, and assume they are each one of the string values "paper", "rock" or "scissors". Display an appropriate message (“paper covers rock”, “rock dulls scissors”, “scissors cuts paper”, or “tie”). If either of the strings passed is NOT one of the valid strings, terminate the function with the message "Invalid moves!"

**L03-07)** Write a series of test calls to **PRS** in a function called **testPRS** (no parameters). It should exercise all interesting combinations of test values (hardcode the calls with various values rather than accepting user input. This hardcoded testing is faster and easier than allowing user input.)

**L03-08)** Given the following code, modify it so that gender is required as a third parameter. Female drivers' rates should be calculated at 2/3 of the calculated premium rate if the female driver is less than 21. Male drivers' rates should be double the calculated premium rate if they are less than 21.

**(Starting code on next page)**

def calcAutoPremium(age,numDoors):

if age<21:

if numDoors==2:

print ("High Risk")

premium=2500

else:

print ("Semi-High Risk")

premium=1900

else:

if numDoors==2:

print ("Medium Risk")

premium=1500

else:

print ("Low Risk")

premium=800

monthlyPayment=premium/12.0

return monthlyPayment

**L03-09)** Write a test function **def testAutoPremium():** for the calcAutoPremium function that will exercise all interesting combinations of test values (2 genders, 2 or 4 doors, ages <21, 21 and >21). Remember, errors tend to occur at boundaries. (**Use at least two nested for loops to implement this test function. Three nested for loops could actually be used!**)