Lab 5 (Functions) -- CPS109

This lab gives you practice writing **functions**, as described in Chapter 4 of the course text: *Introduction to Computation and Programming Using Python*, by John Guttag. There are ten programs for you to write. Put all of your programs into a .txt document and submit your single file on D2L

1. Write a function called **helloWorld**(), which prints "Hello World". Next write a program that calls that function once. In the same program, add a loop which calls the function 10 times.
2. Write a function **hello(name)**, which prints "Hello" followed by the value of name. For example, **hello**('Ahmad') would print "Hello Ahmad". Write a program that asks the user for their name, and then uses the function to greet the person.
3. Write a function **hello(firstname, lastname)**, which, for example, if called with hello('John', 'Smith'), would print two lines:

Hello John Smith
Hello Smith, John

   Use the function in a program where you ask the user for first and last name and then call the function.
4. Write a function **repeatPhrase(phrase, n)**, which prints the given phrase n times, alternating between lowercase and uppercase. Recall, that if aString is a string, then a.upper() is the uppercase of that string, and a.lower() is the lower case of the string. For example, repeat('The sky is blue', 5) would print:

the sky is blue
THE SKY IS BLUE
the sky is blue
THE SKY IS BLUE
the sky is blue

   Use the function in a program where you ask the user for the phrase and the value of n.
5. Write a function **timestable(n)**, which prints a multiplication table of size n. For example, timestable(5) would print:

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

   Use the function in a program where you ask the user for n and you print the corresponding table.
6. Write a function **perfectcube(n)**, which returns True if n is a perfect cube and returns False otherwise. Notice that this function is not printing anything, but rather returning True or False. The function must use exhaustive enumeration to check if n is a perfect cube. That means checking if 0**3 is n or 1**3 is n or 2**3 is n, and so on, up to where it is pointless to check further. (Instead of n, you might be checking against abs(n), since for example, -125 is a perfect cube. Use the function in a program where you ask the user for n and you print a statement like "Yes, that is a perfect cube" or "No, that number is not a perfect cube", as appropriate.
7. Write a function **biggestOdd**(), which reads in numbers from the user until the user enters 0. Then the function returns (not prints) the largest odd number that was entered. For example, if the integers were: 10, 9, 7, 12, 2, 5, 15, 100, 90, 60, 0, then the program would return 15 as the largest odd number. Similarly, if the integers were -55, -33, -10, 100, -5, 0, then the program would return -33 as the largest odd number. If there is no odd number, then the function should return 0. Use the function in a program to check that it works.

8. Write a function **biggestBuried(s)**, which the parameter s is a string, and the function returns the largest integer buried in the string. If there is no integer inside the string, then return 0. For example, biggestBuried('abcd51kkk3kk19ghi') would return 51, and biggestBuried('kkk32abce@@-33bb14zzz') would return 33, since the '-' character is treated like any other non digit. Note that a character is a digit if it is greater than or equal to '0' and less than or equal to '9'. Alternatively, you can use string.isdigit() to check if the string is a digit. Use the function in a program to test that it works properly by saying

print(biggestBuried('abcd51kkk3kk19ghi')) answer should be 51
print(biggestBuried('kkk32abce@@-33bb14zzz')) answer should be 33
print(biggestBuried('this15isast22ring-55')) answer should be 55

9. Write a function **squareRoot(x, epsilon)** that uses bisection search to return a number y that is close enough to the square root of x, so that abs(y**2 - x) < epsilon. Try out the function in a program to verify that it works properly.
10. Write a function **decimalToBinary(n)** that converts a positive decimal integer n to a string representing the corresponding binary number. Do the conversion by repeatedly dividing the number n by 2 using integer division, keeping track of the remainders, until the number is reduced to 0. The remainders written in reverse order form the binary number string. The following table (also shown in an earlier lab) illustrates the process.

N	N//2	N%2
5	2	1
2	1	0
1	0	1
0 (done)		Read upwards: **101** is 5
54	27	0
27	13	1
13	6	1
6	3	0
3	1	1
1	0	1
0 (done)		Read upwards: **110110** is 54

Do integer division of 5 by 2, so that N//2 is 2 with remainder 1. Now divide 2 by 2 to get 1 with remainder 0. Next divide 1 by 2 to get 0 with remainder 1. Concatenating the remainders in reverse order makes **101**, which is 5 in binary. Another example: N = 54. The division sequence is 54, 27, 13, 6, 3, 1, 0; and the remainder sequence is 0, 1, 1, 0, 1, 1. The remainders concatenated in reverse order produce: **110110**, which is 54 in binary. Write a program which converts the values from 0 to 9 to binary, using the decimalToBinary(n) function. The output should look like:

0 is the binary of 0
1 is the binary of 1
10 is the binary of 2

11 is the binary of 3
100 is the binary of 4
101 is the binary of 5
110 is the binary of 6
111 is the binary of 7
1000 is the binary of 8
1001 is the binary of 9
