IC152: Assignment 1
Bitwise Operations and Mathematical Calculations

This assignment will introduce us to the Python programming language. We will perform some bit wise operations and basic mathematical calculations using Python.
While the basic operations such as addition, subtraction, division and multiplication are readily available in the Python environment, other more advanced functions such as square-root, logarithm, etc. are provided by the math library.
Such libraries may be included in your program by adding a statement such as “import math” at the beginning of your program. Thereafter, you may use the square-root function by calling the function math.sqrt(x) which will return the square-root of the number x. Likewise, math.log(x, y) returns the logarithm of x to the base y, math.factorial(3) gives the result of 3!, i.e. 3 X 2 X 1. The questions follow. Open Spyder’s iPython console in interactive mode and try to implement the following
questions:
1. When you will learn advanced topics, you will see a lot of them use powers of two to optimize the programs. E.g. While training deep neural networks, a batch size in powers of two is generally used (batch size means number of samples a model trains on in parallel). In graphics, game designers use matrices of size four to transform objects from one frame of reference to another, although the objects and graphics environments have three dimensions. A table of different bitwise and arithmetic operators, their description and their syntax has been given below for their reference.
The bitwise operations have a relation with powers of two, which we will see in the following questions:
a. On your Spyder’s iPython console , print OR of two numbers: 2, 3 (operator for OR is |). What do you observe? Use bin() function to justify your answer.
b. Assign a natural number to a variable with name n. Find if n is odd or not using & operator. The command should print 1 if the number is odd, else 0.
c. Divide 24 by 2 using bit wise operations (Hint: use right shift operator: >>)

2.
a. Print the result of 27/8 on your Spyder’s iPython console .
b. Print the result of 27//8 on your Spyder’s iPython console.
c. Divide 27 by 8 using bit wise operations and reason why you get same
answer as previous question (i.e. 2-b).
d. Using bit wise operations, print 2**10 and 2**20.
Verify your answers by printing
2**10 and 2**20.
e. What will be the value of binary number 1111 in the decimal number system? Write a formula for converting the value of n bit binary number with 1 at all locations to the decimal number system. Verify your answer using the bin() function.

3.
a. Print numbers -5 and 7 on the same line

b. Do you know that 1729 is known as the Hardy-Ramanujan Number? 1729 is the smallest natural number that can be written in the form of a3 + b3 in two different ways, a and b are also natural numbers <= 15.
Guess the values of a and b, and print the two forms of a3 + b3. (Hint: use math.pow(a,3) to find cube of a)

4.
a. Print the resulting evaluations of the two expressions 50-3*10 and (50-3)*10, on the same line.
b. Print the logarithm of 4096 to the base 2.
c. Print the roots of the quadratic equation: 2x**2-7x + 6, on the same line, in ascending order (use the formula, roots =). Verify your $(-b\pm (b2 - 4ac))/2a$ answers.

5.
a. Convert -14.125 to binary number using the formulae discussed in the class. (Hint: sign bit is easy, divide the remaining part by powers of two till you get a decimal number of form 1 + pure decimal number (of form 0.xyzw….) and thus you can derive e-1023 as the power of two. Then, convert the pure decimal number 0.xyzw…. as a sum of powers of ½.
To achieve this: subtract 0.5 and type the first bit (of 52 remaining bits) as 1 if the result is >=0 (else leftmost bit is 0), then subtract 0.25 and see if the result is >=0 and put the second bit as 1, else 0, and so on …). Here is solution for floating point number 3.5 (discussed in class):
print(3.5/21) gives 1.75, hence e-1023 = 1, e = 1024, i.e., 10000000000
print(0.75 - 0.5) gives 0.25, result >= 0, so the 1st bit of remaining 52 bits is 1
print(0.25 - 0.25) gives 0, result >= 0, so the 2nd bit of rem. 51 bits is 1
Since we have reached 0, all the remaining 50 bits are 0 i.e. 0.75 = 0.5 + 0.25 = ½ + (½)2 , hence the first bit of the remaining 52 bit number is 1, and the 2nd bit is also 1, and remaining bits are 0. So the final number is 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 …
We can print the number using: print("11" + "0" * 10 + "11" + "0"*50),
and verify its length by using function: len("11" + "0" * 10 + "11" + "0"*50)
(it should print 64)
b. Convert the binary number created in Q) 5 a again to floating point and verify your answer.

BITWISE OPERATORS

OPERATOR	**NAME**	**DESCRIPTION**	**SYNTAX**
&	Bitwise AND	Results bit 1,if both operand bits are 1;otherwise results bit 0.	x & y
|	Bitwise OR	Results bit 1,if any of the operand bit is 1; otherwise results bit 0.	x | y

~	Bitwise NOT	Inverts individual bits	~x
^	Bitwise XOR	Results bit 0 if both bits are same, otherwise 1.	x ^ y
>>	Bitwise RIGHT SHIFT	The left operand moves towards the right by the number of bits specified by the right operand.	x>>
<<	Bitwise LEFT SHIFT	The left operand moves towards the left by the number of bits specified by the right operand.	x<<

ARITHMETIC OPERATORS

OPERATOR	DESCRIPTION	SYNTAX
+	Addition: adds two operands	x+y
-	Subtraction: subtracts two operands	x-y
*	Multiplication: multiplies two operands	x*y
/	Division (float): divides the first operand by the second	x/y
//	Division (floor): divides the first operand by the second	x//y
%	Modulus: returns the remainder when the first operand is divided by the second	x%y
**	Power: Returns first raised to power second	x**y