Bits and Pieces

Due: Monday, November 26th

PLEASE READ BEFORE STARTING

Objective: With the introduction of different numeral systems and insight into how to convert from one to the other, and vice-versa, you will now gain some firsthand experience on how to define functions that work with binary and decimal numbers. To be able to differentiate between binary and decimal numbers, on top of being able to work with them, binary numbers will be represented as strings with decimal numbers represented as they are. This will be useful for many reasons. The first is that, when creating a function that deals with binary numbers, it will expect a string. If it receives anything other than a string, there may be another portion of code that will notify the user if this is the case. This is known as exception handling, but this assignment will not be covering it.

Required Functions

is_binary(string)

Description: Determine if the passed string meets the necessary criteria for it to be a binary number.

Output: This function should not print anything onto the screen.

Return Value: Return **False** if the given string is not in the correct binary number format. Return **True** otherwise.

binary_to_decimal(binary_string)

Description: Convert the given binary number to a decimal number. During testing, a non-binary number may be passed as an argument. Ensure that you handle this case as appropriate.

Output: This function should not print anything onto the screen.

Return Value: If the passed string is valid, return an integer representing the decimal value of the binary number. Otherwise, return -1.

Suggestions: Recall that each bit has a weight to it that is based on its position in the sequence of numbers. For example, in the binary number 1100_b, the weight of the third bit is 2^2. This may be difficult to implement in practice, but there is a method in computer science known as Horner's Method (https://en.wikipedia.org/wiki/Horner's_method#Application) that employs a very practical formula in obtaining the resulting decimal value of the binary number, and it is as follows:

baseTen = baseTen * 2 + bit[i],

with i representing the position of the bit.

decimal_to_binary(deci)

Description: Convert the passed decimal number *deci* to a binary format.

Output: This function should not print anything onto the screen.

Return Value: A string that is the binary representation of the decimal number.

Suggestions: The algorithm in converting from decimal to binary can be implemented here. In regards to storing the obtained bit at each step, using a list is advisable but not required. Do recall that the bits are obtained in an "opposite" ordering, meaning that the first bit obtained will actually be the last bit in the sequence, and vice-versa. A big hint to approaching this is first creating an empty list, and an empty string. When implementing the algorithm, append each obtained bit to the end of the list. Once obtaining all bits, utilize string concatenation to include each bit in the string sequence, then return that string.

bitwise_add(binary_string1, binary_string2)

Description: Add two binary numbers to obtain the result of the addition. During testing, one or both passed strings may not be valid binary numbers. Ensure that you handle this case as appropriate.

Output: This function should not print anything onto the screen.

Return Value: A string representing the binary number that is the sum of the two passed binary numbers. If one or both passed strings are not valid, return the string “Error”.

Suggestions: You may use the previous two functions, **binary_to_decimal()** and **decimal_to_binary**() to implement this function.

Testing

Total tests: 22

is_binary() will have 4 test cases. All other functions will have 6 test cases.

Notes

Name your program “bits_and_pieces.py”, otherwise the program tester will NOT work.