CS 220 Numeric Representation

ACTIVITY

1. Python uses the built-in bin() function to convert an integer value to binary. Try typing the following statements into the python console and confirm that you understand the results
    a. bin(5)
    b. bin(8)
    c. bin(65)
    d. bin(-5)

    Notice that function returns a string. Also notice that the number of bits is variable.
2. To convert back – you can use the int function to cast this string to a value. Try these at the python console:
    a. int(0b101)
    b. int("0b101")
    c. int("0b101", 2)
    d. 0b101
    e. "0b101"
3. Write a function called binNbits(num, bits) that allows you to specify a number using exactly nbits. If there are not enough bits, the function should raise a value error. For example, your output should look like:

```
>>> binNbits(5,4)
'0b0101'
>>> binNbits(-5,4)
'-0b0101'
>>> binNbits(5,4)
'0b0101'
>>> binNbits(5,8)
'0b00000101'
>>> binNbits(5,2)
Traceback (most recent call last):
  File "<pyshell#77>", line 1, in <module>
    binNbits(5,2)
  File "C:/Users/shughes/Documents/Courses/Coe
S145/Code/Binary.py", line 9, in binNbits
    raise ValueError("Not enough Bits")
ValueError: Not enough Bits
>>> binNbits(-5,4)
'-0b0101'
```

4. In class we demonstrated why using a sign-magnitude representation isn't a good idea for integers (it is inconsistent with addition). Instead we prefer to use two's complement representation. Create a set of functions that allows you convert between binary and decimal representations of signed integer values using two's complement.

   - `signedBin(num, bits)`
     The function should return a string that represents the two's complement representation of num. In the event that the number is not able to be represented with the given number of bits, the function should raise a ValueError.

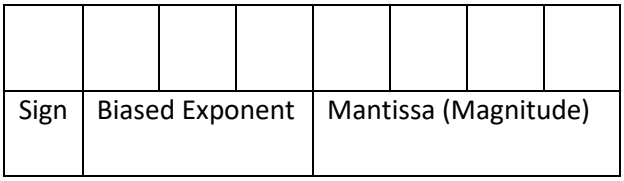
     Example Run:

```
>>> signedBin(-5,4)
'0b1011'
>>> signedBin(5,4)
'0b0101'
>>> signedBin(-1,4)
'0b1111'
>>> signedBin(-8,4)
'0b1000'
>>> signedBin(-9,4)
Traceback (most recent call last):
  File "<pyshell#91>", line 1, in <modu
    signedBin(-9,4)
  File "C:/Users/shughes/Documents/Cour
S145/Code/Binary.py", line 32, in signe
    binNum = binNbits(-num,nbits-1)[2:]
  File "C:/Users/shughes/Documents/Cour
S145/Code/Binary.py", line 9, in binNbi
    raise ValueError("Not enough Bits")
ValueError: Not enough Bits
```

   - `signedInt(binNum)`
     Should allow you to interpret a bit pattern (0bxxxxxx) as a signed integer. If the binNum is not a valid bit pattern, raise a TypeError.

```
>>> signedInt("0b0101")
5
>>> signedInt("0b1111")
-1
>>> signedInt("0b1101")
-3
>>> signedInt("0b1000")
-8
>>> signedInt("1000")
Traceback (most recent call
  File "<pyshell#104>", line
    signedInt("1000")
  File "C:/Users/shughes/Doc
S145/Code/Binary.py", line
    validateBitPattern(binN
  File "C:/Users/shughes/Doc
S145/Code/Binary.py", line
    raise TypeError
TypeError
```

5. Create a function that allows you to perform binary addition. This function should accept two binary numbers (validating that they are actually binary numbers) and return the result of adding them together. (Note – you should attempt to perform the addition in binary – don't convert them to decimal, add them with + and convert back).
6. We also need to be able to represent fractional values (Real numbers) in binary. To do this we rely on the floating point number standard set by the IEEE. In class we described a 1 byte respresentation:



Create the following set of functions that allow you convert between different representations of real numbers.

- `float2bin(num)`
  returns the binary string that represents decimal real value num. If the number is too big to be represented (and exceeds the boundaries of NaN) you should raise a value exception.

  `float2bin(-12.5)` returns the value '11101001'

  - Number is < 0; first bit (Sign) = 1
  - Divide num into whole num and fraction; you already have a function to do the whole numbers – consider creating a new function for the fractional values.
    - 12 = 1100; .5 = .1 --- 12.5 = 1100.1
  - The mantissa assumes the leading 1 – so it can be represented as 1001;
  - Calculate the position of the radix point using the length of the whole number component (subtract 1, to account for the leading 1). In this case, radix is 3; the biased radix is 3+3 = 6 or 110
  - Concatenate these values: 1 110 1001

- `bin2float(binNum)`
  returns the decimal real number value of binary value binNum.

  Notes:

  - `bin2float('01001110')` returns the value 3.75:
    - Sign = 0; biased exponent = 100; mantissa = 1110
    - Bias is 3; the "real" exponent is 1
    - The normalized mantissa is 1.1110
    - The representation is $1.1110 \times 2^1 = 11.1100 = 3.75$

For a grade at the "Competency" level, your program should work with all values in the normalized range

- 0001000 to 01101111
  .25 to 15.5

To meet the Proficiency requirements your program should also address the following:

- Representation of zero, denormalized values, Infiinty and NaN:
- We understand that there is not much practical value in doing representing a float with a single byte of data. In reality, the following parameters are specified as part of the standard.
    - For a 16 bit number (2bytes), 5 bits are used for the exponent and 10 bits are used for the mantissa
    - For a 32 bit number (4 bytes, a.k.a. float), 8 bits are used for the exponent and 23 bits are used for the mantissa
    - For a 64 bit number (8 bytes, a.k.a. double) , 11 bits are used for the exponent and 52 are used for the mantissa

    Note that the bias for the exponent is calculated by $2^{ebits-1}-1$. In the example above, 3 bits are used in the exponent, allowing us to represent the values 0-7 (000 through 111). However, we use the bias to shift those values by the bias ($2^{3-1}-1 = 2^2-1 = 3$).

  Adjust your floating point conversions to allow these representations. That means that for float2bin, you will need to specify the number of bytes to use,