1. Problem 1 - A special "Hello, world!" implementation

1.1 algorithm

Dive the output into three parts, “Hello, ”, “hello, …”, “world!”. The third part is the same whatever the n is. Thus, the algorithm goes:

If n=1,2,3,4 the output needs the first part, then add the rest number of “hello, ”to be the second part;

If n>=5, the first and second part becomes "Lots of hellos, ";

Finally, add the uniform third part to the output string.

Return the output string.

1.2 code

```
def hello(n=1):
    #This function only works for integers input n, or without input;
    if 1<=n<5:
        x="Hello, "
        x=x+"hello, "*(n-1)
    else:
        x="Lots of hellos, "
    x=x+"World!"
return x
```

1.3 demonstration

call code	no input: Hello, World!
print "no input: "+ hello()	1 Hello, World!
print "1 "+ hello(1)	2 Hello, hello, World!
print "2 "+ hello(2)	3 Hello, hello, hello, World!
print "3 "+ hello(3)	4 Hello, hello, hello, hello, World!
print "4 "+ hello(4)	5 Lots of hellos, World!
print "5 "+ hello(5)	88 Lots of hellos, World!
print "88 "+ hello(88)	no input: Hello, World!

2. Problem 2 - The harmonic sequence and the harmonic series

2.1 Algorithm

If input is 0, output that is not valid;

Define two lists, one(a) for 1/I, the other (b) for the accumulation;

Initialization list a and b with the condition i=1, then print I, a and b in proper format, and get I increased by 1;

While I <=the input number,

Add value of 1/I as a new element to list a, and add value (b plus 1/I) as the new element of list b, then print the new I, new added element of a and b; get I increased by 1;

2.2 Code

```
def harmonic(n):#only works for n>1, not n=0
    if n==0:
        print "invalid input"
    else:
        a=[]
        b=[]
        i=1
        a.append(1)
        b.append(a[0])
        print '{0} {1:.6f} {2:.6f}'.format(i, a[i-1], b[i-1])
        i=2
        while i<=n:
            a.append(1.0/i)
            b.append(b[i-2]+a[i-1])
            print '{0} {1:.6f} {2:.6f}'.format(i, a[i-1], b[i-1])
        i+=1
```

2.3 demonstration

Call code	output
harmonic(16)	1 1.000000 1.000000 2 0.500000 1.500000 3 0.333333 1.833333 4 0.250000 2.083333 5 0.200000 2.283333 6 0.166667 2.450000 7 0.142857 2.592857 8 0.125000 2.717857 9 0.111111 2.828968 10 0.100000 2.928968 11 0.090909 3.019877 12 0.083333 3.103211 13 0.076923 3.180134 14 0.071429 3.251562 15 0.066667 3.318229 16 0.062500 3.380729

3. Problem 3 - Binary and Hex code converter

3.1 Algorithm

Frombin(s):

While the calculated digits is less than the length of input string s,

From the last digit of s, add the return value by product of number on the digit and 2 to the power of the order of this digit from the last digit. Move to the left one digit.

Fromhex(s)

While the calculated digits is less than the length of input string s,

From the last digit of s, add the return value by product of number on the digit and 2 to the power of the order of this digit from the last digit. Move to the left one digit.

Need to notice that, if the number on the digit is 0:9, use the number itself; if the number is a:f, use the corresponding number in 10:16, this process is done by the function of fromhexinside(b)), using the relationship of a-f with 97-102 in ASCII code.

tobin(val):

Determine the rightest digit of the output string s as the initialization. Get the remainder of val divided by 2 as the rightest digit of s; let val be the quotient of val divided by 2;

While val is still not 0, add the remainder of val divided by 2 to the left digit of s, make val the quotient of val divided by 2 again.

Return s.

Tohex(val)

Determine the rightest digit of the output string s as the initialization. Get the remainder of val divided by 16 as the rightest digit of s; let val be the quotient of val divided by 16;

While val is still not 0, add the remainder of val divided by 16 to the left digit of s, make val the quotient of val divided by 16 again.

Need to notice that, if the remainder of val divided by 16 is 0:9, use the character converted from the number itself to be the next digit; if the number is 10:15, use the corresponding character in a:f, and this process is done by the function of tohexinside(b)), using the relationship of a-f with 97-102 in ASCII code.

Return s.

3.2 code

```
def frombin(s):
    i=0
    a=0
    le=len(s)
    while i<le:
        a=a+int(s[i])*2**(le-i-1)
        i+=1
    return a
```

```
def fromhex(s):
    #return int((s), base=16)
    i=0
    a=0
    le=len(s)
    while i<le:
        a=a+fromhexinside(s[i])*16**(le-i-1)
        i+=1
    return a


from __builtin__ import str
def tobin(val):
    j=val%2
    s=str(j)
    val=val/2
    while val >0:
        j=val%2
        s=str(j)+s
        val=val/2
    return s

def tohex(val):
    j=val%16
    s=tohexinside(j)
    val=val/16
    while val >0:
        j=val%16
        s=tohexinside(j)+s
        val=val/16
    return s

def tohexinside(n):
        if n<10:
            return str(n)
        else:
            return chr(n+87)

def fromhexinside(b):
    if ord(b)<97:
        return int(b)
    else:
    return ord(b)-87
```

3.3 demonstration

testing algorithm: Use the built-in functions to test the fromhex and frombin functions. once both two are verified, use them to test tobin and tohex functions, by testing if a string converted by fromhex function can be turned back to the input string by tohex, and the testing of tobin works the same way.

test procedure code:

```
def testfromhex(s):
    return    fromhex(s)==int(s,16)
def testfrombin(s):
    return frombin(s) == int(s,2)
def testtobin(s):
    return tobin(frombin(s))==s
def testtohex(s):
    return tohex(fromhex(s))==s
```

test code and outputs

print frombin('1001')	9
print fromhex('1ab')	427
print tobin(5)	101
print tohex(31)	1f
print testfrombin('1011001')	True
print testfromhex('198a8f')	True
print testfrombin('1001111111000001')	True
print testfromhex('1b9a8a86fed')	True
print testtobin('1000111001')	True
print testtohex('1aa9ccc8a8fddd')	True
print testtobin('1001111111000001')	True
print testtohex('1aa9ccc8a8fddd')	True