Chapter 4 -- Number Systems

```
about number systems.
---------------------



here's the decimal number system as an example:
     digits (or symbols) allowed: 0-9
     base (or radix):  10
     the order of the digits is significant

     345 is really
          3 x 100   + 4 x 10    + 5 x 1
          3 x 10**2  + 4 x 10**1  + 5 x 10**0
     3 is the most significant symbol (it carries the most weight)
        5 is the least significant symbol (it carries the least weight)



here's a binary number system:
     digits (symbols) allowed: 0, 1
     base (radix):  2

     each binary digit is called a BIT

     the order of the digits is significant

     numbering of the digits
            msb           lsb
            n-1            0
        where n is the number of digits in the number

      1001 (base 2) is really
           1 x 2**3  + 0 x 2**2  + 0 x 2**1  + 1 x 2**0
           9 (base 10)

      11000 (base 2) is really
           1 x 2**4  +  1 x 2**3  + 0 x 2**2  + 0 x 2**1  + 0 x 2**0
           24 (base 10)




here's an octal number system:
     digits (symbols) allowed: 0-7
     base (radix):  8

     the order of the digits is significant

     345  (base 8) is really
          3 x 8**2  + 4 x 8**1  + 5 x 8**0
            192     +    32     +   5
            229 (base 10)

      1001 (base 8) is really
           1 x 8**3  + 0 x 8**2  + 0 x 8**1  + 1 x 8**0
```

```
              512    +    0      +    0      +    1
           513 (base 10)
```

here's a hexadecimal number system:

```
     digits (symbols) allowed: 0-9, a-f
     base (radix):  16

     the order of the digits is significant

     hex   decimal
     0       0
     1       1
          .
          .
          .
     9       9
     a       10
     b       11
     c       12
     d       13
     e       14
     f       15

      a3 (base 16) is really
           a x 16**1  + 3 x 16**0
            160       +   3
              163 (base 10)
```

given all these examples, here's a set of formulas for the general case.

here's an n-bit number (in weighted positional notation):

```
       S     S    .  .  .   S    S    S
        n-1   n-2            2    1    0
```

given a base b, this is the decimal value

```
          the summation (from i=0 to i=n-1) of S  x  b**i
                                                i
```

TRANSFORMATIONS BETWEEN BASES
----

any base --> decimal
    just use the definition give above.

decimal --> binary
    divide decimal value by 2 (the base) until the value is 0

```
    example:
        36/2 = 18  r=0  <-- lsb
        18/2 =  9  r=0
         9/2 =  4  r=1
```

```
          4/2 =  2  r=0
          2/2 =  1  r=0
          1/2 =  0  r=1  <-- msb

          36 (base 10) == 100100 (base 2)



         14/2 =  7  r=0  <-- lsb
          7/2 =  3  r=1
          3/2 =  1  r=1
          1/2 =  0  r=1  <-- msb

          14 (base 10) == 1110 (base 2)



binary --> octal
    1. group into 3's starting at least significant symbol
       (if the number of bits is not evenly divisible by 3, then
        add 0's at the most significant end)
    2. write 1 octal digit for each group

    example:

        100 010 111  (binary)
         4   2   7   (octal)

         10 101 110  (binary)
         2   5   6   (octal)

binary --> hex
  (just like binary to octal!)
    1. group into 4's starting at least significant symbol
       (if the number of bits is not evenly divisible by 4, then
        add 0's at the most significant end)
    2. write 1 hex digit for each group


    example:

       1001 1110 0111 0000
        9    e    7    0

          1 1111 1010 0011
        1    f    a    3


hex --> binary
     (trivial!)  just write down the 4 bit binary code for
     each hexadecimal digit

     example:

      3    9    c    8
    0011 1001 1100 1000


octal --> binary
     (just like hex to binary!)
     (trivial!)  just write down the 8 bit binary code for
```

```
     each octal digit


hex --> octal
     do it in 2 steps,       hex --> binary --> octal

decimal --> hex
     do it in 2 steps,       decimal --> binary --> hex





on representing nonintegers
---------------------------

what range of values is needed for calculations
     very large:     Avogadro's number 6.022 x 10 ** 23 atoms/mole
                     mass of the earth 5.98 x 10 ** 24 kilograms
                     speed of light    3.0 x 10 ** 8 meters/sec
     very small:     charge on an electron -1.60 x 10 ** (-19)




scientific notation
    a way of representing rational numbers using integers
    (used commonly to represent nonintegers in computers)

                                        exponent
            number =  mantissa x base


        mantissa == fraction == significand
        base == radix
        point is really called a radix point, for a number with
         a decimal base, its called a decimal point.

        all the constants given above are in scientific notation




normalization
  to keep a unique form for every representable noninteger, they
  are kept in NORMALIZED form.  A normalized number will follow the
  following rule:

           1 <=  mantissa  <  base

  In this form, the radix point is always placed one place to
  the right of the first significant symbol (as above).





on precision, accuracy, and significant digits

   These terms are often used incorrectly or ignored.  They are
   important!

   A measurement (in a scientific experiment) implies a certain
   amount of error, depending on equipment used.  Significant
   digits tell about that error.
```

```
For example, a number given as
 3.6  really implies that this number is in the range of
      3.6 +- .05,   which is 3.55 to 3.65
      This is 2 significant digits.
 3.60 really implies that this number is in the range of
      3.6 +- .005,   which is 3.595 to 3.605
      This is 3 significant digits.

So, the number of significant digits given in a number tells
about how accurately the number is known. The larger the number
of significant digits, the better the accuracy.

Computers (or calculators, a more familiar machine) have a fixed
precision.  No matter what accuracy of a number is known, they
give lots of digits in an number.  They ignore how many significant
digits are involved.
For example, if you do the operation 1.2  x  2.2. given that
   each number has 2 significant digits, a correct answer is

               1.2
             x 2.2
             -----
                24
             + 24
             -----
               264 --> 2.64  -->   2.6   or 2.6 +- .05

   a calculator will most likely give an answer of 2.640000000,
   which implies an accuracy much higher than possible.  The
   result given is just the highest precision that the calculator
   has.  It has no knowledge of accuracy -- only precision.
```

```
BINARY FRACTIONS
----------------

       f     f    .  .  .   f    f    f  .  f    f    f  . . .
        n-1   n-2            2    1    0      -1   -2   -3
                                         |
                                         |
                                          binary point

       The decimal value is calculated in the same way as for
       non-fractional numbers,  the exponents are now negative.



  example:
          101.001 (binary)
          1 x 2**2 + 1 x 2**0 + 1 x 2**-3
             4     +    1     +  1/8
                   5  1/8  = 5.125 (decimal)

  2**-1 = .5
  2**-2 = .25
  2**-3 = .125
  2**-4 = .0625   etc.
```

converting decimal to binary fractions

Consider left and right of the decimal point separately.
The stuff to the left can be converted to binary as before.
Use the following algorithm to convert the fraction:

```
fraction  fraction x 2  digit right of point
  .8         1.6             1 <-- most significant (f  )
  .6         1.2             1                        -1
  .2         0.4             0
  .4         0.8             0
  .8  (it must repeat from here!)

        ----
 .8 is  .1100
```

NON-BINARY FRACTIONS
----

same as with binary, only the base changes!

```
f     f    .  .  .   f    f    f  .  f    f    f  . . .
 n-1   n-2            2    1    0     -1   -2   -3
                                   |
                                   |
                                    radix point
```

The decimal value is calculated in the same way as for
non-fractional numbers, the exponents are now negative.

example:

```
101.001 (octal)
1 x 8**2 + 1 x 8**0 + 1 x 8**-3
   64     +    1     +  1/512
         65  1/512  = 65.0019 (approx)

13.a6 (hexadecimal)
1 x 16**1 + 3 x 16**0 + a x 16**-1 + 6 x 16**-2
   16     +    3      +  10/16     +  6/256
        19  166/256 = 19.64 (approx)
```

CONVERSION WITH OTHER BASES
----

another base to decimal:
Just plug into the summation.

```
134 (base 5)
1 x 5**2  +  3 x 5**1  +  4 x 5**0
  25      +    15      +    4
           44 (base 10)
```

decimal to another base:
Keep dividing by the base, same algorithm.

```
100 (base 10)   to base 5

             rem
100/5   20   0
 20/5    4   0
  4/5    0   4

  100 (base 10) = 400 (base 5)
```