

Chapter 2 — Numbering Systems

Introduction

This chapter discusses numbering systems and shows how the computer stores numbers. Manually converting between decimal, binary, and hexadecimal is demonstrated using the remainder and positional methods. Using Python to convert from the three bases, and how to enter integers into your program in the various bases I also shown.

Objectives

Upon completion of this chapter's exercises, you should be able to:

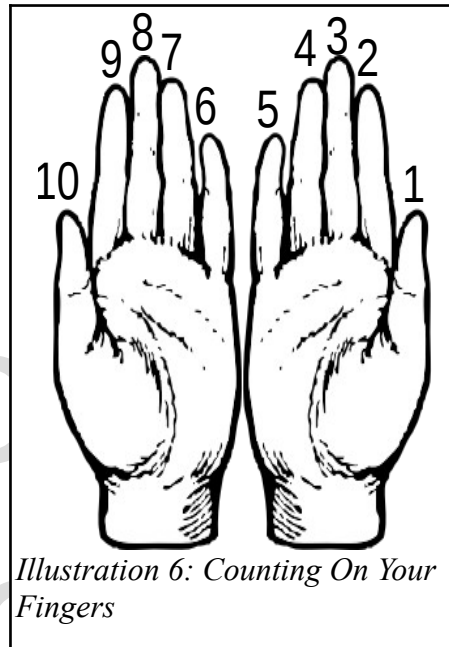
- Recognize decimal, binary, and hexadecimal numbers.
- Define binary and hexadecimal constant numbers in Python using the 0b or 0x prefix.
- Convert numbers from decimal to binary and hexadecimal using the remainder method.
- Calculate the decimal value from a binary or hexadecimal number using the positional method.
- Use Python's built in functions to convert decimal numbers to strings containing binary and hexadecimal.
- Perform the operations of addition and subtraction of binary numbers, using the twos' compliment when necessary.

Prerequisites

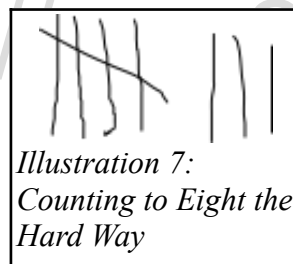
This Chapter requires most of the concepts from Chapter 1; including: basic operations, integers, variables, and the special integer operations.

Ten Fingers

Think about what a number really means...



Counting on our fingers, comes naturally to most of us. Early counting was done making marks with a stick in the dirt or a pencil on a piece of paper. It is easier to make groupings of marks into “scores” like:



Imagine for a moment if humans were born with 8 or 12 fingers, how different would counting have been?

To express numbers greater than the number of fingers we have, people developed the positional notation. It allows us to write large numbers by creating places for tens, hundreds, thousands, and larger powers of 10 (Positional Notation, 2016).

In this discussion we will write the numeric base of a number as a subscript after a number or with a prefix so that we can always be sure of how to interpret the number. Numbers that does not have markings (either before or after) it can be assumed to be base 10.

The Decimal System

The decimal system uses a base of 10 (like our fingers) to express numbers in a positional notation. For example let's look at the number 375_{10} . It represents 3 groups of a hundred, 7 groups of ten, and five singles. We can also represent the number as:

$$375_{10} = 3 \times 100 + 7 \times 10 + 5 \times 1 \quad \text{or} \quad 375_{10} = 3 \times 10^2 + 7 \times 10^1 + 5 \times 10^0$$

In Python we can write this as

```
1| print(3 * 10**2 + 7 * 10 + 5)
```

It is important that we understand that a digit, to the left, is another power of the base. This will become apparent as we move into other bases for numbers. The same principle happens when there is a decimal point, except that the power of 10 becomes negative.

$$3.1415_{10} = 3 \times 10^0 + 1 \times 10^{-1} + 4 \times 10^{-2} + 1 \times 10^{-3} + 5 \times 10^{-4}$$

In Python we could write this as

```
1| print(3 + 1*10**-1 + 4*10**-2 + 10**-3 + 5*10**-4)
3.1415
```

The Binary System

The earliest of computers started with a few on-off switches and as they became more complex they continued to add digits. Current microprocessors have 64 digits, or bits. Instead of using a base N numbering system that would change every time we added digits, early computer scientists represented numbers in base 2 (binary). We still use the base 2 positional notation in the most modern of machines.

In the binary system (base 2) each digit can be either a zero (0) or one (1) and each time we move left the power of two increases.

Power of 2	Decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128

Table 1: Powers of Two

If we number our fingers in the manner shown in the figure (above). We can count any number from 0-1023 on ten fingers. If all fingers are down we are at zero. On your right-hand can we represent 10? On both hands, 100?

A modern computer with a 64 bit processor (64 fingers) can count to a number over 18 quintillion (0 to 18,446,744,073,709,551,615) on one hand. With this example it becomes obvious why binary was selected for early computing.

$$1010_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \equiv 10_{10}$$

In Python we can write that expression as, remembering that anything times zero is zero and anything times one is just itself:

```
1| print(2**3 + 2**1)
10
```

Converting Decimal to Binary (Remainder Method)

To convert from decimal to binary we will use a simple process of integer division. Let's start with an example:

Convert 101_{10} to binary (base 2):

$101 / 2 = 50 \text{ r } 1$

1

$50 / 2 = 25 \text{ r } 0$	01
$25 / 2 = 12 \text{ r } 1$	101
$12 / 2 = 6 \text{ r } 0$	0101
$6 / 2 = 3 \text{ r } 0$	00101
$3 / 2 = 1 \text{ r } 1$	100101
$1 / 2 = 0 \text{ r } 1$	1100101

$101_{10} = 1100101_2$

To convert, using the remainder method, we divide our base 10 number by the base we wish to convert it into (in this case 2). The remainder of that division becomes the next digit as we build the new number. The result of the division is used until it is zero.

In Python we can assign a variable to the decimal value and use the special integer operations of modulo (remainder) and integer division to go through the process for us. In the code you will see that we first find the remainder, find the quotient, print it out, then do it again. We read the answer from the bottom up.

```
1| n = 101
2| r = n % 2
3| n = n // 2
4| print(n,r)
5| r = n % 2
6| n = n // 2
7| print(n,r)
8| r = n % 2
9| n = n // 2
10| print(n,r)
11| r = n % 2
12| n = n // 2
13| print(n,r)
14| r = n % 2
15| n = n // 2
16| print(n,r)
17| r = n % 2
18| n = n // 2
19| print(n,r)
20| r = n % 2
21| n = n // 2
22| print(n,r)
```

50 1

```
25 0
12 1
6 0
3 0
1 1
0 1
```

There is an easier way to do it in Python using something called a while loop. We will learn exactly how they work in a future chapter, but now would be a good time to show a preview. The `while` statement repeats the suite, of three lines of code, while `n` is not zero. We will see the `while` statement in future chapters, but it makes the program much smaller. The example below will work for virtually any positive integer, just change the first line.

```
1| n = 101
2| while(n):
3|     r = n % 2
4|     n = n // 2
5|     print(n,r)
```

```
50 1
25 0
12 1
6 0
3 0
1 1
0 1
```

Another Example:

Convert 364 to binary.

$364 / 2 = 182 \text{ r } 0$	0
$182 / 2 = 91 \text{ r } 0$	00
$91 / 2 = 45 \text{ r } 1$	100
$45 / 2 = 22 \text{ r } 1$	1100
$22 / 2 = 11 \text{ r } 0$	01100
$11 / 2 = 5 \text{ r } 1$	101100
$5 / 2 = 2 \text{ r } 1$	1101100
$2 / 2 = 1 \text{ r } 0$	01101100
$1 / 2 = 0 \text{ r } 1$	101101100

And in Python we can do it, like:

```
1| n = 364
2| while(n):
3|     r = n % 2
4|     n = n // 2
5|     print(n,r)
```

```
182 0
91 0
45 1
22 1
11 0
5 1
2 1
1 0
0 1
```

Binary to Decimal (Positional Method)

To convert a binary number back to a decimal number we need to know our powers of 2 and just add them together.

If the binary number 101101101 actually represents

$1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$, then we should be able to add up the powers of two that are multiplied by 1 and ignore the powers multiplied by zero. So,

$$101101101_2 = 1 \times 2^8 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 256 + 64 + 32 + 8 + 4 + 1 = 365_{10}$$

Examples:

Convert 100101011_2 to decimal:

1	0	0	1	0	1	0	1	1
256			32		8		2	1

299_{10}

In Python we could write

```
1| print(2**8 + 2**5 + 2**3 + 2 + 1)
```

299

Binary in Python

In Python, you may directly include binary integers exactly the same as you would a decimal number. You just need to add a "0b" to the front of the number so that Python knows it is not a decimal number. In the example below, you see the difference adding the "0b" prefix does. Print usually outputs a number in decimal.

```
1| print(0b110101)
2| print(110101)

53
110101
```

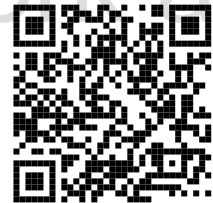
Python will convert a decimal number into a string of binary digits, using the `bin()` function. You will be able to print it out, but the result is no longer a number and can not be directly used in mathematics.

`bin(expression)`

Function

The bin function will return a string of binary digits prefixed with '0b' representing the number passed.

<https://docs.python.org/3.7/library/functions.html#bin>
<http://bit.ly/2SI6d9Q>



```
1| print(bin(42))

0b101010
```

Hexadecimal

A base 16 number is also called a hexadecimal number. Each digit represents a value from 0 to 15. Because we do not have single symbols for 10-15, we use the letters A-F to represent them.



Decimal	Binary	Hexadecimal	Decimal	Binary	Hexadecimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Table 2: Hexadecimal Digits

Hexadecimal is used as a briefer way to represent binary. Each digit in “Hex” represents 4 digits in binary. Hexadecimal numbers are often written with a “#” or “0x” in front of them.

Hexadecimal to Binary and Binary to Hexadecimal

Because 16 is an even power of 2. We can group out binary number into 4s and then convert those blocks into Hexadecimal.

Examples:

Convert 10101010110011_2 to hexadecimal:

$10\ 1010\ 1011\ 0011 = 0x2AB3$

Convert $0xFE9$ to binary:

$F = 1111\ E = 1110\ 9 = 1001$, so $0xFE9 = 111111011111001_2$

Converting Decimal to Hexadecimal (Remainder Method)

The process for converting a decimal number to Hexadecimal is the same as used to convert it to a binary number except that we divide by 16: If the remainder is 10-15 we use the letters A-F in the result.

Example:

Convert 63987 to Hexadecimal:

$63987 / 16 = 3999 \text{ r } 3$	<u>3</u>
$3999 / 16 = 249 \text{ r } 15$	<u>F</u> 3
$249 / 16 = 15 \text{ r } 9$	<u>9</u> F3
$15 / 16 = 0 \text{ r } 15$	<u>F</u> 9F3

In Python we can convert to base 16 using code just like we did to convert to binary. We find the remainder from 26 and divide by 26. Reading the remainders from bottom to top.

```
1| n=63987
2| r = n % 16
3| n = n //16
4| print(n,r)
5| r = n % 16
6| n = n //16
7| print(n,r)
8| r = n % 16
9| n = n //16
10| print(n,r)
11| r = n % 16
12| n = n //16
13| print(n,r)
```

```
3999 3
249 15
15 9
0 15
```

This can also be written using the while loop to simplify.

```
1| n=63987
2| while(n):
3|     r = n % 16
4|     n = n //16
5|     print(n,r)
```

Hexadecimal to Decimal (Positional Method)

We can also convert from Hexadecimal to decimal by using the powers of 16.

Power of 16	Decimal	Power of 16	Decimal
16^0	1	16^4	65,536
16^1	16	16^5	1,048,576
16^2	256	16^6	16,777,216
16^3	4,096	16^7	268,435,456

Table 3: Powers of 16

Hexadecimal in Python

In Python, you may directly include base 16 integers exactly the same as you would a decimal number. You just need to add a "0x" to the front of the number so that Python knows it is not a string or a decimal number. You will use the symbols 0-9 for 0-9 and a-f for 10-15. The "0x" prefix is not case-sensitive and the letters a-f may also be in upper and lower case. In the example below, you see the difference adding the "0x" prefix does. Print will always out a number as a decimal.

```
1| print(0X42, 0Xff, 0xfffa, 0XABBACAB)
```

```
66 255 65530 180071595
```

Python will convert a decimal number into a hexadecimal string, using the `hex()` function. You will be able to print it out, but the result is no longer a number and can not be directly used in mathematics.

`hex(expression)`

Function

The hex function will return a string of hexadecimal digits prefixed with '0x' representing the number passed.

<https://docs.python.org/3.7/library/functions.html#hex>
<http://bit.ly/2EMo6dJ>



```
1| print(hex(42))
```

```
0x2a
```

Binary Addition

We add binary numbers the same way that we add decimal numbers. Write them aligned positionally, start on the right and add each column, If the column exceeds, carry the extra to the next column.

$0 + 0 = 0$
 $0 + 1 = 1$
 $1 + 0 = 1$
 $1 + 1 = 10$

Text 1: Binary Addition

Example:

Add 101_2 to 1001_2 .

$$\begin{array}{r} 101 \\ 1001 \\ \hline \end{array}$$

Add ones' column $1+1=10$, carry the 1 to the twos' column.

$$\begin{array}{r} 1 \\ 101 \\ 1001 \\ \hline 0 \end{array}$$

Add twos' column $1+0+0=1$.

$$\begin{array}{r} 1 \\ 101 \\ 1001 \\ \hline 10 \end{array}$$

Add fours' column $1+0=1$.

$$\begin{array}{r} 1 \\ 101 \\ 1001 \\ \hline \end{array}$$

110

Add sixteens' column $1+0=1$.

```
  1
101
1001
1
1110
```

Twos' Complement — Negative Binary Numbers

In our discussion so far we have not talked about how computers handle negative numbers. All of the numbers we have been working with are positive counting numbers. Computers typically store negative numbers as a “Twos' complement” number.

To calculate a twos' complement of a positive number (to negate a number) we need to know how many bits (or digits) that the processor used to do binary mathematics. To do the conversion: 1) write the binary number out with zeros padded on the left to the proper width, 2) subtract one, 3) change all 0s to 1s and 1s to 0s.

If the computer word size is N bits then numbers from the range of $-2^{N-1} \leq X \leq 2^{N-1} - 1$ can be expressed as twos' complement numbers.

Example:

Assuming a word size of 8 bits we can calculate the twos' complement of the binary number 10111:

```
 10111
00010111 — pad on left to word size
00010110 — subtract one
11101001 — change all 1 to 0 and 0 to 1
```

Binary	Decimal	Binary	Decimal
1000	-8	0000	0
1001	-7	0001	1
1010	-6	0010	2
1011	-5	0011	3
1100	-4	0100	4
1101	-3	0101	5
1110	-2	0110	6
1111	-1	0111	7

Table 4: Twos' Compliment Numbers

Subtraction of Binary Numbers (Using Twos' Compliment)

The easiest method for subtracting two binary numbers is to convert one of them to the twos' compliment and add them.

Example:

Given a 4 bit processor subtract 3 from 7:

```
0111 — decimal 7
1101 — twos' compliment of 3
-----
10100 — add
0100 — ignore the last carry bit — answer is decimal 4
```

Summary

Summary goes here

Important Terms

- addition
- binary
- bit

- compliment
- decimal
- hexadecimal
- positional
- remainder
- sixteen
- subtraction
- ten
- two

Exercises

1. Convert the decimal number 123 to binary using the remainder method. Check your answer with a short Python program to print the binary number.
2. Convert the decimal number 983 to binary using the remainder method. Check your answer with a short Python program to print the binary number.
3. Convert the binary number 10111010 to decimal using the powers of two for the position of each one. Check your answer with a short Python program to print the decimal number.
4. Convert the binary number 1001101 to decimal using the powers of two for the position of each one. Check your answer with a short Python program to print the decimal number.
5. Convert the binary number 10100011 to hexadecimal by grouping the binary digits using. Check your answer with a short Python program to print the hexadecimal number.
6. Convert the binary number 1000111110 to hexadecimal by grouping the binary digits using. Check your answer with a short Python program to print the hexadecimal number.
7. Convert the hexadecimal number 0x92 to binary and then to decimal. Check your answer with a short Python program to print the number as binary and hexadecimal.
8. Convert the hexadecimal number 0x12a7 to binary and then to decimal. Check your answer with a short Python program to print the number as binary and hexadecimal.
9. Perform binary addition of:

$$\begin{array}{r} 101101 \\ + 1001 \\ \hline \end{array}$$

9. Perform binary addition of:

$$\begin{array}{r} 10101101 \\ + 1110 \\ \hline \end{array}$$

10. Calculate the 8 bit twos-compliment of 7, 22, and 89.

11. Calculate the 16 bit twos-compliment of 592, 1024, 30000.

12. Perform binary subtraction by calculating the twos-compliment of the subtrahend and adding the two 8 bit numbers:

```
  11001100
-00011010
            
```

13. Perform binary subtraction by calculating the twos-compliment of the subtrahend and adding the two 16 bit numbers:

```
  0011 1100 0000 0011
-0011 0100 1111 1110
                      
```

Word Search

s c o m p l i m e n t t b
d e c i m a l . b b k . i
n g g w o . r p z a j x n
a q t r e m a i n d e r a
b x n t e n g f k . k l r
h m p a d d i t i o n k y
s u b t r a c t i o n t e
q z e j s i x t e e n w v
h e x a d e c i m a l o h
h s s k o b j r . c a l d
b i t o t d a y b k x . t
p o s i t i o n a l k c p
l a i z a g . . a s i l w

addition, binary, bit, compliment, decimal, hexadecimal, positional, remainder, sixteen, subtraction, ten, two

References

Positional Notation. (2016). Wikipedia. Retrieved 2016-04-02 from
https://en.wikipedia.org/wiki/Positional_notation

Hand Images (Right Hand and Left Hand). (2007). By Johnny Automatic. Retrieved 2016-04-03 from
<https://openclipart.org/detail/7580/right-hand> and <https://openclipart.org/detail/7531/left-hand>



Free eBook Edition

Please support this work at
<http://syw2l.org>

Free eBook Edition

