Basics

- Number Systems
- Python Number Converter
- Storage Units
- Gates

Number Systems

Number Types

The number system holds significance in terms of writing and expressing code to a computer, typically in a programming language. Note that we (as humans) do not use hex or binary numbers that much outside of the computer science domain. For example, we don't walk into a supermarket and read prices in binary such as `$0x10` :)

Often times in programming, we need to express numbers more quickly, and we might say int `x = 0x10000000` to quickly indicate 32-bit value with `bit31` set to 1. Notation `x = 0x10000000` is easier than writing `x = 268435456` which would be more cryptic for a programmer to realize the significance of because the reader of the programming code will not be able to quickly realize that it is specifically setting `bit31` to value of `1`.

Decimal

Typical numbers we are familiar with are decimals which are technically "base 10" numbers. So an ordinary number that we may be aware of such as 123 can be written as 123_{10}.

The number 123 could also be written as:

$1*10^2 + 2*10^1 + 3*10^0$ which is equal to $100 + 20 + 3 = 123_{10}$

Binary

Binary numbers are always 1s and 0s only. Similar to decimal numbers, binary numbers increase in powers of 2, rather than powers of 10. Binary numbers are written by with the "0b" notation, such as 0b1100

For example, binary 101 or `0b101` can be written as:

$1*2^2 + 0*2^1 + 1*2^0$ which is equal to $4 + 0 + 1 = 5_{10}$

Hex

One digit of a hex number can count from 0-15, but since we have to represent the hex number using a single character, the numbers 0-9 are usual numbers, and the numbers 10-15 are represented by A, B, C, D, E, F

Where decimal is a power of 10, and binary is power of 2, hex numbers are powers of 16. Hex numbers are written with the "0x" notation, such as 0x10.

For example, hex `0x12` can be written as:

$1*16^1 + 2*16^0$ which is equal to $16 + 2 = 18_{10}$

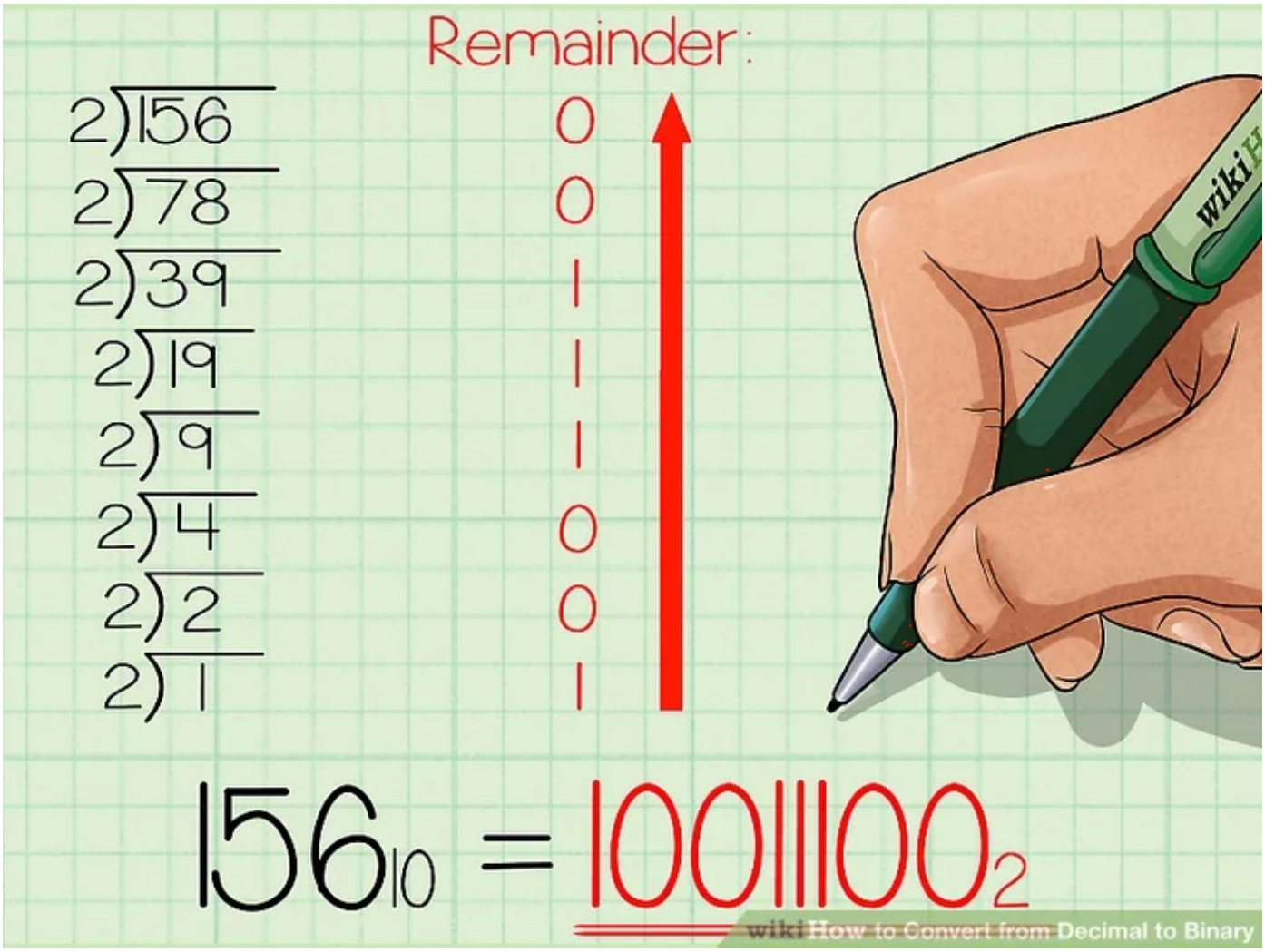
As another example, hex `0xC5` can be written as:

$12*16^1 + 5*16^0$ which is equal to $192 + 5 = 197_{10}$

Exercises

Decimal to Binary

Decimal (base 10) numbers can be converted in a couple of different ways as described here. One of the methods is to continue dividing by 2 and note down the remainder as described in the image below. The article above also describes a potentially faster method of conversion so be sure to read it!



Please try converting the following to binary:

1. 125
2. 255
3. 500

Decimal to Hex

Decimal to hex is similar to Decimal to Binary except that we are dealing with powers of 16 rather than powers of 2.

My favorite method of conversion from decimal to hex is to first convert the number to binary. For example, let's start with a large number such as 23912. We can use the Decimal to Binary method to convert this first to binary:

- 23912_{10}

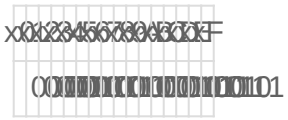
- `0b101110101101000`
- Split it up to nibbles:
  - `0b101 1101 0110 1000`
- Then use the lookup table listed in Hex to Binary:
  - `0x5D68`

Please try converting the following to hex:

1. 125
2. 255
3. 500

Hex to Binary

The following table can be utilized to convert hex to binary very instantly:



First row is HEX, and the second row is binary. For whatever hex number we wish to convert, we simply locate its equivalent in binary. For instance, if we wish to convert `0x5` to binary, it is `0b0101`, and `0xA5` would be `0b1010.0101` as you can convert one "nibble" (4-bits) at a time.

Let's take another example to convert `0x1BF` to binary; simply break it down by "nibbles":

- `0x1` --> `0b0001`
- `0xB` --> `0b1011`
- `0xF` --> `0x1111`
- Answer: `0b0001 1011 1111`

Please try converting the following to binary:

1. 0x55
2. 0x125
3. 0x40000000

Hex to Decimal

For Hex to Binary, we used a lookup table as a "cheat code" :). For Hex to decimal, it would be easier to re-write the numbers as powers of 16. For example, to convert `0x1BF` to decimal, we can break it down

to:

- `0x1` --> `1 * 16`^{2} --> `256`
- `0xB` --> `11 * 16`^{1} --> `176`
- `0xF` --> `15 * 16`^{0} --> `15`
- `256+176+15 = 447`

Please try converting the following to decimal:

1. 0x55
2. 0x125
3. 0x40000000

Python Number Converter

Generally speaking, practiced skill cannot be easily forgotten. It is far better to go through the process and practice converting a number, rather than to memorize the process.

Before we get started, have a look at the Tools Page to get started with a Python Interpreter we could use for this exercise.

Number to Printable Hex

```
def nibble_to_ascii(nibble: int) -> str:
  """
  This is a comment
  Input: Nibble (4-bits)
  Output: Single character HEX as a string
  Example: Input = 10, Output = 'A'
  Example: Input = 8,  Output = '8'
  """
  table = ['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
  return table[nibble]
def to_hex(number: int) -> str:
    """
    This is a comment
    Input: Number (integer)
    Output: String
    Example: Input = 43605, Output = "0xAA55"
    """
    answer = ""

    # Forever loop
    while True:
        # Integer divide using the // operator
```

```
        quotient = number // 16
        # Get the remainder using the % operator
        remainder = number % 16

        # Accumulate result
        answer = nibble_to_ascii(remainder) + answer
        # Set the number we need to use for next time
        number = quotient

        # We break the "loop" when division turns to zero
        if (quotient == 0):
            break

    return "0x" + answer
print(to_hex(123456789))
print(to_hex(0b1010101))print(to_hex(0xDEADBEEF))
```

Exercise

Write a function `to_binary()` that takes a number, and returns the string equivalent version of the number in binary. You can borrow the template above of `to_hex()` function and most of the logic might be similar except that we would be dividing number by 2 rather than 16.

Storage Units

I fear that most of the technical articles on the Internet misinterpret some of the common storage units.

THERE'S BEEN A LOT OF CONFUSION OVER 1024 vs 1000, KBYTE vs KBIT, AND THE CAPITALIZATION FOR EACH.

HERE, AT LAST, IS A SINGLE, DEFINITIVE STANDARD:

SYMBOL	NAME	SIZE	NOTES
kB	KILOBYTE	1024 BYTES OR 1000 BYTES	1000 BYTES DURING LEAP YEARS, 1024 OTHERWISE
KB	KELLY-BOOTLE STANDARD UNIT	1012 BYTES	COMPROMISE BETWEEN 1000 AND 1024 BYTES
KiB	IMAGINARY KILOBYTE	$1024\sqrt{-1}$ BYTES	USED IN QUANTUM COMPUTING
kb	INTEL KILOBYTE	1023.937528 BYTES	CALCULATED ON PENTIUM F.P.U.
Kb	DRIVEMAKER'S KILOBYTE	CURRENTLY 908 BYTES	SHRINKS BY 4 BYTES EACH YEAR FOR MARKETING REASONS
KBa	BAKER'S KILOBYTE	1152 BYTES	9 BITS TO THE BYTE SINCE YOU'RE SUCH A GOOD CUSTOMER

This article does a good job at clearly providing the relevant information:

> A **kilobyte** is made up of either 1,000 or 1,024 bytes. This distinction can be a little tricky and has to do with the difference between binary math (which computers rely on) and base-10 math (which most humans use in daily life). In practical terms, both

> definitions of kilobyte are used. In some cases, a distinction will be made between a kilobyte (1,000 bytes) and a kibibyte (1,024 bytes), though this is less common.

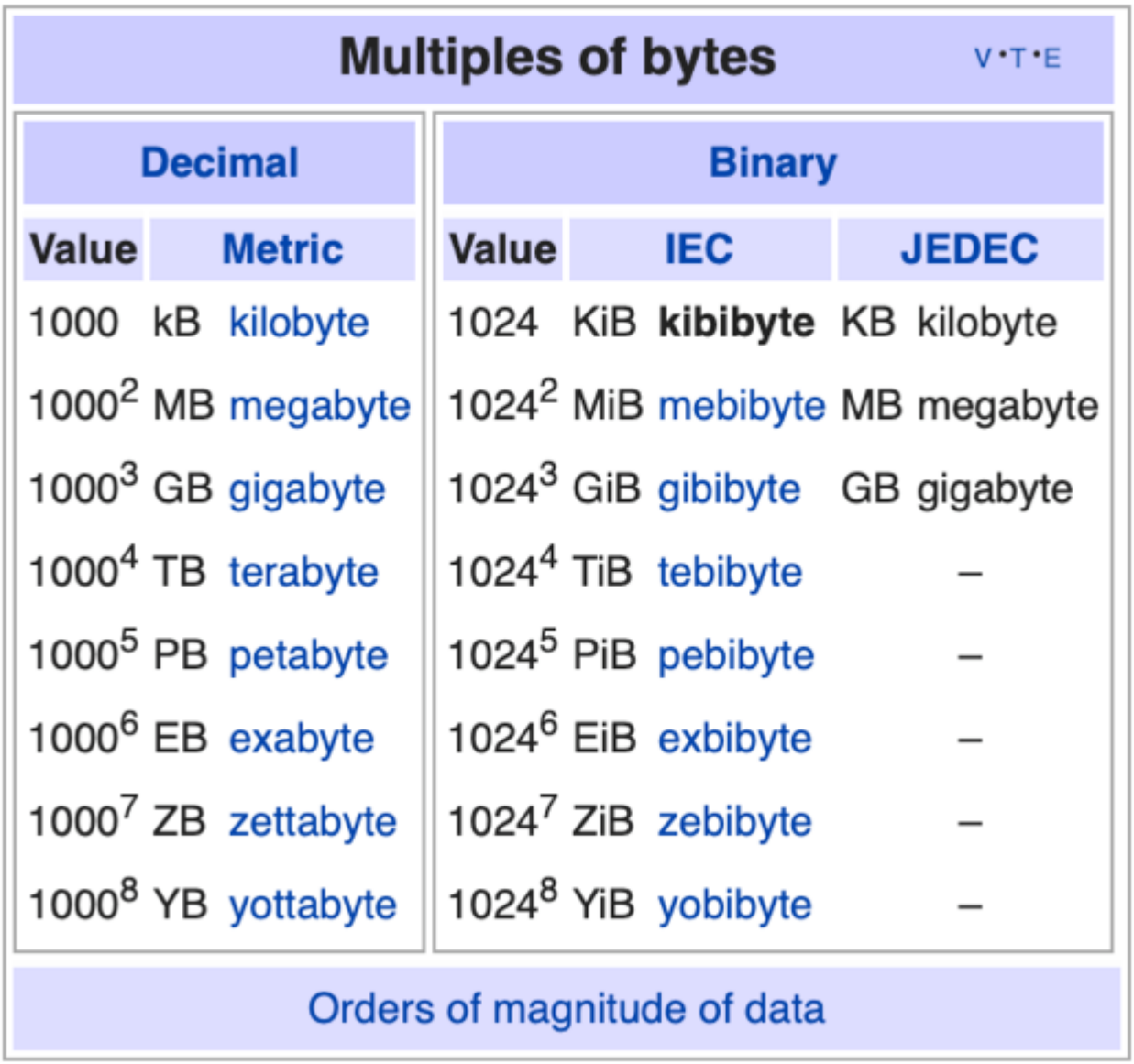
The Real Story

Apart from the funny picture above (Baker's Kilobyte?), the real story can be uncovered by referencing the picture below. Thanks to **this original article** that does a great job at providing the valuable information.

Decimal Prefix (SI)	Value	Value (1000)	Binary Prefix (IEC)	Value	Value (1024)
kilo (k)	10^3	1000	kibi (ki)	2^{10}	1024
mega (M)	10^6	1000^2	mebi (Mi)	2^{20}	1024^2
giga (G)	10^9	1000^3	gibi (Gi)	2^{30}	1024^3
tera (T)	10^{12}	1000^4	tebi (Ti)	2^{40}	1024^4
peta (P)	10^{15}	1000^5	pebi (Pi)	2^{50}	1024^5
exa (E)	10^{18}	1000^6	exbi (Ei)	2^{60}	1024^6
zetta (Z)	10^{21}	1000^7	zebi (Zi)	2^{70}	1024^7
yotta (Y)	10^{24}	1000^8	yobi (Yi)	2^{80}	1024^8

So while most people might misinterpret "kilo" as 1024 when it comes to storage units, the right way is thus "kibibytes". It would be an interesting conversation to discuss kibibytes as most people may not be aware, and this would make you look incredibly smart (and correct) :)

Here is another great image for reference:



Multiples of bytes V·T·E

Decimal			Binary				
Value	Metric		Value	IEC		JEDEC	
1000	kB	kilobyte	1024	KiB	**kibibyte**	KB	kilobyte
1000^2	MB	megabyte	1024^2	MiB	mebibyte	MB	megabyte
1000^3	GB	gigabyte	1024^3	GiB	gibibyte	GB	gigabyte
1000^4	TB	terabyte	1024^4	TiB	tebibyte	–	
1000^5	PB	petabyte	1024^5	PiB	pebibyte	–	
1000^6	EB	exabyte	1024^6	EiB	exbibyte	–	
1000^7	ZB	zettabyte	1024^7	ZiB	zebibyte	–	
1000^8	YB	yottabyte	1024^8	YiB	yobibyte	–	

Orders of magnitude of data

Based on the image above, the following should be used using capital letter first, then lowercase i and then finally capital B for bytes.

- KiB
- MiB
- GiB
- TiB
- PiB
- etc.

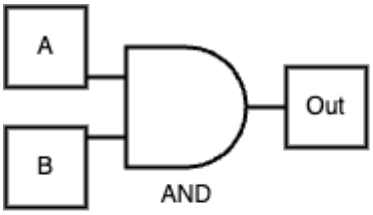
Reference Articles

- https://danielmiessler.com/blog/the-difference-between-kilobytes-and-kibibytes/
- https://study.com/learn/lesson/data-storage-units-kb-mb-gb-tb.html
- https://ozanerhansha.medium.com/kilobytes-vs-kibibytes-d77eb2ff6c2a

Gates

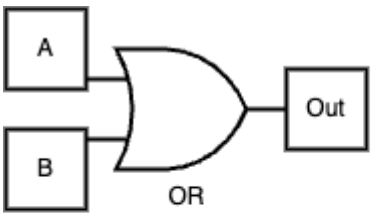
Diagram link

AND



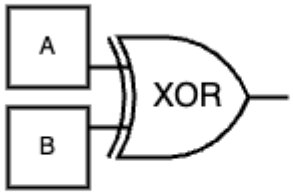
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

OR



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

XOR



Equation:
(A * -B) + (-A * B)
(-A + -B) * (A + -B)

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0