Data 301 Data Analytics
Data Representation

Dr. Irene Vrbik

University of British Columbia Okanagan
irene.vrbik@ubc.ca

Term 1, 2019

Computer Terminology

There is a tremendous amount of terminology related to technology.

Using terminology precisely and correctly demonstrates understanding of a domain and simplifies communication.

We will introduce terminology as needed.

Basic Computer Terminology

- A *computer* is a device that can be programmed to solve problems.
- *Harware* includes the physical components of computer
  - (eg. central processing unit, monitor, keyboard, computer data storage, graphic card, speakers).
- *Software* programs that a computer follows to perform functions
  - (eg. operating system, internet browser).

Basic Computer Terminology

- *Memory* is a device which allows the computer to store data either temporarily (lost when computer reboots, eg. RAM) or permanently (data is preserved even if power is lost, eg. hard drive).
- There are many different technologies for storing data with varying performance.
- Some live inside your computer while others are portable and can be used on difference devices (e.g. USB drives).

"The Cloud"

"The Cloud" is not part of your computer but rather a network of distributed computers on the Internet that provides storage, applications, and services for your computer.

Examples:

- *Dropbox* is a cloud service that allows you to store your files on machines distributed on the Internet. Automatically synchronizes any files in folder with all your machines.
- *iCloud* is an Apple service that stores and synchronizes your data, music, apps, and other content across Apple devices.
- Google Docs you can write, edit, and collaborate wherever you are. For free.

What is data?

> Data: *information, especially facts or numbers, collected to be examined and considered and used to help decision-making, or information in an electronic form that can be stored and used by a computer.*
>
> – Cambridge Dictionary

However, is can be argued (see this article for example) that

$$\text{data} \neq \text{information}.$$

In addition, on might refer to *raw* data as a collection of number/facts that don't have meaning until it has been analyzed or has been given meaning.

How is data measured?

- Computers represent data digitally meaning that data is represented using discrete units called as bits (**B**inary Dig**its**).
- The real-world is *analog* where the information is encoded on a continuous signal (spectrum of values, ie. infinite sounds/colours).

"Like with the artist's abstract composition, the trick is to take all of the real-world sound, picture, number, etc. data that we want in the computer and convert it into the kind of data that can be represented in (on/off) switches."

University of Rhode Island

How is data measured?

Data size is measured in bytes.

- A bit is either a 0 or a 1.
- A *byte* contains 8 *bits* (*B*inary Dig*its*)
- A *nibble* contains 4 *bits* (*B*inary Dig*its*)

Larger units:

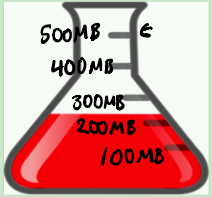
- kilobyte (KB) = 1000 bytes
- megabyte (MB) = 10^6 bytes (or 1000 KB)
- gigabyte (GB) = 10^9 bytes (or 1000 MB)
- terabyte (TB) = 10^{12} bytes (or 1000 GB)
- petabyte (PB) = 10^{15} bytes (or 1000 TB)
- exabyte (EB) = 10^{18} bytes (or 1000 PB)
- zettabyte (ZB) = 10^{21} bytes (or 1000 EB)

Memory/Data Size

Memory size is a measure of memory storage capacity in bytes. It represents the *maximum* capacity of data in the device.

Example 1

Given this flask, assume the red liquid is data and each mark represents 100 MB of data. Select a true statement.



A) Memory size is 200 MB.
B) Flask can hold 0.5 GB of data.
C) Data size is about 200 KB.
D) Data size of 1000 KB would "overflow device".
E) All of the above statements are false.

Solution

Given this flask, assume the red liquid is data and each mark represents 100 MB of data. Select a true statement.

A) Memory size is 200 MB. ~500MB

B) Flask can hold 0.5 GB of data.

C) Data size is about 200 KB. ~200 MB

D) Data size of 1000 KB would "overflow device". 1000 KB = 1 MB < 500 MB

E) All of the above statements are false.

1GB = 1000MG

Solution

Given this flask, assume the red liquid is data and each mark represents 100 MB of data. Select a true statement.

A) Memory size is 200 MB. *~500MB*
B) Flask can hold 0.5 GB of data.
C) Data size is about 200 KB. ~200 MB
D) Data size of 1000 KB would "overflow device". 1000 KB = 1 MB < 500 MB
E) All of the above statements are false.

Solution

Given this flask, assume the red liquid is data and each mark represents 100 MB of data. Select a true statement.

A) Memory size is 200 MB. *~500MB*

B) Flask can hold 0.5 GB of data.

C) Data size is about 200 KB. *~200 MB*

D) Data size of 1000 KB would "overflow device". 1000 KB = 1 MB < 500 MB

E) All of the above statements are false.

Solution

Given this flask, assume the red liquid is data and each mark represents 100 MB of data. Select a true statement.

A) Memory size is 200 MB. $\sim 500MB$

B) Flask can hold 0.5 GB of data.

C) Data size is about 200 KB. $\sim 200\ MB$

D) Data size of 1000 KB would "overflow device". $1000\ KB = 1\ MB < 500\ MB$

E) All of the above statements are false.

Solution

Given this flask, assume the red liquid is data and each mark represents 100 MB of data. Select a true statement.

A) Memory size is 200 MB. *∼500MB*

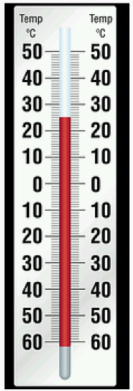
B) Flask can hold 0.5 GB of data.

C) Data size is about 200 KB. *∼200 MB*

D) Data size of 1000 KB would "overflow device". *1000 KB = 1 MB < 500 MB*

E) All of the above statements are false.

Analog vs. Digital: Thermometer example



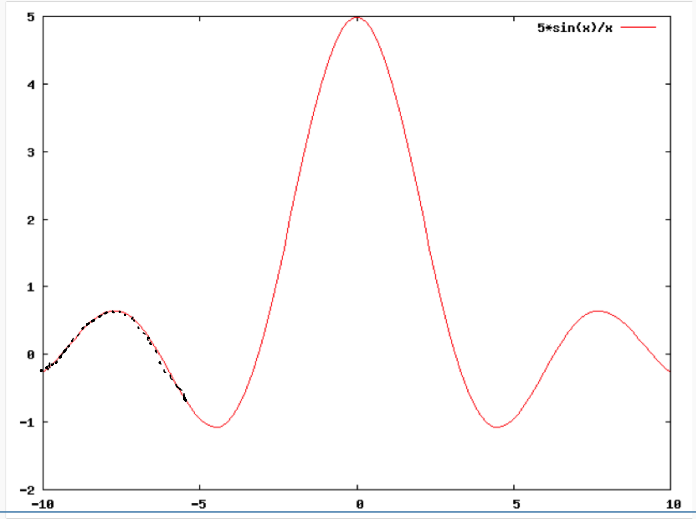
A thermometer contains liquid which expands and contracts in response to temperature changes.

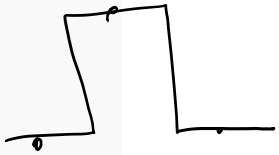
The liquid level is analog, and its expansion continuous over the temperature range.

This information can be represented using discrete units using digital thermometer, for example.

Conversion of Analogue to Digital

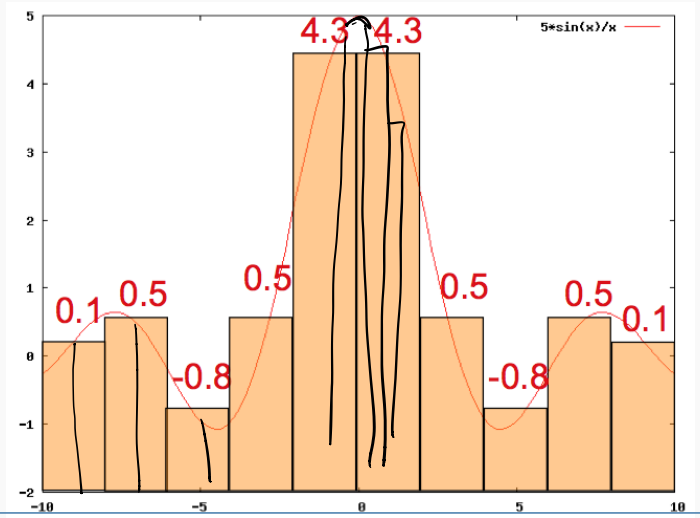
How would you digitize this analog data into 10 discrete points?





Conversion of Analogue to Digital

How would you digitize this analog data into 10 discrete points?



Why go digital over analogue?

1) Computers are digital and many home electronics are interfacing with computers.

2) Analog signals are more susceptible to noise that degrades the quality of the signal (sound, picture, etc.). The effect of noise also makes it difficult to preserve the quality of analog signals across long distances.

3) Reading data stored in analog format is susceptible to data loss and noise. Copying analog data leads to declining quality

A computer memory consists of billions of bits which allows for an almost limitless number of possible states.

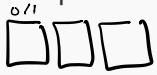
Bits are combined to allow more information to be represented including characters and numbers

- eg. 0 = off, 1= on
- 01100010 = "b"

To do this, it needs a set of rules on how to translate binary information into things like numbers, text, photos, video, etc.

Bits and Bytes

Numbers are encoded in a computer using a fixed number of bits (usually 32 or 64).



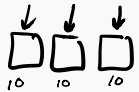
# of bits	Unique patterns	# of unique patterns
1	0, 1	$2^1 = 2$
2	00, 01, 10, 11	$2^2 = 4$
3	000, 001, 010, 100, 011, 101, 110, 111	$2^3 = 8$
⋮		⋮
32	...	$2^{32} = 4,294,967,296$
64	...	$2^{64} >$ 18 quintillion

The more bits you have, the more values you can represent.

Decimal System

- Assuming we use a 32-bit register, we now need a way of mapping or converting these unique patterns of 0s and 1s to a specific meaning (in this case a number). 0 1 2 3
- A *binary number* is a number expressed using only 0s and 1s (ie. in the base-2 numeral system or *binary numeral system*).
- Before discussing the binary system, let's first discuss a conversion system you should all be familiar with: the *decimal system*

Decimal System

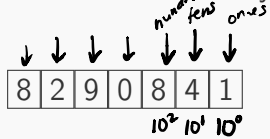


The decimal system uses digit placeholders, say $\square$, that can take on values from the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

The number of digits in the set is called the *base*. So the base for this system is 10

Reading from right to left, the first placeholder represents ones, the second, 10s, the third hundreds, and so on ...

We write eight million, two hundred ninety thousand, eight hundred forty one as:



$$= 8 * 10^6 + 2 * 10^5 + 9 * 10^4 + 0 * 10^3 + 8 * 10^2 + 4 * 10^1 + 1 * 10^0$$

Representing Data: Integers

A *binary system* works in the same way, only now, the placeholder must take a value from the set $\{0, 1\}$.

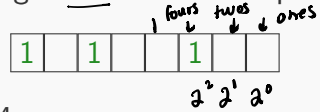
To put another way, instead of using base 10 wherein

ones=10^0, tens=10^1, hundreds=10^2, thousands=10^3, , etc.

we use base 2 where:

ones=2^0, 'twos'=2^1, 'fours'=2^2, 'eights'=2^3, etc. .

For example, the integer 164 would be expressed as



$164 = 128 + 32 + 4$

$1 \cdot 2^7 + 0 + 1 \cdot 2^5 + + 1 \cdot 2^2 +$

Representing Data: Integers

A *binary system* works in the same way, only now, the placeholder must take a value from the set $\{0, 1\}$.

To put another way, instead of using base 10 wherein

ones=10^0, tens=10^1, hundreds=10^2, thousands=10^3, , etc.

we use base 2 where:

ones=2^0, 'twos'=2^1, 'fours'=2^2, 'eights'=2^3, etc. .

For example, the integer 164 would be expressed as

1	0	1	0	0	1	0	0

$164 = 128 + 32 + 4$

$1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + z\, 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^0$

Converting decimal to binary

There are a number of websites online (ex 1, ex 2) that can convert numbers from our decimal system (or simply decimal) to the binary system (or simply binary). However the steps to do it on paper are quite easy:

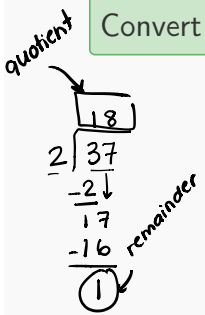
1. Divide the decimal number by 2.
2. Keep track of the integer quotient for the next iteration.
3. Keep track of the the remainder for the binary digit.
4. Repeat steps 1–3 until the quotient is equal to 0.
5. Construct the base 2 representation, by taking all the remainders starting from the bottom up.

Let's look at an example...

$37_{10} = ?_2$

Exercise

Convert 37_{10} from base 10 (i.e decimal) to binary base 2.



division = quotient + remainder

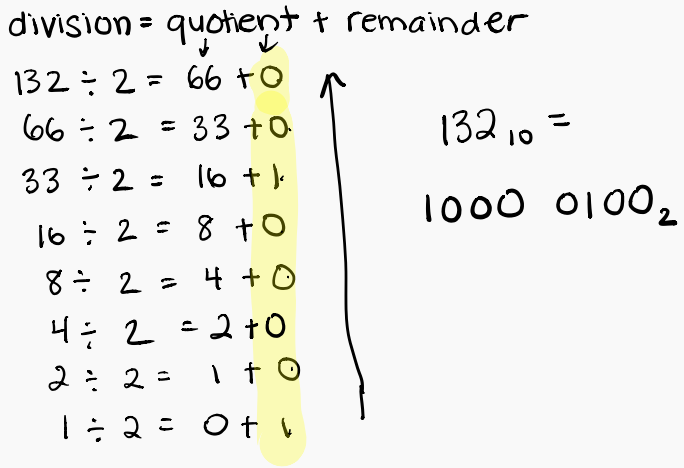
1) $37 \div 2 = 18 + 1$
2) $18 \div 2 = 9 + 0$
3) $9 \div 2 = 4 + 1$
4) $4 \div 2 = 2 + 0$
5) $2 \div 2 = 1 + 0$
6) $1 \div 2 = 0 + 1$

100101

$37_{10} = 10\ 0101_2$

Try it!

Convert 132_{10} from base 10 (i.e decimal) to binary base 2.



Question

Does any see a problem with this system?

Question

Does any see a problem with this system? **Hint:** this system is called *unsigned binary*

Representing Data: Integers

Recall a 32-bit register can store 2^{32} different values.

The range of integer values it will represent depends on the *encoding* type.

Unsigned Binary Range is 0 through 4,294,967,295 $=(2^{32}-1)$

2's compliment We use the first bit to store the sign (0=+, 1=-), so the range is -2,147,483,648 (-2^{31}) through 2,147,483,647 (2^{31} - 1).

Representing Data: Real Numbers

There are many standards for representing real numbers which include integers, rationals, fractions (eg -4, $\sqrt{2}$, 1/3) .

The most common is IEEE 754 format which uses floating-point (FP) representation.

Similar to scientific notation, FP expresses real numbers using a base and an exponent: $N = m * r^e$

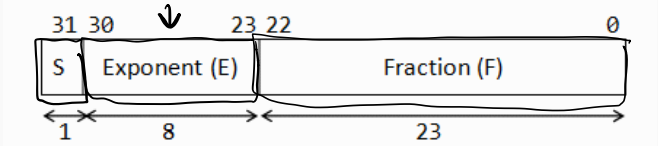
- m = mantissa (the decimal component of a number)
- e = exponent
- r = radix

IEEE 754 adopts a binary FP where $r = 2$.

Representing Data: Doubles and Floats [Photo souce]

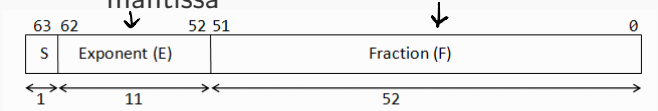
Modern computers adopt *IEEE 754* for floating-point numbers with two representation schemes:

32 bit/single-precision (or "float') 1-bit sign; 8-bit exponent; 23-bit mantissa



32-bit Single-Precision Floating-point Number

64 bit/double-precision 1-bit sign; 11-bit exp; 52-bit mantissa



64-bit Double-Precision Floating-point Number

- As before, let's revisited a related concept (which we all would have learned about in high school) to make this new concept easier to understand.
- Scientific notation operates in very much the same way as FP representation.
- Key features of normalized standard scientific notation:
  - There is a single non-zero digit to the left of the decimal point
  - The power indicates how far we've moved the decimal point to the left (+ exponent) or right (- exponent)

Representing Data: Normalized scientific notation

Example: Normalized scientific notation:

The number 55,125.17 in normalized scientific notation is:

$$5.512517 \times 10^{4}$$

Key features of normalized standard scientific notation:

- 5.512517 is our mantissa
- 4 is our exponent
- 10 is our radix

Representing Data: Normalized scientific notation

Example: Normalized scientific notation:

The number $0.000\,000\,007\,51$ in normalized scientific notation is:

$$7.51 \times 10^{-9}$$

Key features of normalized standard scientific notation:

- 7.51 is our mantissa
- -9 is our exponent
- 10 is our radix

Converting decimal fraction to binary - Phase 1

-37.17

1. Convert the integer part of decimal to binary (as on this slide)
2. Convert the fractional part of decimal to binary equivalent
   i) *Multiply* the fractional part by 2.
   ii) Keep track of the integer part for the binary digit
   iii) Keep track of the fractional part for the next iteration
   iv) Repeat steps 1–3 until the fractional part is equal to 0 or we have enough digits to fill the mantissa
   v) Construct the base 2 representation, by taking all the *integer* parts *starting from the top*
3. Write the result from step 1 to the left of the decimal and the result from step 2 to the right of the decimal.

multiplication = integer + fractional part
fract × 2 = integer + fra.

Converting decimal fraction to binary - Phase 2

4. Normalize the result from step 3 by shifting the decimal (either left or right) so that only one non zero digit remains to the left of the decimal. The number of places we shift will determine our exponent
5. Adjust the exponent by adding $2^{(8-1)} - 1$ to the exponent
6. Convert the result in step 5 to to binary (as on this slide)
7. Construct the binary number:
   i) Fill in the sign bit (0 = positive, 1 = negative)
   ii) Fill in the exponent bits with the result from step 6
   iii) Fill in the mantissa with the first 23 digits to the right of the decimal from step 4

Representing Data: Doubles and Floats

Example: 32-bit single precision

The number -37.17 stored as 4 consecutive bytes is:

sign	exponent			mantissa				
1	1000	0100	001	0100	1010	1110	0001	0100

Step 1) Convert the number to binary scientific notation

- Integer part (37) in binary 100101 (as shown in the previous exercise)

Step 2) Convert the fractional part of decimal to binary equivalent

1) $0.17 * 2 = 0 + 0.34$
2) $0.34 * 2 = 0 + 0.68$
3) $0.68 * 2 = 1 + 0.36$
4) $0.36 * 2 = 0 + 0.72$
5) $0.72 * 2 = 1 + 0.44$
6) $0.44 * 2 = 0 + 0.88$
7) $0.88 * 2 = 1 + 0.76$
8) $0.76 * 2 = 1 + 0.52$
9) $0.52 * 2 = 1 + 0.04$
10) $0.04 * 2 = 0 + 0.08$
11) $0.08 * 2 = 0 + 0.16$
12) $0.16 * 2 = 0 + 0.32$

. . . continued

. . .

13) $0.32 * 2 = 0 + 0.64$
14) $0.64 * 2 = 1 + 0.28$
15) $0.28 * 2 = 0 + 0.56$
16) $0.56 * 2 = 1 + 0.12$
17) $0.12 * 2 = 0 + 0.24$
18) $0.24 * 2 = 0 + 0.48$
19) $0.48 * 2 = 0 + 0.96$
20) $0.96 * 2 = 1 + 0.92$
21) $0.92 * 2 = 1 + 0.84$
22) $0.84 * 2 = 1 + 0.68$
23) $0.68 * 2 = 1 + 0.36$
24) $0.36 * 2 = 0 + 0.72$

. . . continued

We didn't (and won't) get a fractional part equal to zero but since we have enough iterations to fill the mantissa we can stop. $0.17_{10} = 0.00101011100001010001111 0 \ldots_2$

Step 3: Write the result from step 1 to the left of the decimal and the result from step 2 to the right of the decimal.

$$37.17_{10} = 100101.00101011100001010001111010_2$$

Step 4: Normalize the result from step 3 by shifting the decimal (either left or right) so that only one non zero digit remains to the left of the decimal (form `1.xxxxxx`).

$$= 100101.00101011100001010001111010_2$$
$$= 1.00101001010111000010100011110 10_2 \times 2^5$$

Since decimal moved 5 spaces to the left, the exponent becomes (positive) 5.

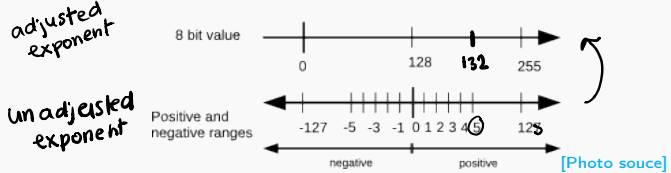
Step 5 Adjust the exponent by adding $2^{(8-1)} - 1$ to the exponent

$$5 \text{ becomes } 5 + 2^{(8-1)} - 1 = 132$$

Step 6 Convert the result in step 5 to to unsigned binary (done on this slide)

$$132_{10} = 1000\ 0100_2$$

Why the exponent adjustment?



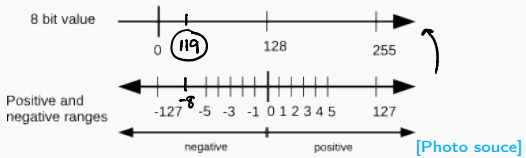
[Photo souce]

The 8-bits set aside for the exponent can represent $2^8 = 256$ different values (0– 255 using unsigned binary)

However, had the decimal moved to the *right*, the exponent would have been a negative number.

To accommodate negative integers in unsigned binary system we simply allow he lower half of the range (0–127) to be used for negative exponents and the upper other half (128–255) will be used for positive exponents.

Why the exponent adjustment?



[Photo souce]

- Our unadjusted positive exponent (eg 5) is now adjusted to $5 + (2^{8-1} - 1) = 5 + 127 = 132_{10} = 1000\ 0100_2$.
- To provide another examples:

  -8 is represented as $-8 + 127 = 119_{10} = 01110111_2$

  0 is represented as $0 + 127 = 127_{10} = 01111111_2$

  +1 is represented as $+1 + 127 = 128_{10} = 10000000_2$
- This scheme (called Excess-127) supports unadjusted exponents of -127 to 128

Example: 32-bit single precision

The number -37.17 stored as 4 consecutive bytes is:

sign	exponent	mantissa
1		

Step 7 Construct the binary number:

i) Fill in the sign bit (0 = positive, 1 = negative)

- since -37.17 is a negative number the first bit = 1.

Example: 32-bit single precision

The number -37.17 stored as 4 consecutive bytes is:

sign	exponent	mantissa
1	1000 0100	

Step 7 Construct the binary number:

ii) Fill in the exponent bits with the result from step 6

- Recall our unadjusted positive exponent (eg 5) was adjusted to
  $5 + (2^{8-1} - 1) = 5 + 127 = 132_{10} = 1000\ 0100_2$.

Example: 32-bit single precision

The number -37.17 stored as 4 consecutive bytes is:

sign	exponent	mantissa
1	1000 0100	001 0100 1010 1110 0001 0100

Step 7 Construct the binary number:

iii) Fill in the mantissa with the first 23 digits to the right of the decimal from step 4

1.00101001010111000010100 01111010

Precision

Take note of the fact that we deleted some information in order to get the number -37.17 to fit into the 32-bit single representation.

As a result, the storage of this number is -37.1699981689453125.

Lack of precision

Rounding errors will occur since some real numbers will have repeating bit representations. This lack of precision may be important in scientific applications!

Precision

Rational numbers of the form $x/2^k$, where x and k are integers, can have exact fractional binary representation

For example:

- $0.015625 = 1/2^6$, $-1.5 = -3/2$, $96 = 3/2^{-5}$ will have exact representation.
- 0.1, 123.4, 0.025 will not have exact representation.

Try It! You can check your answer here.

Convert 0.015625 to 32 bit single precision.

Step 1 Convert the integer part of decimal to binary

division = quotient + remainder

$0 \div 2 = \boxed{0} + 0$ $\quad 0_{10} = 0_2$

Step 2 Convert the fractional part of the decimal to binary

multiplication = integer + fractional part

$0.015625 \times 2 = 0 + 0.03125$

$0.03125 \times 2 = 0 + 0.0625$

$0.0625 \times 2 = 0 + 0.125$

$0.125 \times 2 = 0 + 0.25$

$0.25 \times 2 = 0 + 0.5$

$0.5 \times 2 = 1 + \boxed{0}$ ← done

0.015625_{10}

$= 0.000001_2$

Step 3 Write the result from step 1 to the left of the decimal and the result from step 2 to the right of the decimal.
Step 4 Normalize the result from step 3

Step 5 Adjust the exponent by adding $2^{(8-1)} - 1$ to the exponent **Step 6** Convert the result in step 5 to to binary

Unadj exp = -6

Adj exp = -6 + 127 = 121

Convert to unsigned binary

121 ÷ 2 = 60 + 1
60 ÷ 2 = 30 + 0
30 ÷ 2 = 15 + 0
15 ÷ 2 = 7 + 1
7 ÷ 2 = 3 + 1
3 ÷ 2 = 1 + 1
1 ÷ 2 = 0 + 1

$121_{10} = 01111001_2$

$= 0111\ 1001_2$

Step 7 Construct the binary number:

i) Fill in the sign bit (0 = positive, 1 = negative)

ii) Fill in the exponent bits with the result from step 6

iii) Fill in the mantissa with the first 23 digits to the right of the decimal from step 4

0.015625

0 0111 1001 0.... 0

23 zeros

Comment

- While 64-bit can accommodate a wider range of number as shown in the table below, in most scenarios, you will be fine using 32-bit.

Table: Source: here

Type	Size	Range
float	32 bits	-3.4E+38 to +3.4E+38
double	64 bits	-1.7E+308 to +1.7E+308

Hexadecimal

We saw how binary and decimal systems consist of two and ten digits respectively.

For that reason, binary is also known as *base 2* and decimal as *base 10*.

Hexadecimal is another such system that contains sixteen digits and is therefore known as *base 16*.

Like decimal, hexadecimal uses the same 10 digits (`0--9`)

In addition, it uses: `A`,`B`,`C`,`D`,`E`, and `F`. $\{0, 1, \ldots 9, A, B, C \ldots F\}$

16 choices

Hexadecimal Base 16	Decimal Base 10	Binary Base 2
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Notice, it takes 4 binary digits (a nibble) to represent a single hexadecimal digits.

Consequently, hexadecimal provides a compact short hand for binary.

Another benefit for using hexadecimal is that it is easier (for a human) to read.

The most common place to see this hexadecimal notation is when describing colours, eg.

Roses are #FF0000 (in decimal (255, 0, 0))
Violets are #0000FF (in decimal (0, 0, 255))

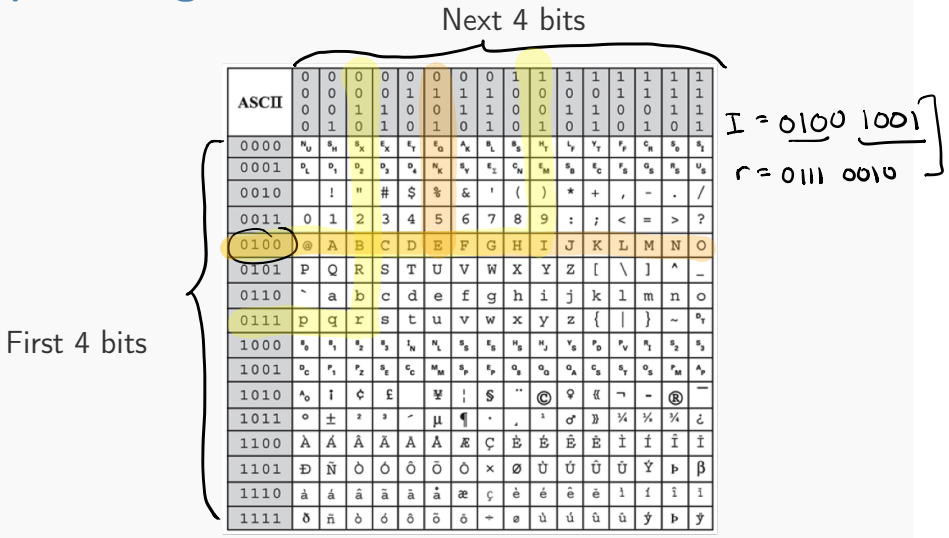
Representing Data: Characters

A character is mapped to a sequence of bits using a *lookup or translation table*.

A common encoding is *ASCII* (American Standard Code for Information Interchange), which uses 8 bits to represent characters.

bits	character
01000001	A
01000010	B
01000011	C
01000100	D
01000101	E
01000110	F
...	...

Representing Data: Characters



ASCII	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0010		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
1000	80	81	82	83	IND	NEL	86	87	HTS	89	VTS	PLD	PLU	8D	8E	8F
1001	90	91	92	93	94	MW	SPA	EPA	98	99	9A	CSI	ST	OSC	PM	APC
1010	NBSP	¡	¢	£		¥	¦	§	¨	©	ª	«	¬	-	®	¯
1011	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
1100	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
1101	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
1110	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
1111	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Exercise: Try writing your name in ASCII!

Representing Data: Characters

Example 2

What ASCII character is 0100 0101?

A) T

B) !

C) @

D) E

Representing Data: Characters

Answer:

What ASCII character is 0100 0100?

A) A

B) !

C) @

D) *E*

ASCII Encoding

Example 3

What is "Test" encoded in ASCII?

A) 01110100 01100101 01110011 01110100

B) 01010100 01100101 01110011 01110100

C) 01000101 01010110 00110111 01000111

D) 01010100 01000101 01010011 01010100

"e"

"E"

ASCII Encoding

Answer:

What is "Test" encoded in ASCII?

A) 01110100 01100101 01110011 01110100

B) *01010100 01100101 01110011 01110100*

C) 01000101 01010110 00110111 01000111

D) 01010100 01000101 01010011 01010100

- While these conversions are useful to see, these conversions need not be done 'by hand'.
- If there was a time when we need to see these conversions, there are countless sites available online for doing so, eg. ASCII to Binary

Limitations with ASCII?

Does anyone see a problem with using ASCII as a character encoding?

- While these conversions are useful to see, these conversions need not be done 'by hand'.
- If there was a time when we need to see these conversions, there are countless sites available online for doing so, eg. ASCII to Binary

Limitations with ASCII?

Does anyone see a problem with using ASCII as a character encoding?

Although ASCII is suitable for English text, many world languages, including Chinese, require a larger number of symbols to represent their basic alphabet.

Representing Text Beyond ASCII - Unicode

The *Unicode* standard uses patterns of 16-bits (2 bytes) to represent the major symbols used in all languages.

- First 256 characters exactly the same as ASCII.
- Maximum number of symbols: 65,536.

Unicode can be implemented by different character encodings (eg. UTF-8, UTF-16, UTF-32) with new versions released on a regular basis.

UTF-8, the dominant encoding on the World Wide Web (used in over 92% of websites).

As of May 2019 the most recent version, Unicode 12.1, contains 137,994 characters covering 150 modern and historic scripts, as well as multiple symbol sets and emojis. 🙌.

Representing Data: Strings

A string is a sequence of characters allocated in consecutive memory bytes.

A string has a terminator to know when it ends:

- **Null-terminated string** last byte value is '\0' to indicate end of string.
- **length-prefixed** length of string in bytes is specified (usually in the first few bytes before string starts).

Representing Data: Dates and Times

A *date* value can be represented in multiple ways:

Integer representation number of days past since a given date

- Example: Julian Date (astronomy) - number of days since noon, January 1, 4713 BC. Why this date?

String representation represent a date's components (year, month, day) as individual characters of a string

- Example: YYYYMMDD or YYYYDDD

Representing Data: Dates and Times

-/+ integers = time before/after (+/- of 136 years)
1901-12-13 to 2038-01-19 03:14:07 UTC

A *time* value can also be represented in similar ways:

Integer representation number of sec since a given time

- Example: Number of seconds since Thursday, January 1, 1970 (UNIX)

String representation hours, minutes, seconds, fractions

- Example: HHMMSSFF

Read here about the year 2038 problem (analogy to the Y2K problem).

Encoding other data

We have seen how we can encode characters, numbers, and strings using only sequences of bits (and translation tables).

The documents, music, and videos that we commonly use are much more complex. However, the principle is exactly the same. We use sequences of bits and *interpret* them based on the *context* to represent information.

As we learn more about representing information, always remember that everything is stored as bits, it is by interpreting the context that we have information.

Metadata

Metadata is data that describes other data.

Examples of metadata include:

- names of files
- column names in a spreadsheet
- table and column names and types in a database

Metadata helps you understand how to interpret and manipulate the data.

Files

A *file* is a sequence of bytes on a storage device.

- A file has a name.
- A computer reads the file from a storage device into memory to use it.

The operating system manages how to store and retrieve the file bytes from the device.

The program using the file must know how to interpret those bytes based on its information (e.g. metadata) on what is stored in the file.

File Encoding

A file *encoding* is how the bytes represent data in a file.

A file encoding is determined from the file extension (e.g. .txt or .xlsx) which allows the operating system (OS) to know how to process the file.

The extension allows the OS to select the program to use.
The program understands how to process the file in its format.

Binary vs Text files

- At a generic level of description, there are two kinds of computer files: text files and binary files.
- The difference between binary and text files is in how these bytes are interpreted.
- A text file is a file encoded in a character format such as ASCII or Unicode. These files are readable by humans.
- Data analytics will often involve processing text files.
- We can usually tell if a file is binary or text based on its file extension.

The text looks readable to a human (may contain punctuation symbols (like HTML, RTF, and other markup formats)), there is some visible structure! Any text file can be viewed or edited in any text editor.

File Encodings: Text Files

There are many different text file encodings:

- *Web standards*: html, xml, css, svg, json, ...
  - *JSON file* data encoded in JSON (*J*ava*S*cript *O*bject *N*otation) format
  - *XML file* data encoded in XML (Extensible Markup Language) format
- *Tabular data*: csv, tsv, ...
  - *CSV comma-separate file* each line is a record, fields separated by commas
  - *tab-separated file* each line is a record, fields separated by tabs
- *Documents*: txt, tex, markdown, asciidoc, rtf, ps, ...

CSV (comma-separated) file

```
Id,Name,Province,Balance
1,Joe Smith,BC,345.42
```

Question:

In these file encodings, what is data and what is metadata?

tab separated file

```
Id        Name         Province        Balance
1        Joe Smith         BC         345.42
```

JSON file

```
{"Id":1, "Name":"Joe Smith", "Province":"BC", "Balance":345.42}
```

XML file

```
<customers><customer>
        <id>1</id>            <name>Joe Smith</name>
    <province>BC</province> <balance>345.42</balance>
</customer></customers>
```

CSV (comma-separated) file

```
Id,Name,Province,Balance
1,Joe Smith,BC,345.42
```

Answer:

In these file encodings, this is data and what is metadata?

CSV (tab-separated) file

```
Id      Name        Province      Balance
1       Joe Smith       BC        345.42
```

JSON file

```
"Id":1, "Name":"Joe Smith", "Province":"BC", "Balance":345.42
```

XML file

```
<customers><customer>
        <id>1</id>            <name>Joe Smith</name>
    <province>BC</province> <balance>345.42</balance>
</customer></customers>
```

File Encoding: Binary File

A binary file always needs a matching software to read or write it. For example, an MP3 file can be produced by a sound recorder or audio editor, and it can be played in a music player or audio editor. But an MP3 file cannot be played in an image viewer or a database software.

A *binary file* encodes data in a format that is not designed to be human-readable and is in the format used by the computer.

Binary files are often faster to process as they do not require translation from text form and may also be smaller.

Processing a binary file requires the user to understand its encoding so that the bytes can be read and interpreted properly.

If you use a text editor to open a binary file, you will see copious amounts of garbage, seemingly random accented and Asian characters, and long lines overflowing with text – this exercise is safe but pointless. However, editing or saving a binary file in a text editor will corrupt the file, so never do this.

File Encodings: Binary Files

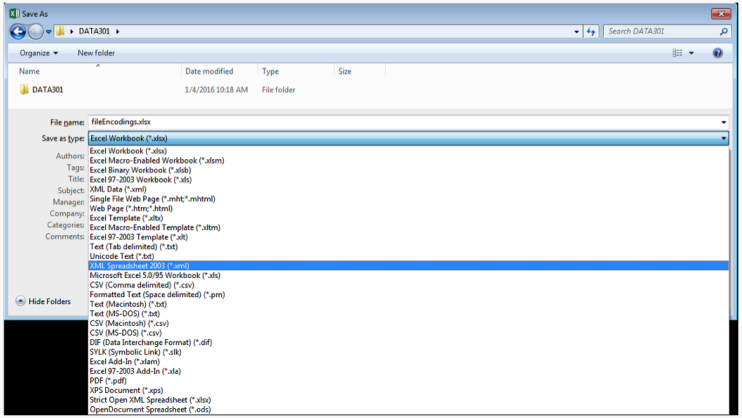
There are many different text file encodings:

- *Image* jpg, png, gif, bmp, tiff, psd, ...
- *Videos*: mp4, mkv, avi, mov, mpg, vob, ...
- *Audio*: mp3, aac, wav, flac, ogg, mka, wma, ...
- *Documents*: pdf, doc, xls, ppt, docx, odt, ...
- *Archive*: zip, rar, 7z, tar, iso, ...
- *Database*: mdb, accde, frm, sqlite, ...
- *Executable*: exe, dll, so, class, ...

Try It: File Encodings

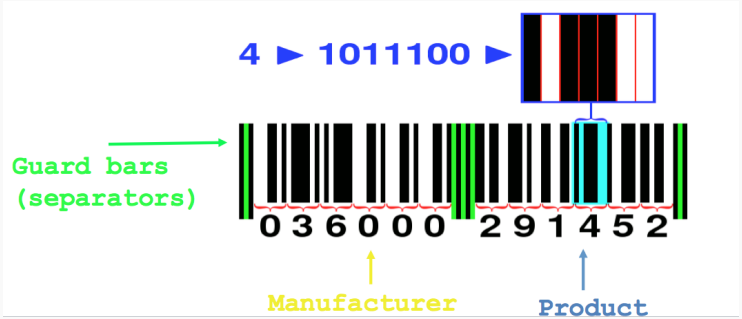
Exercise:

Use the **fileEncodings.xlsx** file and save the file as **CSV**, **tab-separated**, and **XML**. Look at each file in a text editor.



UPC Barcodes

Universal Product Codes (UPC) encode manufacturer on left side and product on right side. Each digit uses 7 bits with different bit combinations for each side (can tell if upside down).



QR code

A *QR* (*Q*uick *R*esponse) code is a 2D optical encoding developed in 1994 by Toyota with support for error correction.

Make your own codes at: `www.qrstuff.com`.

NATO Broadcast Alphabet

The code for broadcast communication is purposefully inefficient, to be distinctive when spoken amid noise.

A	Alpha	J	Juliet	S	Sierra
B	Bravo	K	Kilo	T	Tango
C	Charlie	L	Lima	U	Uniform
D	Delta	M	Mike	V	Victor
E	Echo	N	November	W	Whiskey
F	Foxtrot	O	Oscar	X	X-ray
G	Golf	P	Papa	Y	Yankee
H	Hotel	Q	Quebec	Z	Zulu
I	India	R	Romeo		

Advanced: The Time versus Space Tradeoff

A fundamental challenge in computer science is encoding information efficiently both in terms of space and time.

At all granularities (sizes) of data representation, we want to use as little space (memory) as possible. However, saving space often makes it harder to figure out what the data means (think of compression or abbreviations). In computer terms, the data takes longer to process.

The *time versus space tradeoff* implies that we can often get a faster execution time if we use more memory (space). Thus, we often must strive for a balance between time and space.

Review: Memory Size

Example 4

Which is bigger?

A) 10 TB

B) 100 GB

C) 1,000,000,000,000 bytes

D) 1 PB

Review: Memory Size

Answer:

Which is bigger?

A) 10 TB *=10,000 GB*

B) 100 GB

C) 1,000,000,000,000 bytes *= 1000 GB*

D) *1 PB = 1,000,000 GB*

Review: Metadata vs. Data

Example 5

How many of the following are TRUE?

- It is possible to have data without metadata.
- Growth rates of data generation are decreasing.
- It is possible to represent all decimal numbers precisely on a computer.
- A character encoded in Unicode uses twice as much space as ASCII.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Review: Metadata vs. Data

Answer:

How many of the following are TRUE?

- It is possible to have data without metadata.
- Growth rates of data generation are decreasing.
- It is possible to represent all decimal numbers precisely on a computer.
- A character encoded in Unicode uses twice as much space as ASCII.

A) 0 B) 1 C) *2* D) 3 E) 4

Review: Metadata vs. Data

Answer:

How many of the following are TRUE?

- It is possible to have data without metadata.
- Growth rates of data generation are decreasing.
- It is possible to represent all decimal numbers precisely on a computer.
- A character encoded in Unicode uses twice as much space as ASCII.

A) 0 B) 1 C) *2* D) 3 E) 4

Review: Metadata vs. Data

Answer:

How many of the following are TRUE?

- It is possible to have data without metadata.
- Growth rates of data generation are decreasing.
- It is possible to represent all decimal numbers precisely on a computer.
- A character encoded in Unicode uses twice as much space as ASCII.

A) 0 B) 1 C) *2* D) 3 E) 4

Review: Metadata vs. Data

Answer:

How many of the following are TRUE?

- It is possible to have data without metadata.
- Growth rates of data generation are decreasing.
- It is possible to represent all decimal numbers precisely on a computer.
- A character encoded in Unicode uses twice as much space as ASCII.

A) 0 B) 1 C) *2* D) 3 E) 4

Conclusion

- All *data* is encoded as bits on a computer.
- *Metadata* provides the context to understand how to interpret the data to make it useful.
- Memory capacity and data sizes are measured in bytes.
- *Files* are sequences of bytes stored on a device. A *file encoding* is how the bytes are organized to represent data
  - Text files (comma/tab separated, JSON, XML) are often processed during data analytics tasks.
  - Binary files are usually only processed by the program that creates them.

As a data analyst, understanding the different ways of representing data is critical as it is often necessary to transform data from one format to another.

Objectives

- Define: computer, software, memory, data, memory size/data size, cloud
- Explain "Big Data" and describe data growth in the coming years.
- Compare and contrast: digital versus analog
- Explain how integers, doubles, and strings are encoded.
- Explain why ASCII table is required for character encoding.
- Explain why Unicode is used in certain situations instead of ASCII.
- Explain the role of metadata for interpreting data.
- Define: file, file encoding, text file, binary file
- Encode using the NATO broadcast alphabet.
- Discuss the time-versus-space tradeoff.