

# Data 301 Data Analytics

## Data Representation

**Dr. Irene Vrbik**

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

Term 1, 2018

# 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.
- ▶ *Hardware* 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.

# 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

The real-world is *analog* where the information is encoded on a continuous signal (spectrum of values).

Computers represent data *digitally*, meaning that data is represented using discrete units called as bits (**B**inary **D**igits).

# 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 *D*igits)
- ▶ A byte can store one character of text.



Larger units:

• kilobyte (KB)	$10^3$ bytes	(1000 bytes)
• megabyte (MB)	$10^6$ bytes	(1000 KB)
• gigabyte (GB)	$10^9$ bytes	(1000 MB)
• terabyte (TB)	$10^{12}$ bytes	:
• petabyte (PB)	$10^{15}$ bytes	:
• exabyte (EB)	$10^{18}$ bytes	:
• zettabyte (ZB)	$10^{21}$ bytes	:

# Memory/Data Size

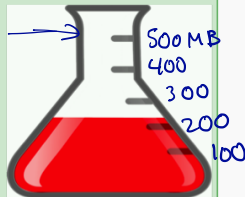
*Memory size* is a measure of memory storage capacity in bytes.

- ▶ It represents the *maximum* capacity of data in the device.

## Question:

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".





# Memory/Data Size

*Memory size* is a measure of memory storage capacity in bytes.

- It represents the *maximum* capacity of data in the device.

## Question:

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. 500 MB
- C) Data size is about 200 KB.
- D) Data size of 1000 KB would "overflow device".



# Memory/Data Size

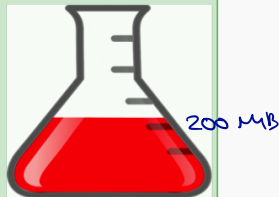
*Memory size* is a measure of memory storage capacity in bytes.

- ▶ It represents the *maximum* capacity of data in the device.

## Question:

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".



# Memory/Data Size

*Memory size* is a measure of memory storage capacity in bytes.

- It represents the *maximum* capacity of data in the device.

## Question:

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. ~200 MB
- D) Data size of 1000 KB would "overflow device".



# Memory/Data Size

*Memory size* is a measure of memory storage capacity in bytes.

- ▶ It represents the *maximum* capacity of data in the device.

## Question:

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.  $\sim 200$  MB
- D) Data size of 1000 KB would "overflow device".  $1000 \text{ KB} = 1 \text{ MB} < 500 \text{ MB}$



# Massive Growth of Data - “Big Data”

*Big Data* is the general term used to describe the explosion of data collected and the opportunities to transform this data into useful insights to benefit society.

- ▶ "Big" enough to challenge how the data can be processed.

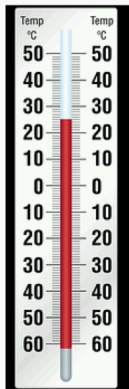
Some of the goals of big data analytics is to discover meaningful patterns, extract information, draw conclusions and make decisions.

# Massive Growth of Data - “Big Data”

Data facts from Forbes Magazine, May 21, 2018 [1]:

- ▶ Over 90% of the data in world history was created over the last two years.
- ▶ An estimated 2.5 quintillion bytes (2.5 EB) generated per day.
- ▶ Google processes about 3.5 billion requests/day and stores about 10 EB of data.
- ▶ Facebook collects 500 TBs/day (~2.5 billion items) and stores 100+ PB of photos.
- ▶ Users watch 4,146,600 videos every minute on YouTube

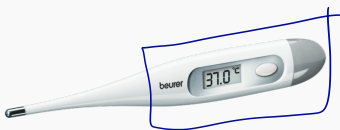
# Analog vs. Digital: Thermometer example



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

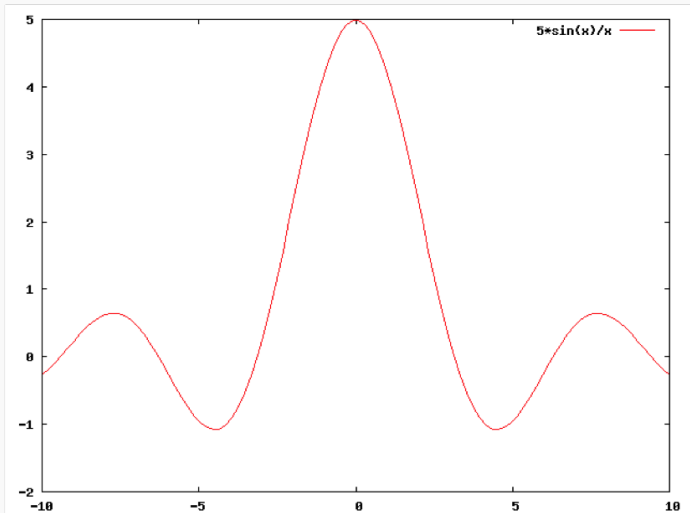
The liquid level is analog, and its expansion continuous over the temperature range.

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



# Conversion of Analogue to Digital

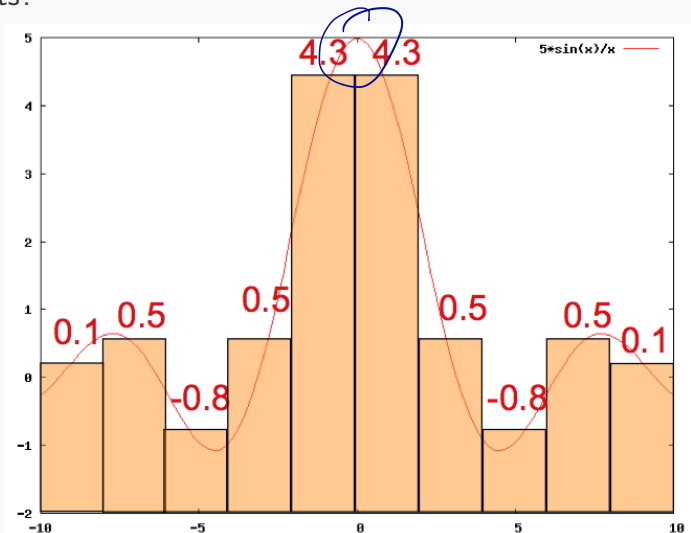
How would you digitize this analog data into 10 discrete points?





# 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

Let's take a look at how a computer represents integers.

An integer is a whole number. It is encoded in a computer using a fixed number of bits (usually 32 or 64).

Number of bits	Unique Patterns Attainable
1	<u>0</u> , <u>1</u>
2	<u>00</u> , <u>01</u> , <u>10</u> , <u>11</u>
3	000, 001, 010, 100, 011, 101, 110, 111

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

A 32-bit register can store  $2^{32}$  different values.

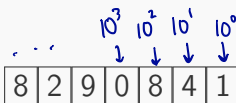
# Decimal System

Before discussing binary, let's first discuss a conversion system you should all be familiar with: the *decimal system*

This system uses digit placeholders (denoted by  $\square$ ) that can take on numerical values between 0 and 9.

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 of 0 or 1.

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

$$164 = 128 + 32 + 4$$

$$1 \cdot 2^7 + \quad + 1 \cdot 2^5 + \quad + z \quad + 1 \cdot 2^2 +$$

# Representing Data: Integers

A *binary system* works in the same way, only now, the placeholder must take a value of 0 or 1.

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
---	---	---	---	---	---	---	---

$$164 = 128 + 32 + 4$$

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

### Example

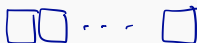
Convert  $37_{10}$  from base 10 (i.e. decimal) to binary base 2.

$$\begin{array}{rcl} 37 \div 2 & = & 18 + 1 \\ 18 \div 2 & = & 9 + 0 \\ 9 \div 2 & = & 4 + 1 \\ 4 \div 2 & = & 2 + 0 \\ 2 \div 2 & = & 1 + 0 \\ 1 \div 2 & = & 0 + 1 \end{array}$$

$$37_{10} = 100101_2$$



# Representing Data: Integers



Using 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 complement** 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:

$$N = m * r^e$$

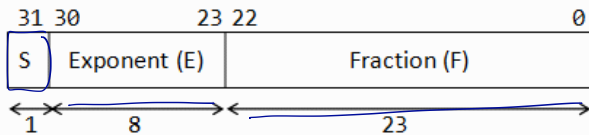
- ▶  $m$  = mantissa (the decimal component of a number)
- ▶  $e$  = exponent
- ▶  $r$  = radix

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

# Representing Data: Doubles and Floats<sup>1</sup>

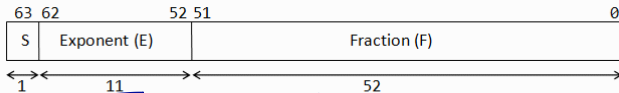
Modern computers adopt *IEEE 754* for floating-point numbers with two representation schemes:

**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**

# Representing Data: Normalized scientific notation

## Example: Normalized scientific notation:

The number 55,125.17 is normalized scientific notation is:

$$\boxed{5.512517} \times 10^{\textcircled{4}} =$$

Key features of normalized standard scientific notation:

- ▶ There is a single digit to the left of the decimal point
- ▶ The power indicates how far we've moved the decimal point to the left
- ▶ 5.512517 is our mantissa
- ▶ 4 is our exponent
- ▶ 10 is our radix

# Representing Data: Doubles and Floats

## Example: 32-bit single precision

32 "float"

The number -37.17 stored as 4 consecutive bytes is:

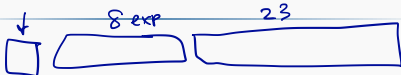
sign	exponent	mantissa	
1	1000 0100	001 0100 1010 1110 0001 0100	

**Step 1)** Convert the number to binary scientific notation

- ▶ Integer part (37) in binary 100101 (typical to exclude leading 0s)
- ▶ The fractional part (0.17) in binary is

0.0010 1011 1000 0101 0001 1110 10

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



## Example

Convert 0.17 to binary.

$$1) \quad 0.17 \times 2 = 0 + \underline{0.34}$$

$$2) \quad 0.34 \times 2 = 0 + \underline{0.68}$$

$$3) \quad 0.68 \times 2 = 1 + \underline{0.36}$$

$$4) \quad 0.36 \times 2 = 0 + \underline{0.72}$$

$$5) \quad 0.72 \times 2 = 1 + \underline{0.44}$$

⋮

$$23) \quad 0.68 \times 2 = 1 + \underline{0.36}$$

$$24) \quad 0.36 \times 2 = 0 + \underline{0.72}$$

$0.17_{10} =$

0010 1011  
1000 0101  
0001 1110

# 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 2)** normalize by shifting decimal so that there is no leading 0 (form 1.xxxxxx). Adjust the exponent accordingly.

The decimal moved 15 spaces to the left, so

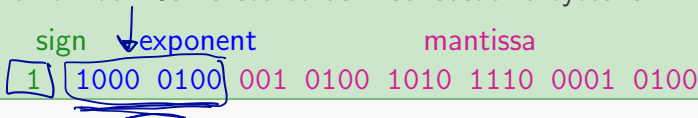
$$\begin{aligned}
 &= 1.\overset{37}{\text{00101}}\overset{0.17}{001010111100001010001111010}_2 \\
 &= 1.\text{00101001010111100001010001111010}_2 \times 2^{\textcircled{5}}
 \end{aligned}$$

23    0s & 1s

# Representing Data: Doubles and Floats

## Example: 32-bit single precision

The number -37.17 stored as 4 consecutive bytes is:



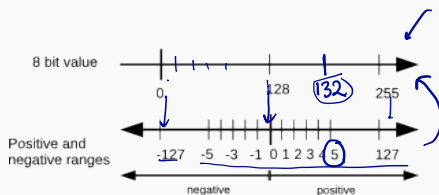
**Step 3)** Adjust the exponent in 8 bit excess/bias notation and convert it from decimal binary.

- ▶ The (unadjusted) exponent is 5
- ▶ (Adjusted) exponent in bias notation is  $5 + (2^{8-1} - 1) = 5 + 127 = 132_{10}$
- ▶  $132_{10}$  into binary = 10000100<sub>2</sub>

*verify as exercise.*



# Why the exponent adjustment?

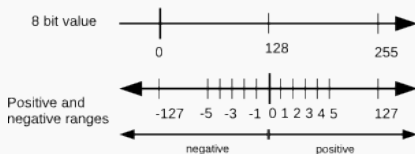


$$132 \div 2 = 66$$

The 8-bits set aside for the exponent can represent  $2^8 = 256$  different values.

The lower half of the range (0–127) will be used for negative exponents and the upper other half (128–255) will be used for positive exponents. Photo sourced from: [\[2\]](#)

# Why the exponent adjustment?



- ▶ An unadjusted positive exponent (eg 5) would be adjusted to  $5 + 127 = 132$  and converted to the 8-bit value of  $10000100_2$ .
- ▶ An unadjusted negative exponent (eg. -8) would have an 8-bit value of  $-8 + 127 = 119_{10} = 1110111_2$

Photo sourced from: [\[2\]](#)

# 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	

Most accurate representation = -3.71699981689453125E1

-37.1699...-

**Step 3)** Normalize the mantissa: remove leading 1 and extract first 23 bits (fill 23 bits with 0s need be)

~~1~~.0010100101011100001010001111010

See an [online converter](#) for -37.17

# 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. Hence 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.

# 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.

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.



# Representing Data: Characters

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

A common encoding is <sup>extended</sup>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

7bit = 128.

Next 4 bits

8bit  
256  
characters

First 4 bits

ASCII	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0000	N	S	S	S	E	E	A	S	S	H	L	V	F	C	S	S
0001	D	D	D	D	D	N	S	S	C	M	S	E	F	S	S	U
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	{		}	~	0 <sub>r</sub>
1000	0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0 <sub>3</sub>	0 <sub>4</sub>	0 <sub>5</sub>	0 <sub>6</sub>	0 <sub>7</sub>	0 <sub>8</sub>	0 <sub>9</sub>	0 <sub>A</sub>	0 <sub>B</sub>	0 <sub>C</sub>	0 <sub>D</sub>	0 <sub>E</sub>	0 <sub>F</sub>
1001	0 <sub>G</sub>	0 <sub>H</sub>	0 <sub>I</sub>	0 <sub>J</sub>	0 <sub>K</sub>	0 <sub>L</sub>	0 <sub>M</sub>	0 <sub>N</sub>	0 <sub>O</sub>	0 <sub>P</sub>	0 <sub>Q</sub>	0 <sub>R</sub>	0 <sub>S</sub>	0 <sub>T</sub>	0 <sub>U</sub>	0 <sub>V</sub>
1010	0 <sub>W</sub>	0 <sub>X</sub>	0 <sub>Y</sub>	0 <sub>Z</sub>	0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0 <sub>3</sub>	0 <sub>4</sub>	0 <sub>5</sub>	0 <sub>6</sub>	0 <sub>7</sub>	0 <sub>8</sub>	0 <sub>9</sub>	0 <sub>A</sub>	0 <sub>B</sub>
1011	0 <sub>C</sub>	0 <sub>D</sub>	0 <sub>E</sub>	0 <sub>F</sub>	0 <sub>G</sub>	0 <sub>H</sub>	0 <sub>I</sub>	0 <sub>J</sub>	0 <sub>K</sub>	0 <sub>L</sub>	0 <sub>M</sub>	0 <sub>N</sub>	0 <sub>O</sub>	0 <sub>P</sub>	0 <sub>Q</sub>	0 <sub>R</sub>
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

Try writing your name in ASCII!

# Representing Data: Characters

## Question:

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) D

# ASCII Encoding

## Question:

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

# 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

Quick link: <http://www.binaryhexconverter.com/ascii-text-to-binary-converter>

# Representing Text Beyond ASCII - Unicode

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

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.

# 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:

- C-string*


▶ **Null-terminated string** last byte value is `'\0'` to indicate end of string. *NUL ASCII*
- ▶ **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

**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

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

 **Integer representation** number of seconds since a given time

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

**String representation** hours, minutes, seconds, fractions

- ▶ Example: HHMMSSFF

# 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.

# File Encodings: Text Files

A text file is a file encoded in a character format such as ASCII or Unicode. These files are readable by humans.

There are many different text file encodings:

- ▶ *CSV comma-separated file* each line is a record, fields separated by commas
- ▶ *tab-separated file* each line is a record, fields separated by tabs
- ▶ *JSON file* data encoded in JSON (*J*ava*S*cript *O*bject *N*otation) format
- ▶ *XML file* data encoded in XML (*E*xtensible *M*arkup *L*anguage) format

Data analytics will often involve processing text files.

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>
```

*Customer: Joe Smith  
Id: 1*



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 **meta-data**?

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

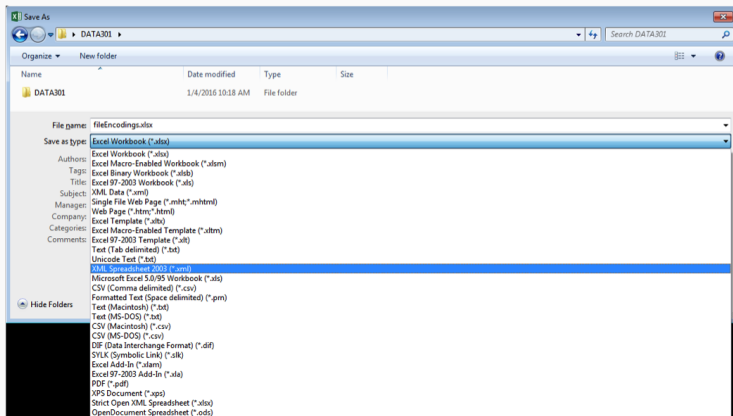
Binary files are often faster to process as they do not require translation from text form and may also be smaller.

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

# Try It: File Encodings

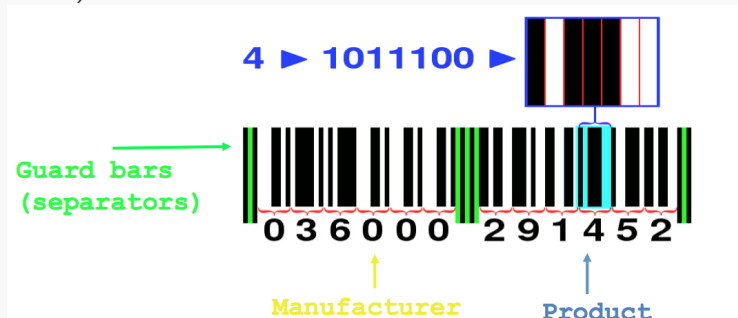
## Question:

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.



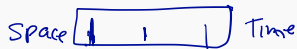
Make your own codes at: [www.qrstuff.com](http://www.qrstuff.com).

# 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

## Question:

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: Memory Size

## Answer:

Which is bigger?

- A) 10 TB
- B) 100 GB
- C) 1,000,000,000,000 bytes
- D) *1 PB*

# Review: Metadata vs. Data

## Question:

Select a TRUE statement.

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

## Review: Metadata vs. Data

### Answer:

---

Select a TRUE statement.

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

## Review: Metadata vs. Data

### Answer:

---

Select a TRUE statement.

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

# Review: Metadata vs. Data

## Answer:

Select a TRUE statement.

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

extended  
8 bits

16 bits

## Review: Metadata vs. Data

### Answer:

---

Select a TRUE statement.

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

# 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 of devices 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 the 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.





**Forbes magazine.**

<https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-statistics/#6c09172160ba>.

Accessed: 2018-08-21.



**penjee.**

<https://blog.penjee.com/binary-numbers-floating-point-conversion/>.

Accessed: 2018-09-10.