Data Science
Survival Skills

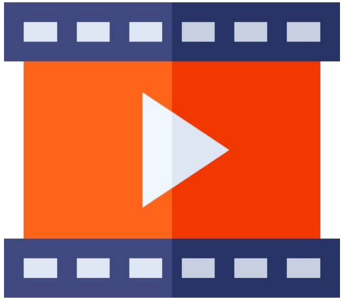
What is actually data?

Data.

"Data" originates from the Latin word *datum*, meaning "something given."

a collection of facts, measurements, or observations that are recorded and used for analysis.

Data comes in various flavors.



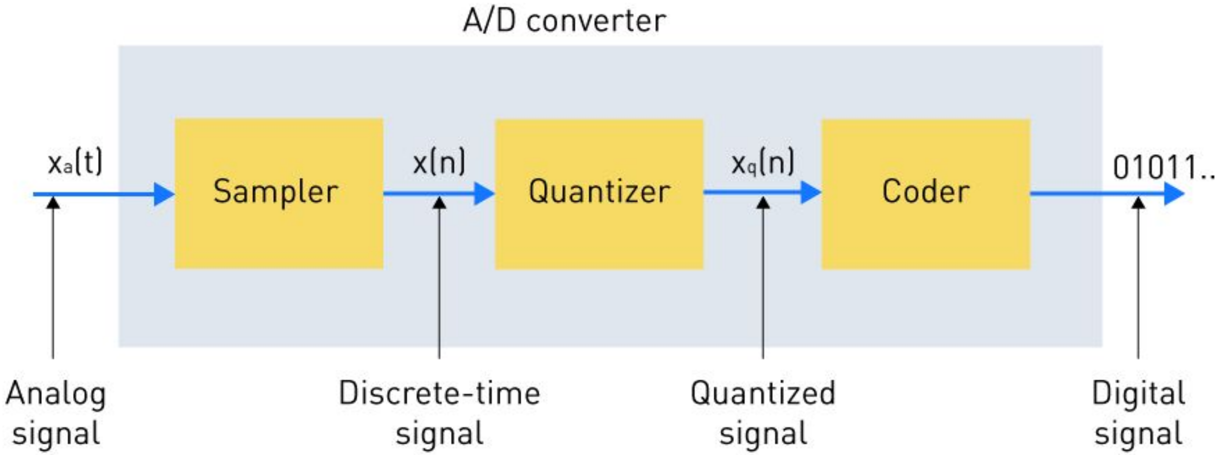
Measuring the world around us.

We use our senses to digitize the world and make it available to our brain ...and more!

NTC-Thermistor (Resistance depending on temperature)

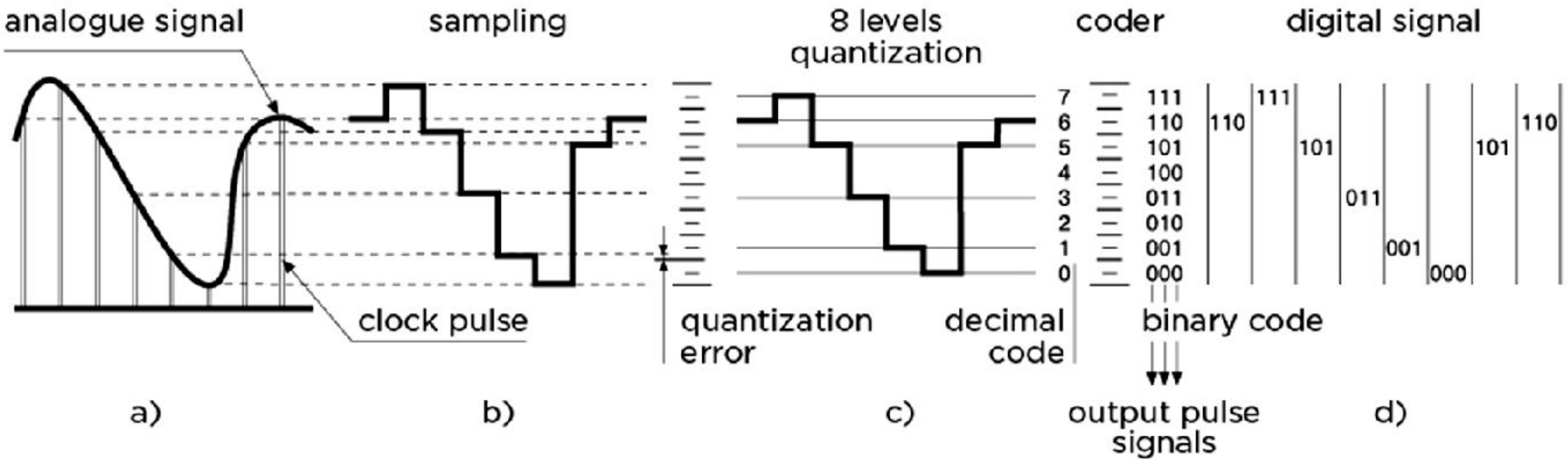
https://speakingofautismcom.wordpress.com/2020/04/30/the-twenty-six-senses/

Digitizing an analogue signal



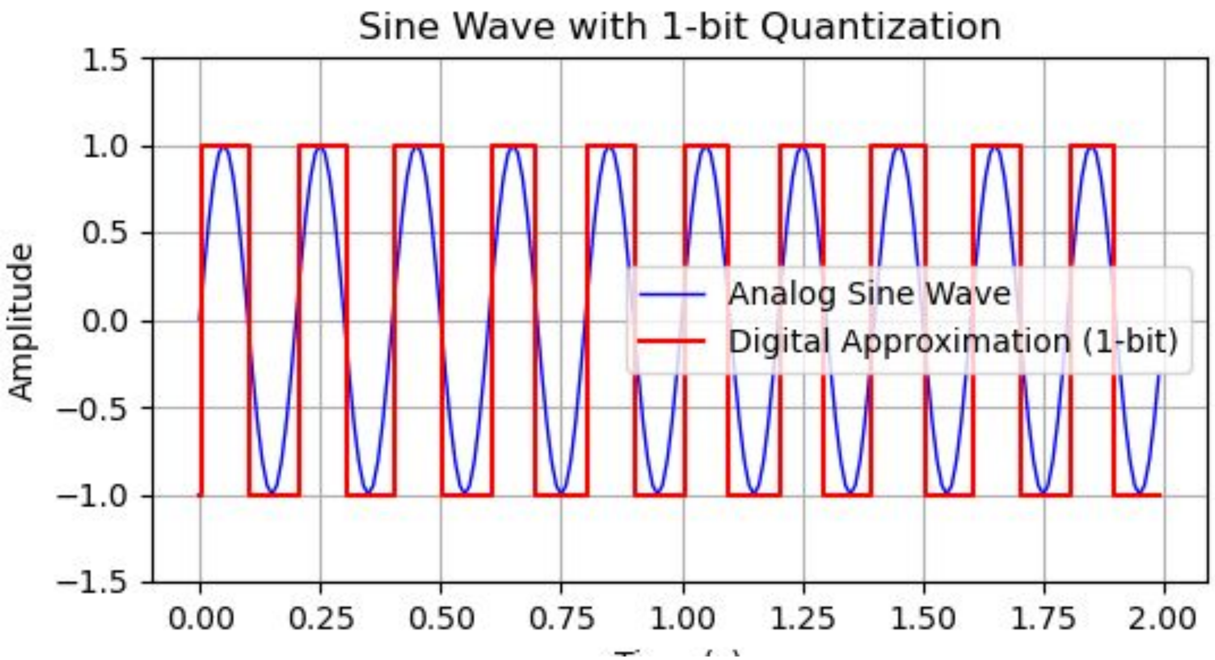
https://www.monolithicpower.com/en/learning/mpscholar/analog-to-digital-converters/introduction-to-adcs/fundamental-concepts

ADC



Principles of Image Printing Technology

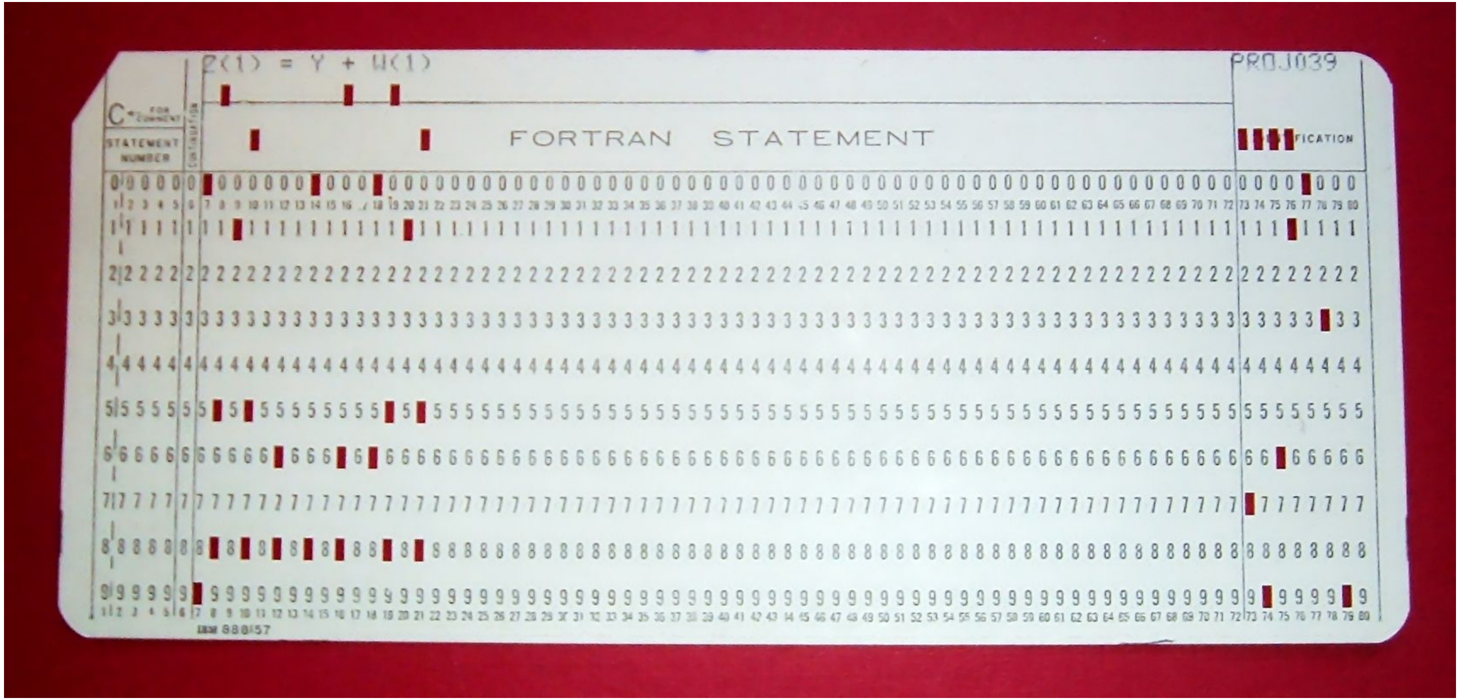
Quantization bit rates



A file

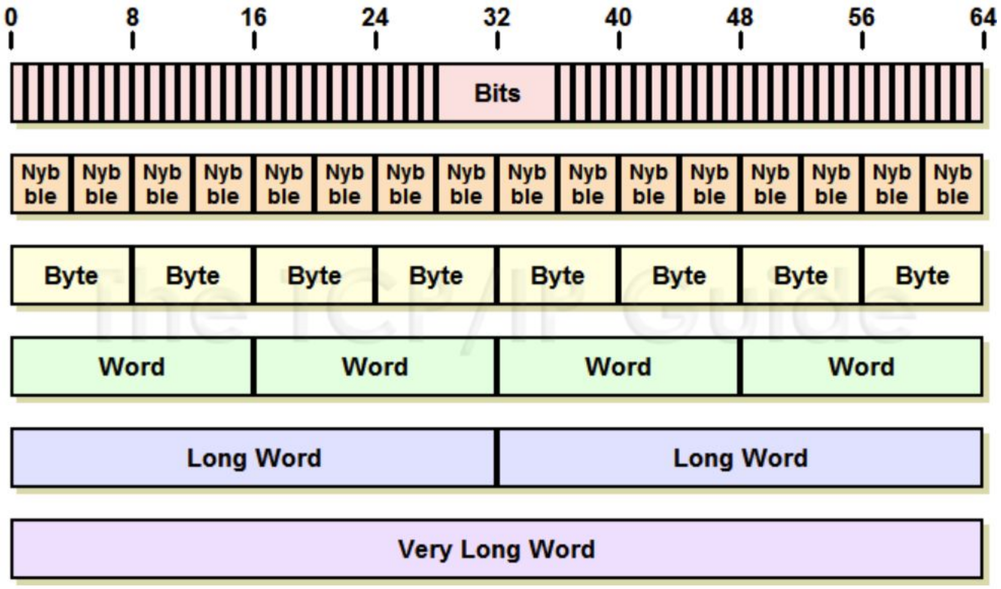
A file

- Entity of content
- Back in the days: punch cards



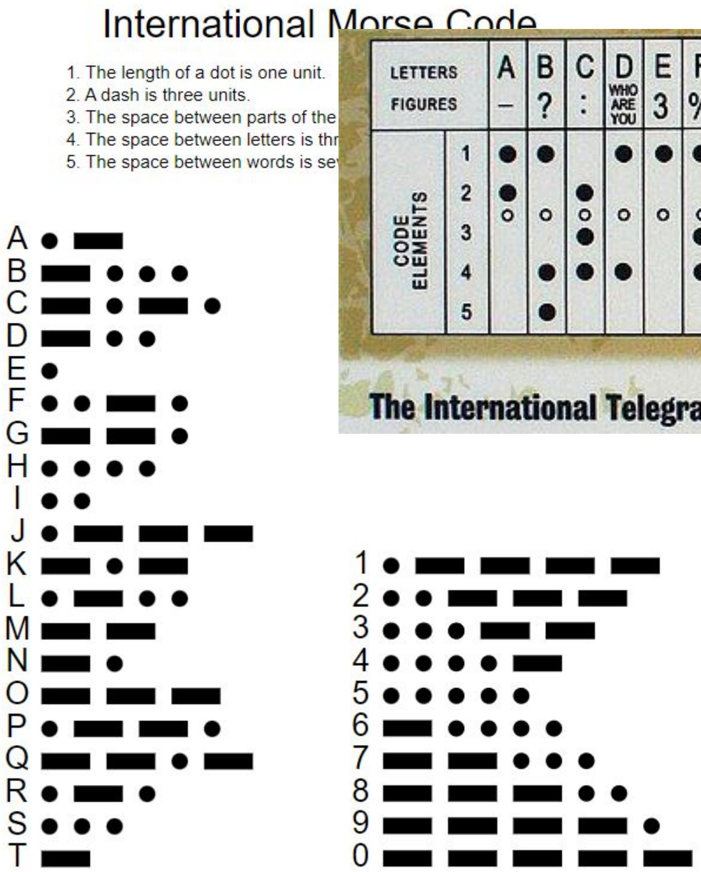
A bit and a byte

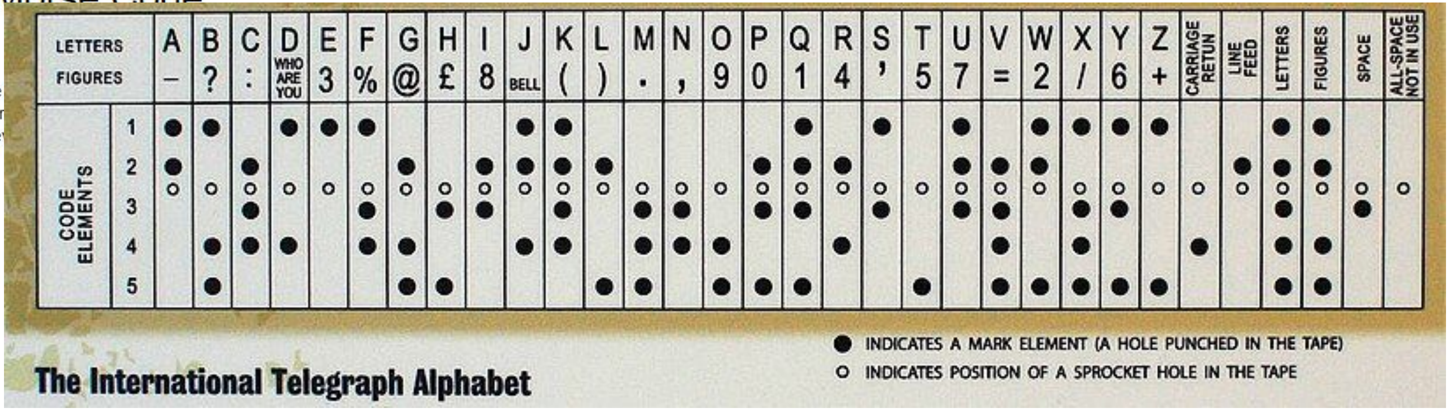
Number of Bits	Common Representation Terms
1	Bit / Digit / Flag
4	Nybble / Nibble
8	Byte / Octet / Character
16	Double Byte / Word
32	Double Word / Long Word
64	Very Long Word



http://www.tcpipguide.com/free/t_BinaryInformationandRepresentationBitsBytesNibbles-2.htm

Morse code to telegraphs





User:Huestones with derivative work by User:TedColes - Old version of

Storing information as bits and bytes

1960s:

ASCII (American Standard Code for Information Interchange) originally designed as 7 bit system (bandwidth and memory limited systems)

Bit #8: error checking, parity

Later repurposed.

USASCII code chart

Bits b_4	b_3	b_2	b_1	Row ↓ / Column → ($b_7 b_6 b_5$)	0 (000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)	6 (110)	7 (111)
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

File identification

- Root/stem ➜ identifier
- Extension ➜ File type
- Path ➜ Location

File root, stem

dir_name

C:\users\OddThinking\Documents\My Source\Widget\foo.src

Basename

Extension

full/absolute path

Unix / (forward slash)

Windows \ (back slash)

https://stackoverflow.com/questions/2235173/what-is-the-naming-standard-for-path-components

File size

- Maybe trivial, but it is measured in bytes
- Remember the 4 GB max file size on FAT**32**?

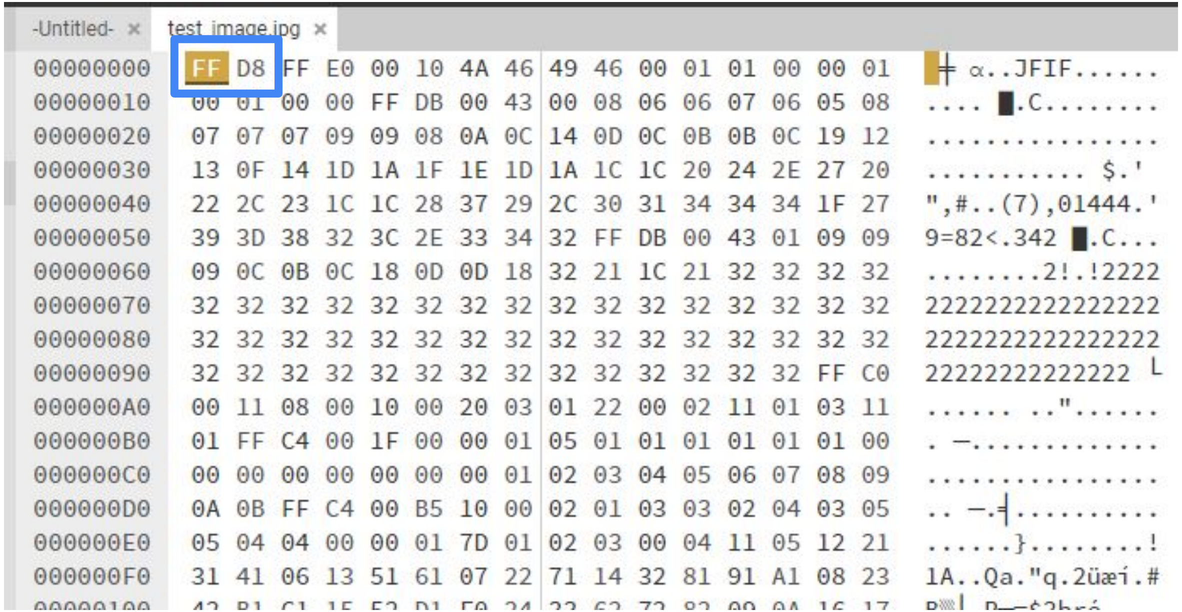
  2^32 - 1 ➜ 4,294,967,295 (2^{32} − 1) bytes, ca 4 GB max

Traditional units				
Name	**Symbol**	**Binary**	**Number of bytes**	**Equal to**
Kilobyte	kB	2^{10}	1,024	1024 B
Megabyte	MB	2^{20}	1,048,576	1024 KB
Gigabyte	GB	2^{30}	1,073,741,824	1024 MB
Terabyte	TB	2^{40}	1,099,511,627,776	1024 GB
Petabyte	PB	2^{50}	1,125,899,906,842,624	1024 TB
Exabyte	EB	2^{60}	1,152,921,504,606,846,976	1024 PB
Zettabyte	ZB	2^{70}	1,180,591,620,717,411,303,424	1024 EB
Yottabyte	YB	2^{80}	1,208,925,819,614,629,174,706,176	1024 ZB

Files' internal metadata

Magic Numbers:

Beginning of file tells you which file type it is!



What can I do with files - in general?

- Create a new file
- Change the access permissions and attributes of a file
- Open a file, which makes the file contents available to the program
- Read data from a file
- Write data to a file
- Delete a file
- Close a file, terminating the association between it and the program
- Truncate a file, shortening it to a specified size within the file system without rewriting any content

File extensions are arbitrary

Extensions help to decipher the file content, but the file needs still to follow the file type's organization.

For example:
Renaming image.**png** to image.**jpg** <u>does not convert the file to the JPG standard.</u>
It has still the SAME CONTENT (--> being a PNG file)

File systems

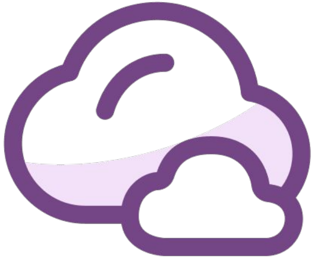
1960s:
IBM's Generalized Sequential Access Method (GSAM) - sequential data processing, efficient for tape-based data

1977:
File Allocation Table (FAT). Introduced w/ DOS and early Windows with an 8-bit table. FAT16 and FAT32 were developed to allow larger volumes and file sizes.

1992:
Extended File System (ext): For Linux and other Unix systems. Ext2-4 for performance improvements.

1993:
New Technology File System (NTFS). For Windows NT and later, after M$ broke up w/ IBM - brought rich metadata, advanced data structures and access control lists

slido



Which file extensions do you know?

ⓘ Click **Present with Slido** or install our [Chrome extension] to activate this poll while presenting.

File types commonly used in Data Science

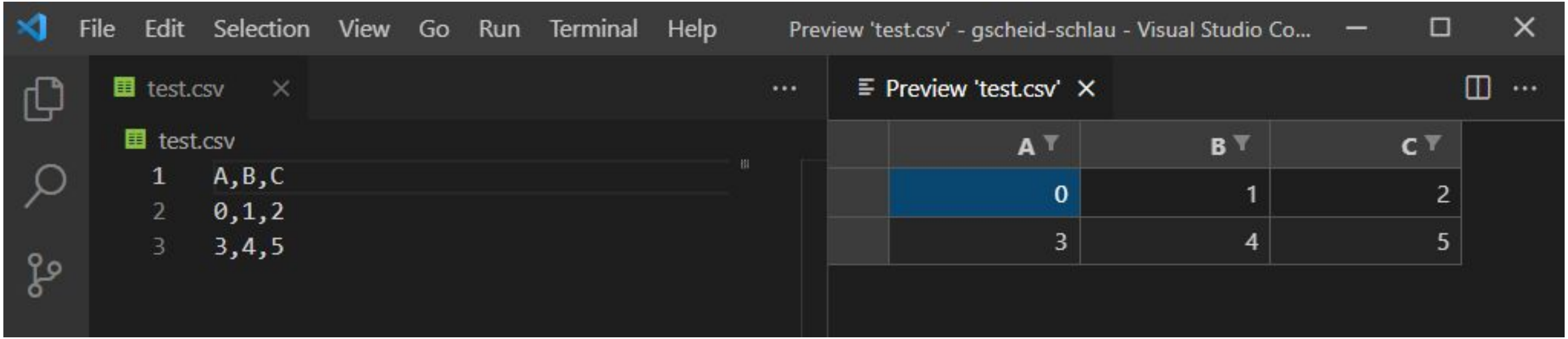
- Plain text (common extensions *.txt, *.csv, *.log, *.json, *.xml) - Python program code!
- Spreadsheets (*.xlsx)
- Word processing files (*.docx)
- Images (*.jpg -> Camera, *.png -> Scientific data, *.tif -> Microscopy)
- Videos (*.avi -> mostly raw data, *.mp4 almost everything, commonly h264 codec)
- Medical imaging data (DICOM, Nifti *.nii and *.nii.gz)
- Vector graphics (*.pdf, *.svg, *.ai)
- Container files (*.hdf5)
- Archives (*.zip, *.tar.gz, *.7z, *.rar)
- Database (*.sqlite)
- Deep Neural Networks (*.pb, *.h5, *.tflite, ...)

Software you should have around

These are EXAMPLES that e.g. work for me. They can be replaced by various other tools. Everything is free except indicated.

- Visual Studio Code (plain text, CSV files, JSON, XML)
- LibreOffice/M$ Office/Google Docs (docx, xlsx, pptx,...)
- FIJI / ImageJ (Microscopy images) and paint.NET (all purpose images)
- VLC (Videos)
- Inkscape (free) or Adobe Illustrator ($$$) (vector graphics)
- 7zip (all kinds of archives)
- HDF5View (HDF5 container files)
- Netron (universal cross-platform deep neural network viewer)

Plain text file



Let's deepdive

How is this file stored?

⇒ HEX Editor

Text file - encoding

Latin-1 (ISO 8859-1) is one-byte encoding, compatible to ASCII

UTF-8 offers 1-4 one-byte encodings, also compatible to ASCII

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	[b]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

UTF-8 encoding process

Character		Binary code point	Binary UTF-8	Hex UTF-8
$	U+0024	010 0100	00100100	24
£	U+00A3	000 1010 0011	11000010 10100011	C2 A3
И	U+0418	100 0001 1000	11010000 10011000	D0 98
ह	U+0939	0000 1001 0011 1001	11100000 10100100 10111001	E0 A4 B9
€	U+20AC	0010 0000 1010 1100	11100010 10000010 10101100	E2 82 AC
한	U+D55C	1101 0101 0101 1100	11101101 10010101 10011100	ED 95 9C
𐍈	U+10348	0 0001 0000 0011 0100 1000	11110000 10010000 10001101 10001000	F0 90 8D 88

Comparison of plain text files

- Older OS did not track how large a file is -
  They used the EOF-tag (end of file)
- Newer OS track how large a file is - no need for EOF
- CR/LF (EOL ➔ \r\n, 0x0D, 0x0A ➔ 13 and 10 in decimal)
  (carriage return, line feed)

  \r ➔ advances to the beginning of the line
  \n ➔ goes to new line

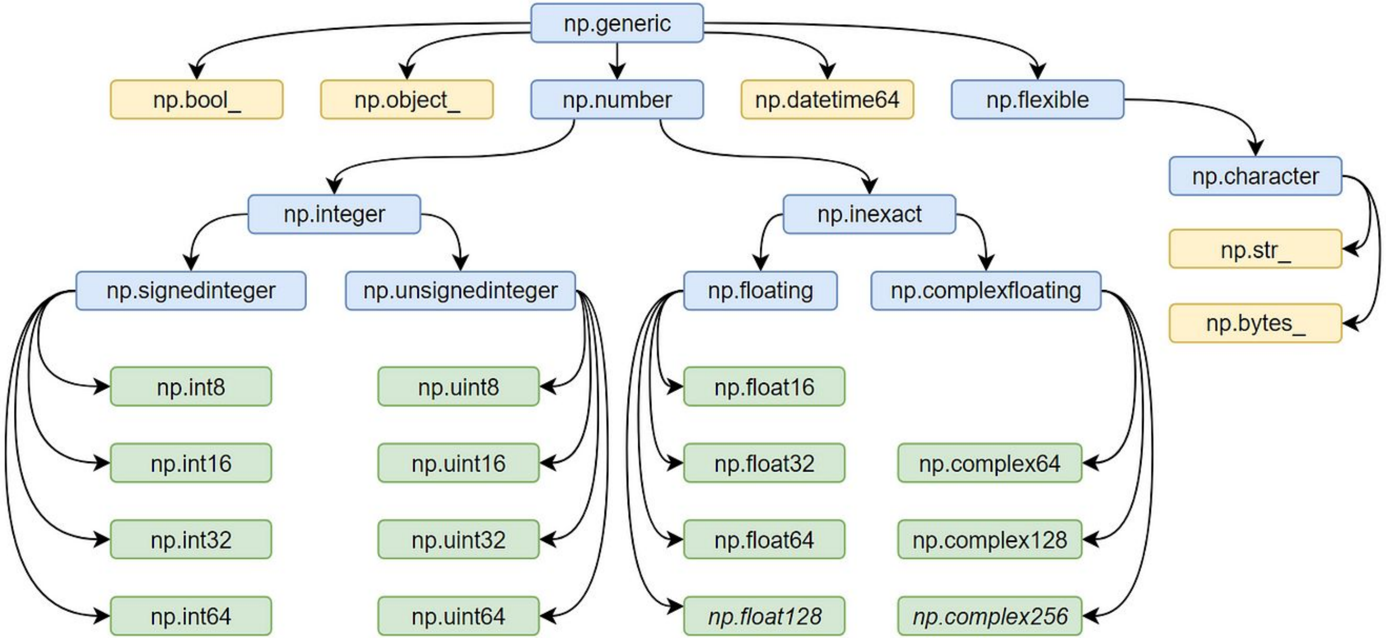
Tabular data

Table I

Iris setosa				*Iris versicolor*				*Iris virginica*			
Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width
5·1	3·5	1·4	0·2	7·0	3·2	4·7	1·4	6·3	3·3	6·0	2·5
4·9	3·0	1·4	0·2	6·4	3·2	4·5	1·5	5·8	2·7	5·1	1·9
4·7	3·2	1·3	0·2	6·9	3·1	4·9	1·5	7·1	3·0	5·9	2·1
4·6	3·1	1·5	0·2	5·5	2·3	4·0	1·3	6·3	2·9	5·6	1·8
5·0	3·6	1·4	0·2	6·5	2·8	4·6	1·5	6·5	3·0	5·8	2·2
5·4	3·9	1·7	0·4	5·7	2·8	4·5	1·3	7·6	3·0	6·6	2·1
4·6	3·4	1·4	0·3	6·3	3·3	4·7	1·6	4·9	2·5	4·5	1·7
5·0	3·4	1·5	0·2	4·9	2·4	3·3	1·0	7·3	2·9	6·3	1·8
4·4	2·9	1·4	0·2	6·6	2·9	4·6	1·3	6·7	2·5	5·8	1·8
4·9	3·1	1·5	0·1	5·2	2·7	3·9	1·4	7·2	3·6	6·1	2·5
5·4	3·7	1·5	0·2	5·0	2·0	3·5	1·0	6·5	3·2	5·1	2·0
4·8	3·4	1·6	0·2	5·9	3·0	4·2	1·5	6·4	2·7	5·3	1·9
4·8	3·0	1·4	0·1	6·0	2·2	4·0	1·0	6·8	3·0	5·5	2·1
4·3	3·0	1·1	0·1	6·1	2·9	4·7	1·4	5·7	2·5	5·0	2·0
5·8	4·0	1·2	0·2	5·6	2·9	3·6	1·3	5·8	2·8	5·1	2·4
5·7	4·4	1·5	0·4	6·7	3·1	4·4	1·4	6·4	3·2	5·3	2·3
5·4	3·9	1·3	0·4	5·6	3·0	4·5	1·5	6·5	3·0	5·5	1·8
5·1	3·5	1·4	0·3	5·8	2·7	4·1	1·0	7·7	3·8	6·7	2·2
5·7	3·8	1·7	0·3	6·2	2·2	4·5	1·5	7·7	2·6	6·9	2·3
5·1	3·8	1·5	0·3	5·6	2·5	3·9	1·1	6·0	2·2	5·0	1·5
5·4	3·4	1·7	0·2	5·9	3·2	4·8	1·8	6·9	3·2	5·7	2·3
5·1	3·7	1·5	0·4	6·1	2·8	4·0	1·3	5·6	2·8	4·9	2·0
4·6	3·6	1·0	0·2	6·3	2·5	4·9	1·5	7·7	2·8	6·7	2·0
5·1	3·3	1·7	0·5	6·1	2·8	4·7	1·2	6·3	2·7	4·9	1·8
4·8	3·4	1·9	0·2	6·4	2·9	4·3	1·3	6·7	3·3	5·7	2·1
5·0	3·0	1·6	0·2	6·6	3·0	4·4	1·4	7·2	3·2	6·0	1·8
5·0	3·4	1·6	0·4	6·8	2·8	4·8	1·4	6·2	2·8	4·8	1·8
5·2	3·5	1·5	0·2	6·7	3·0	5·0	1·7	6·1	3·0	4·9	1·8
5·2	3·4	1·4	0·2	6·0	2·9	4·5	1·5	6·4	2·8	5·6	2·1
4·7	3·2	1·6	0·2	5·7	2·6	3·5	1·0	7·2	3·0	5·8	1·6
4·8	3·1	1·6	0·2	5·5	2·4	3·8	1·1	7·4	2·8	6·1	1·9
5·4	3·4	1·5	0·4	5·5	2·4	3·7	1·0	7·9	3·8	6·4	2·0
5·2	4·1	1·5	0·1	5·8	2·7	3·9	1·2	6·4	2·8	5·6	2·2
5·5	4·2	1·4	0·2	6·0	2·7	5·1	1·6	6·3	2·8	5·1	1·5
4·9	3·1	1·5	0·2	5·4	3·0	4·5	1·5	6·1	2·6	5·6	1·4
5·0	3·2	1·2	0·2	6·0	3·4	4·5	1·6	7·7	3·0	6·1	2·3
5·5	3·5	1·3	0·2	6·7	3·1	4·7	1·5	6·3	3·4	5·6	2·4
4·9	3·6	1·4	0·1	6·3	2·3	4·4	1·3	6·4	3·1	5·5	1·8
4·4	3·0	1·3	0·2	5·6	3·0	4·1	1·3	6·0	3·0	4·8	1·8
5·1	3·4	1·5	0·2	5·5	2·5	4·0	1·3	6·9	3·1	5·4	2·1
5·0	3·5	1·3	0·3	5·5	2·6	4·4	1·2	6·7	3·1	5·6	2·4
4·5	2·3	1·3	0·3	6·1	3·0	4·6	1·4	6·9	3·1	5·1	2·3
4·4	3·2	1·3	0·2	5·8	2·6	4·0	1·2	5·8	2·7	5·1	1·9
5·0	3·5	1·6	0·6	5·0	2·3	3·3	1·0	6·8	3·2	5·9	2·3
5·1	3·8	1·9	0·4	5·6	2·7	4·2	1·3	6·7	3·3	5·7	2·5
4·8	3·0	1·4	0·3	5·7	3·0	4·2	1·2	6·7	3·0	5·2	2·3
5·1	3·8	1·6	0·2	5·7	2·9	4·2	1·3	6·3	2·5	5·0	1·9
4·6	3·2	1·4	0·2	6·2	2·9	4·3	1·3	6·5	3·0	5·2	2·0
5·3	3·7	1·5	0·2	5·1	2·5	3·0	1·1	6·2	3·4	5·4	2·3
5·0	3·3	1·4	0·2	5·7	2·8	4·1	1·3	5·9	3·0	5·1	1·8

Iris dataset, Fisher 1939

Data types

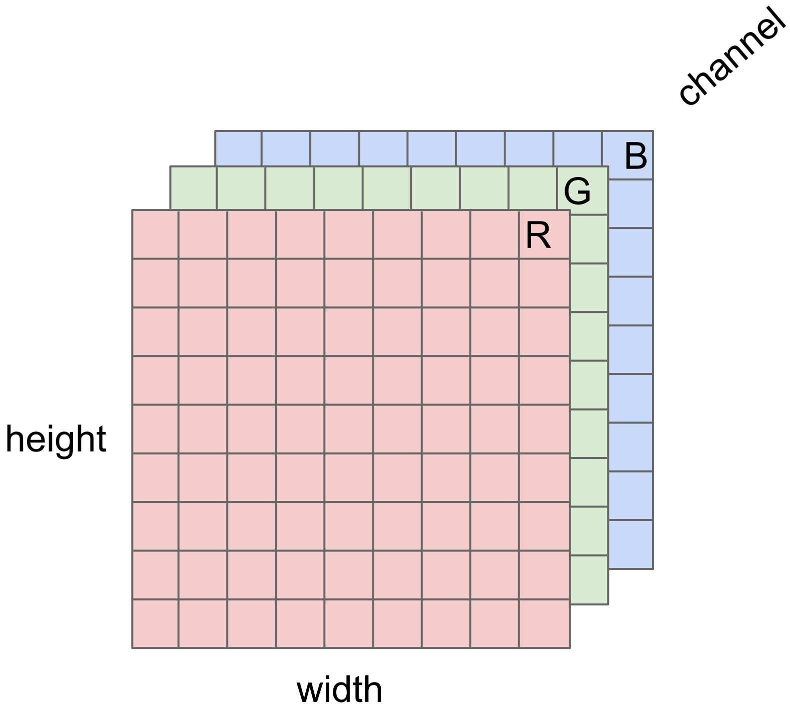


https://betterprogramming.pub/a-comprehensive-guide-to-numpy-data-types-8f62cb57ea83?gi=4d56b0703884

Datatypes


IEEE half-precision 16-bit float
sign
exponent (5 bit)
fraction (10 bit)
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0
15 14 10 9 0
IEEE 754 single-precision 32-bit float
sign
exponent (8 bit)
fraction (23 bit)
0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
31 30 23 22 0
bfloat16
sign
exponent (8 bit)
fraction (7 bit)
0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0
15 14 7 6 0
NVidia's TensorFloat
sign
exponent (8 bit)
fraction (10 bit)
0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0
18 17 10 9 0
AMD's fp24 format
sign
exponent (7 bit)
fraction (16 bit)
0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23 22 16 15 0
Pixar's PXR24 format
sign
exponent (8 bit)
fraction (15 bit)
0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23 22 15 14 0

An image consists of many pixels



<u>Very common</u>:

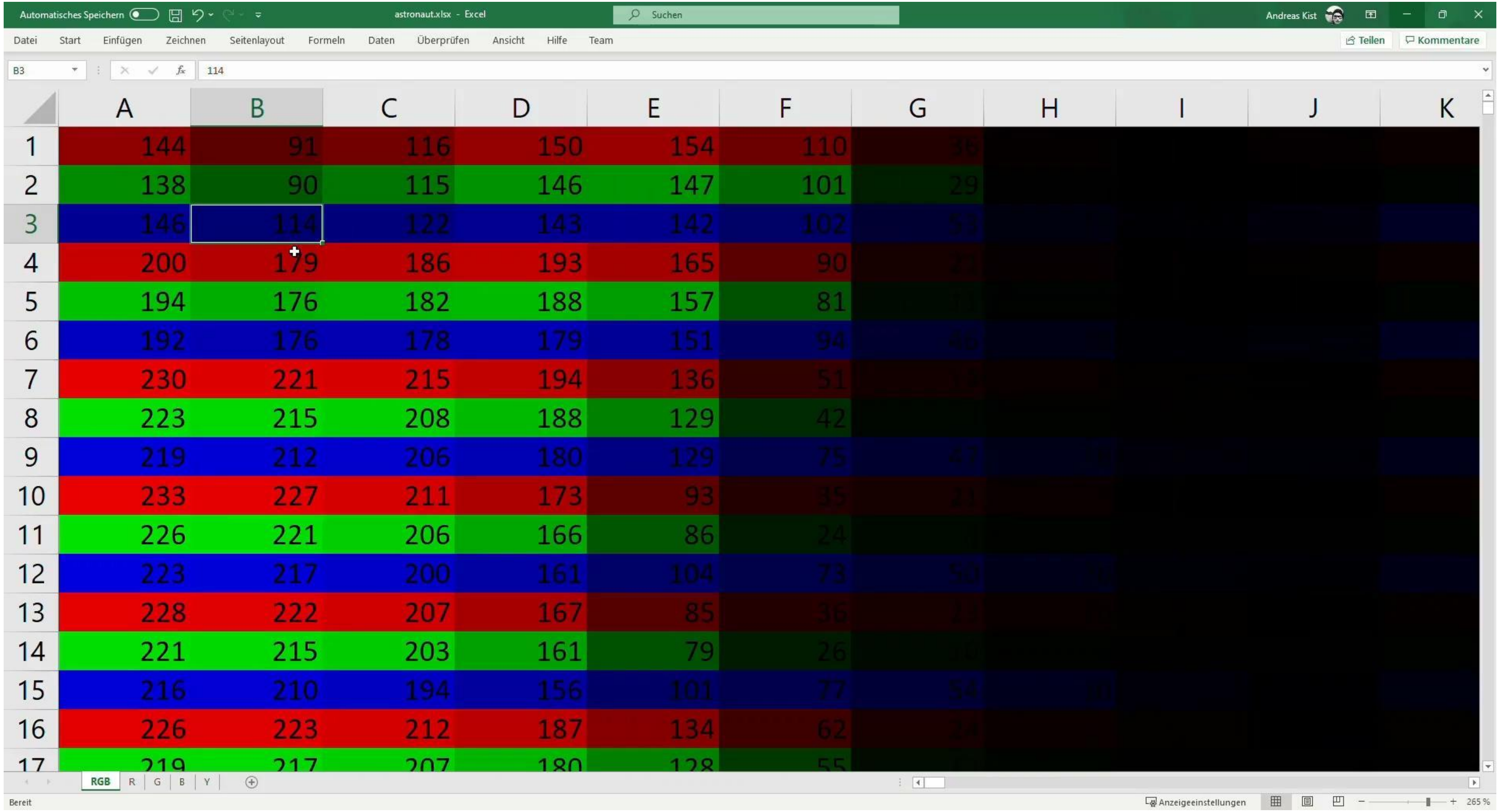
RGB (height x width x channels ⇒ HxWx3)
RGBA (HxWx4, last channel is alpha ⇔ transparency)
Monochrome (HxWx1 ⇒ HxW)

<u>Microscopy data:</u>

HxWxC,
where C is e.g. DAPI, GFP, Alexa488, mCherry, ….

E.g. an image of HxWxC = 256x256x3,
has 256x256x3 = 196,608 units, that we call **pixels**!

Images are just Excel sheets



	A	B	C	D	E	F	G	H	I	J	K
1	144	91	116	150	154	110	36				
2	138	90	115	146	147	101	29				
3	146	114	122	143	142	102	53				
4	200	179	186	193	165	90	21				
5	194	176	182	188	157	81	[illegible]				
6	192	176	178	179	151	94	46				
7	230	221	215	194	136	51	[illegible]				
8	223	215	208	188	129	42	[illegible]				
9	219	212	206	180	129	75	47				
10	233	227	211	173	93	35	21				
11	226	221	206	166	86	24	[illegible]				
12	223	217	200	161	104	73	50				
13	228	222	207	167	85	36	[illegible]				
14	221	215	203	161	79	26	[illegible]				
15	216	210	194	156	101	77	54				
16	226	223	212	187	134	62	[illegible]				
17	219	217	207	180	128	55	[illegible]				

Storing information efficiently

Example: WWII

The war is over (8 bit * 15 characters = 120 bits)
The war is not over (8 bit * 19 characters = 152 bits)

Information can be reduced to **1 (!) bit** (either we won or we didn't)

Formalize with Shannon entropy:

$$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)], \tag{3.49}$$

Expected value of information I(x)

Log base 2: bits,
Base e: nats,
Base 10: dits or bans

Deeper...

$$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)], \tag{3.49}$$

I(x) is **the information content of X**.

I(x) itself is **a random variable.** In our example, the possible outcomes of the War. Thus, **H(x)** is **the expected value of every possible information.**

Encoding

Transmitting 4 characters: A, B, C and D

Everyone is equally likely (i.e. 25%) ➔ transmitting H(X)=2 bit (00, 01, 10, 11).

Change the likelihood: A=70%, B=26%, C and D=2%

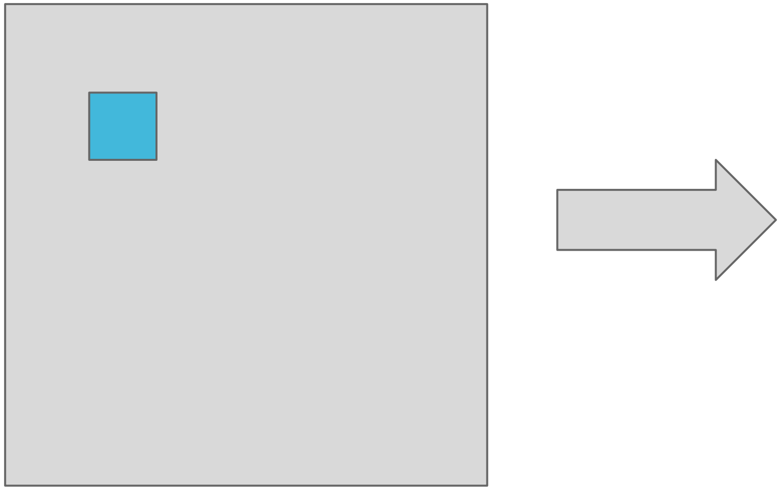
Do the math,... H(x) = 1.0881 bit

A: 0, B: 10, C: 110, D: 111

=> **Efficient encoding ensures efficient transmission**

Compression

Increasing entropy! Removing redundant information!



Rectangle size,
Blue rectangle size and location

Compression algorithms

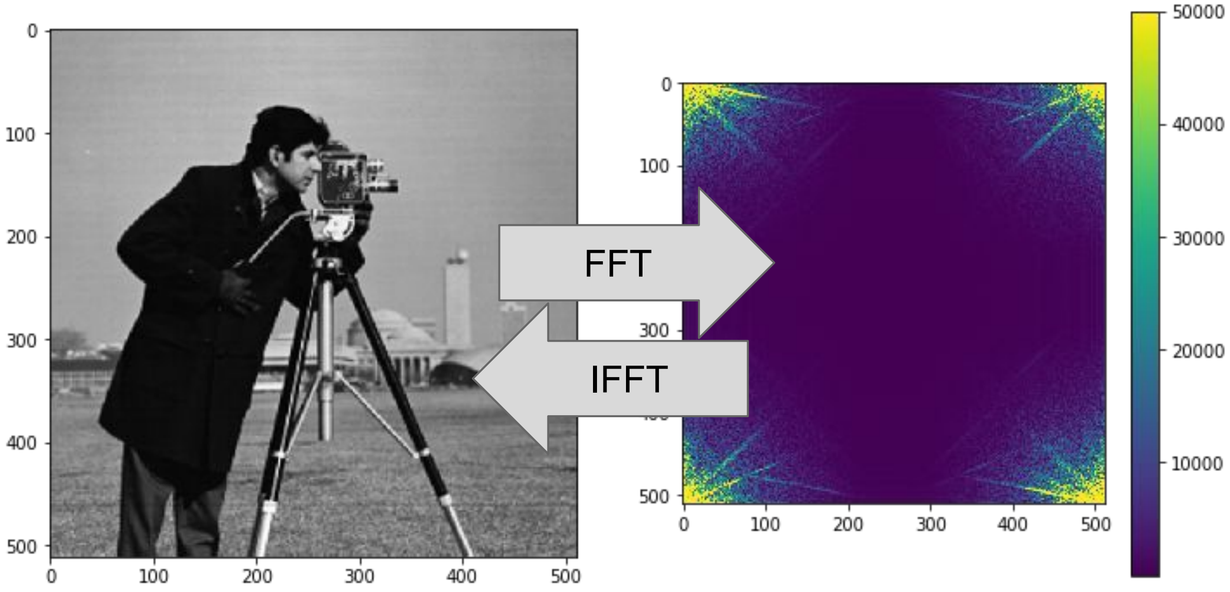
LOSSY

E.g. Discrete Cosine Transformations
As in JPEG files or MP3 files

LOSSLESS

E.g. ZIP/7z files, PNG files

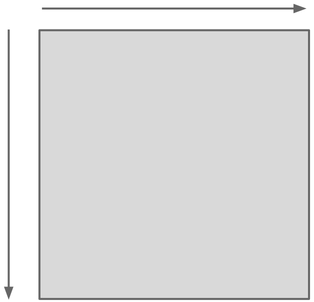
Images in Fourier space

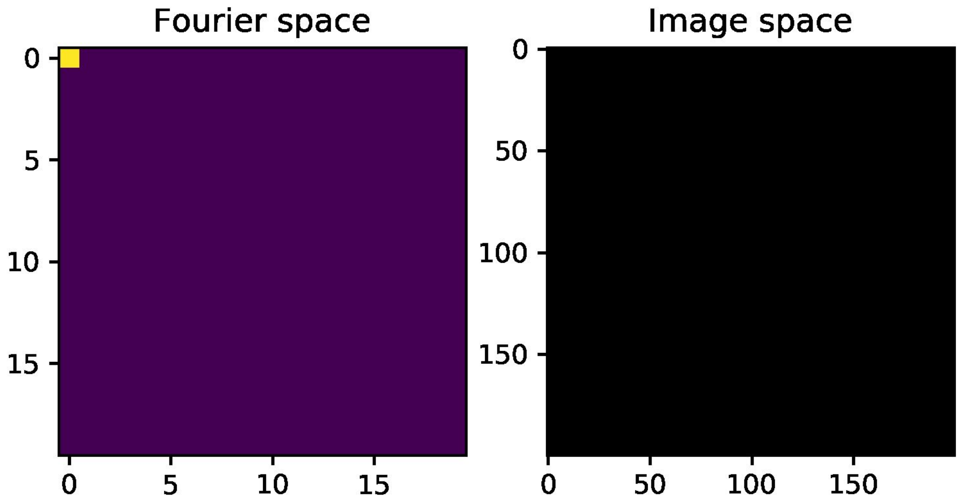


What is exactly happening?

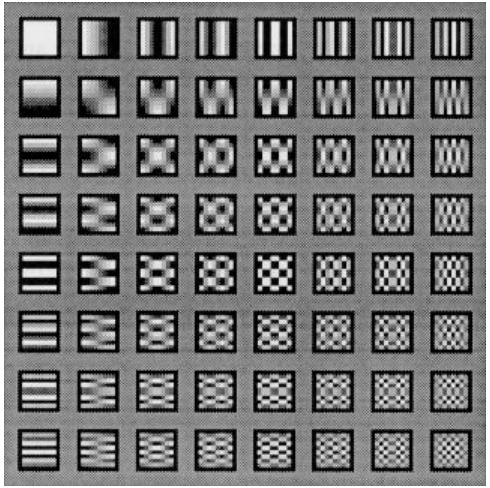
Frequency, vertical

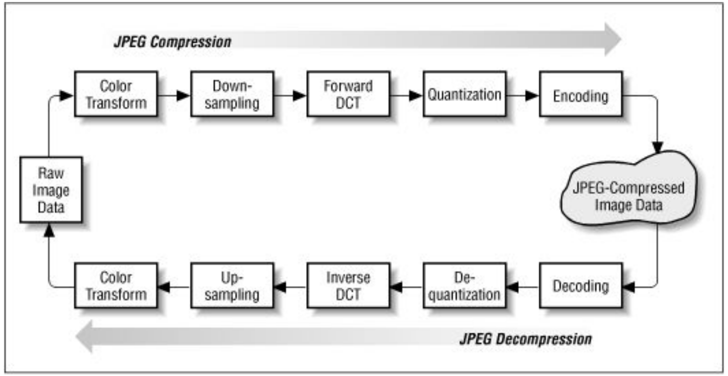
Frequency, horizontal

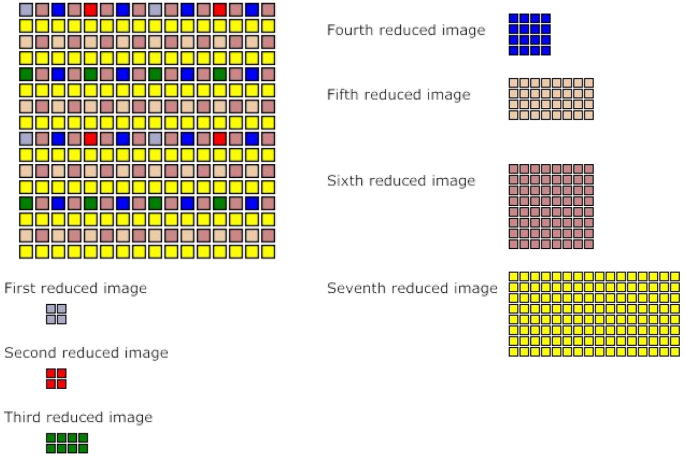




Lossy and lossless compression



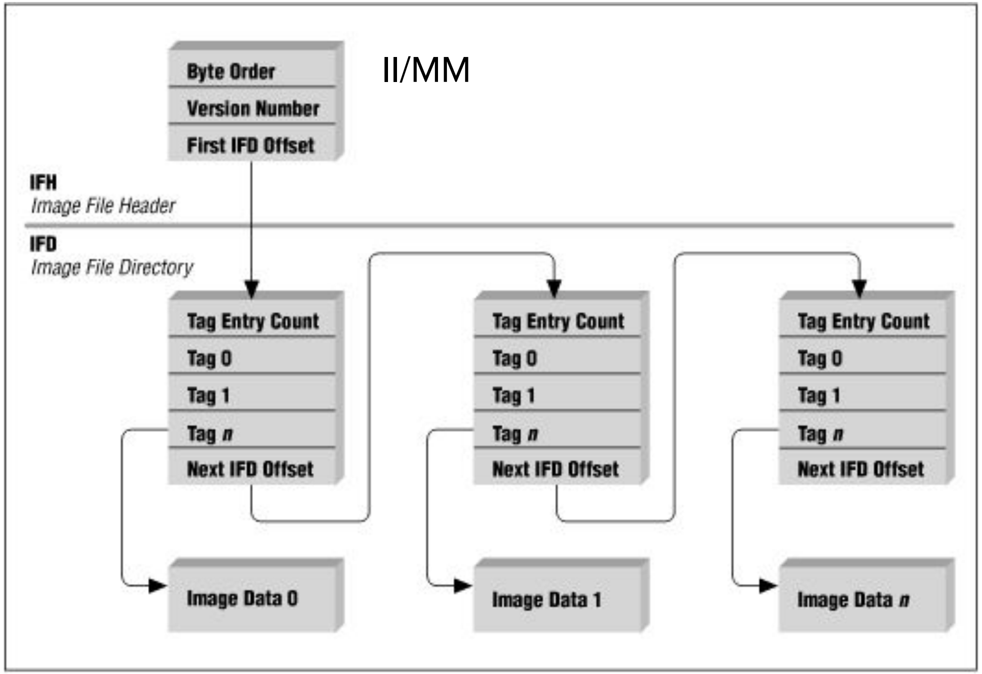




PNG: LZ77-based lossless compression (LUT, +Huffmann encoding)

The TIF file format header

Some files need more information, such as bit depth of an image (8 bit, 16 bit), color or grayscale, size of the image etc.



Images in a scientific environment

TIFF

- Saves raw data
- Multiple channels
- Multiple bit depth levels

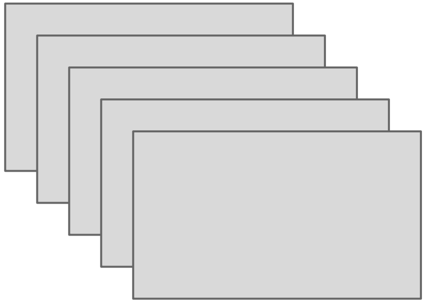
PNG

- Lossless compression
- Up to 4 channels (RGBA)

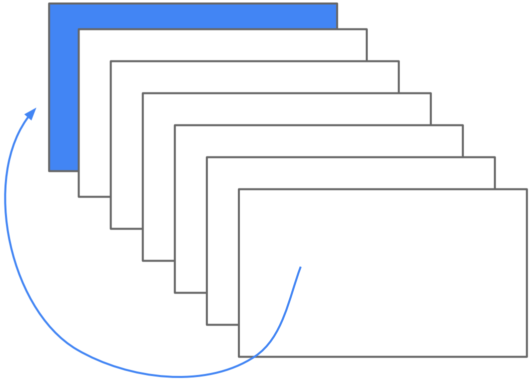
JPG

- Lossy compression
- Fine for photography
- Compression artifacts

Videos

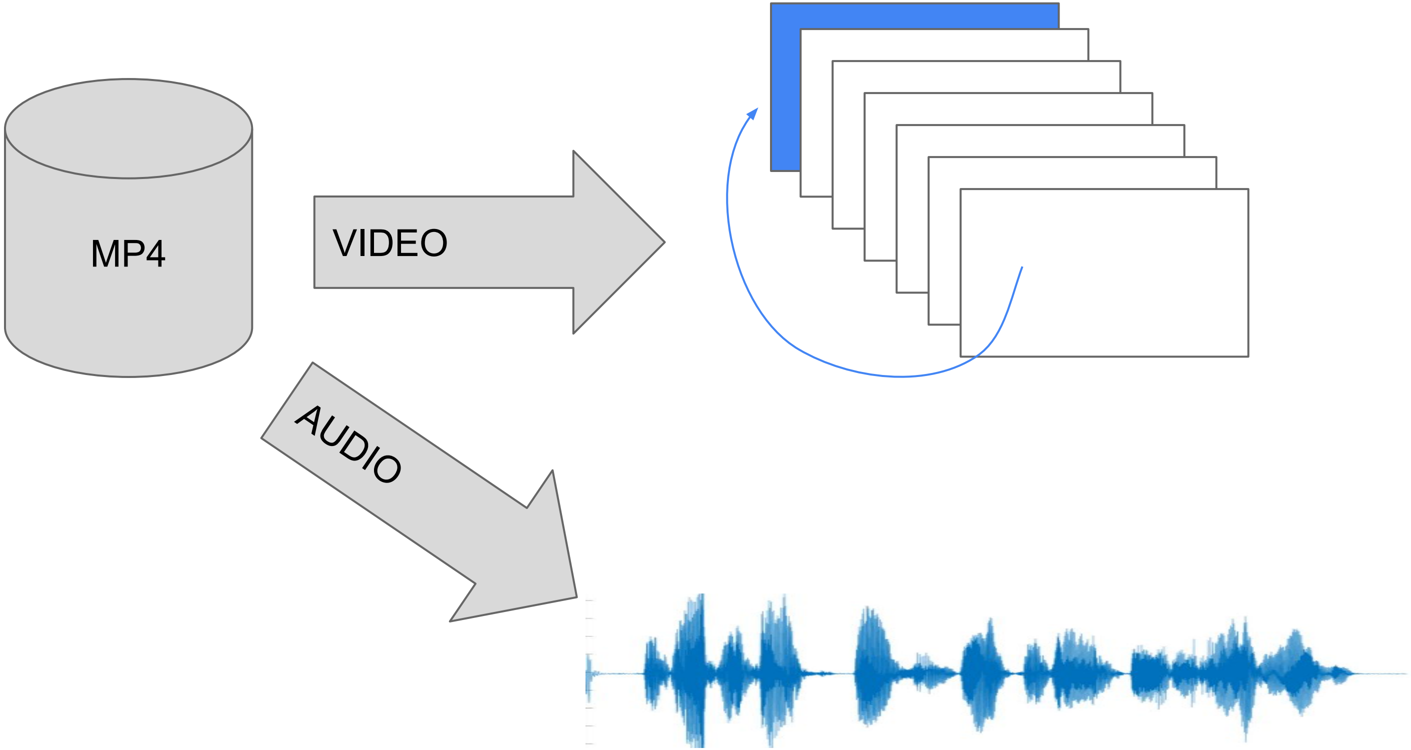


WAY 1: Store each frame one after another,
each frame is independent



WAY 2: Store **key frames** and then store only the
difference relative to the key frames

H264 codec in MP4 container



H264

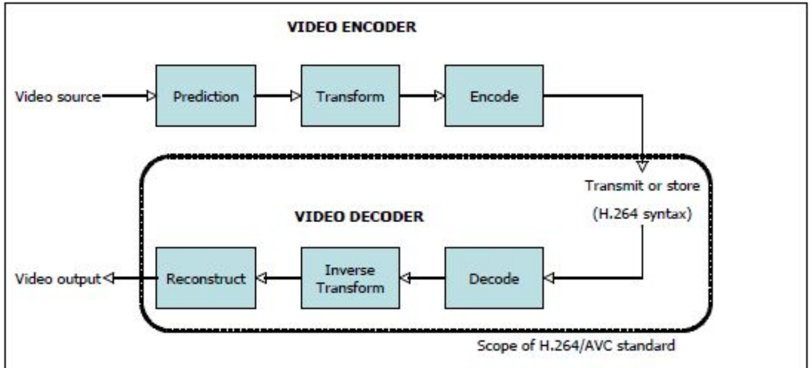


Figure 1 The H.264 video coding and decoding process

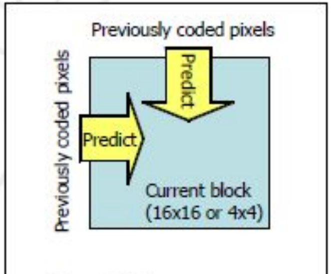


Figure 2 Intra prediction

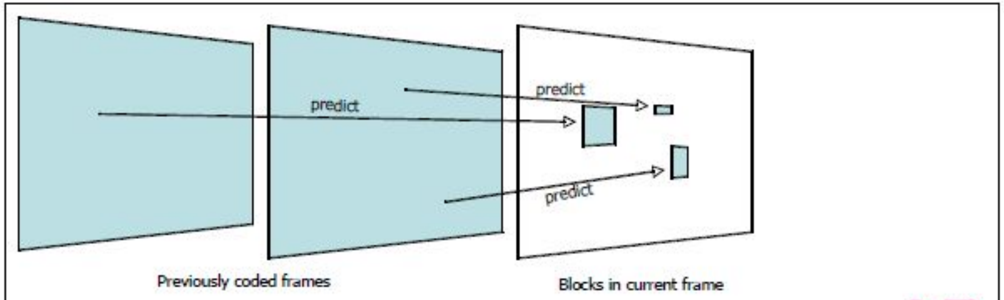


Figure 3 Inter prediction

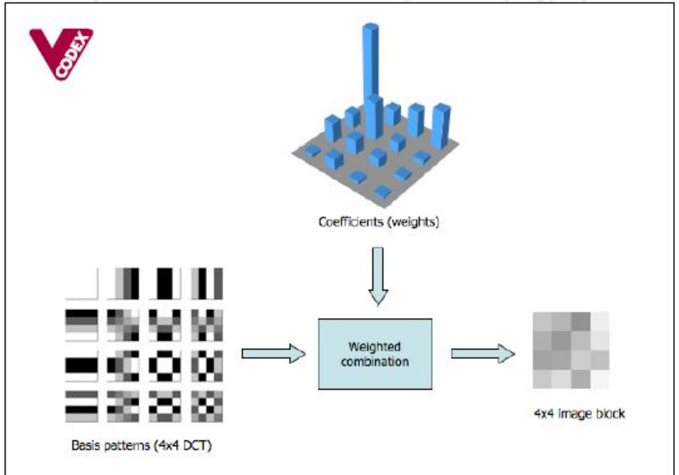


Figure 4 Inverse transform: combining weighted basis patterns to create a 4x4 image block

H264 performance

Figure 5 A video frame compressed at the same bitrate using MPEG-2 (left), MPEG-4 Visual (centre) and H.264 compression (right)

H264 is a great encoder, however, with the default settings you encode your data **lossy**!

LOSSLESS!!!

```
np.random.seed(42)
# Random video
ims_in = (np.random.randn(200, 256, 256, 3) ** 2).astype(np.uint8)

io.mimwrite("file.mp4",
            ims_in, # images
            codec='libx264rgb', # use the right codec
            pixelformat='rgb24', # and pixel format
            output_params=['-crf', '0', # Ensure setting crf to 0
                           '-preset', 'ultrafast']) # Maximum compression: veryslow,
                                                    # maximum speed: ultrafast

ims_out = io.mimread("file.mp4")
np.allclose(ims_in, ims_out)
# True
```

anki-xyz / lossless Public

Storing in mp4 is convenient for sharing and inspection using VLC

“New” kids on the block

H.264

HEVC

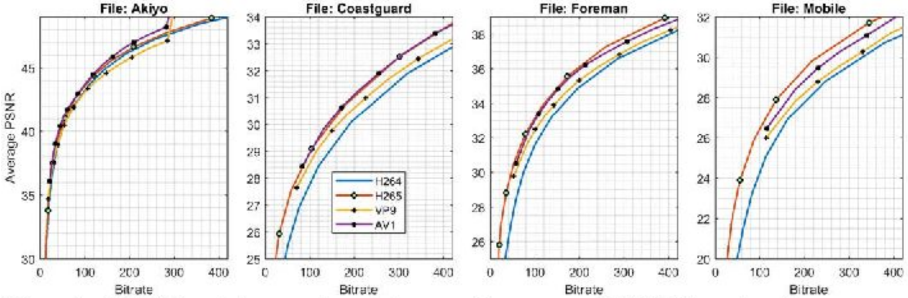


Fig. 8: PSNR with varying bitrates in case of CRF level adjustment (*placebo* presets for H.264 and H.265)

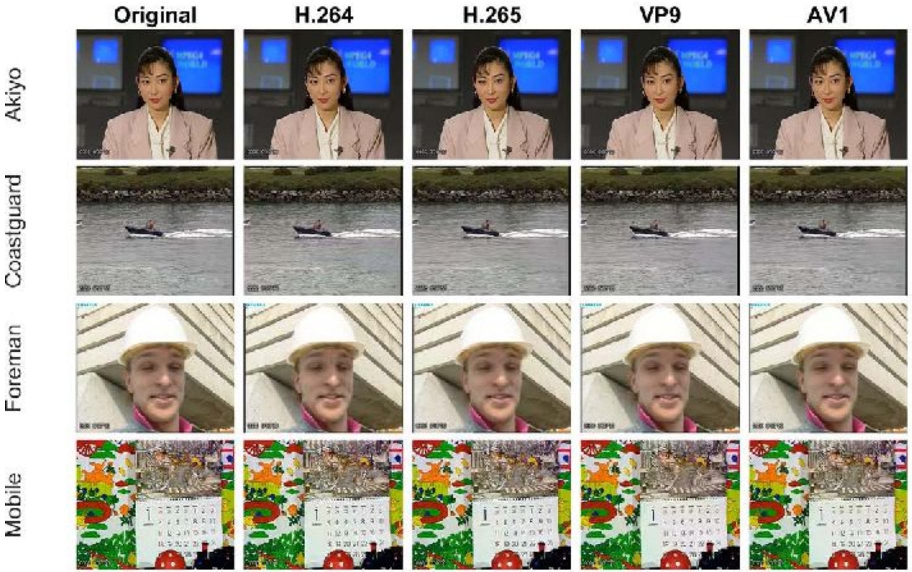
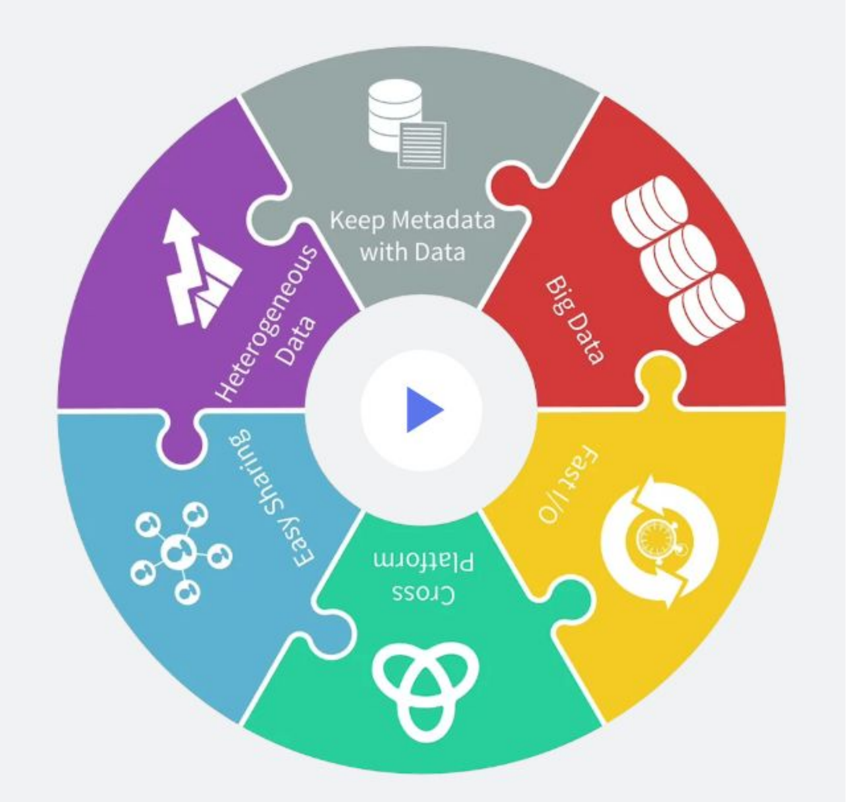


Fig. 9: First frames of the originals and the encoded videos at the

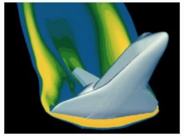
Layek, Md. Abu et al. “Performance analysis of H.264, H.265, VP9 and AV1 video encoders.” *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (2017): 322-325.

HDF5 - the universal file container



Scientific Fields

Astronomy



Computational Fluid Dynamics

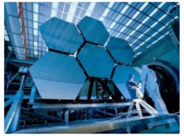
Earth Sciences

Engineering

Finance

Genomics

Medicine



Physics

Hierachical Data Format 5 (HDF5)

Groups: Similar to directories in a file system, groups contain sets of related data and can be nested within each other to create a hierarchical organization.

Datasets: The actual data arrays, similar to files in a file system. A dataset consists of metadata and raw data, and it can be of various multidimensional array types.

Datatypes: Define the nature of the data in the datasets, such as integer, float, or string.

Attributes: Metadata that can be attached to groups and datasets to describe the contained data.

Space: Describes the size and shape of the data array.

Why should you consider HDF5 files?

Scalability: It can store and organize massive volumes of data in a compact and efficient manner.

Flexibility: It supports a wide variety of data types and is capable of handling both homogeneous and heterogeneous data in a single file.

Portability: HDF5 files are self-describing, allowing them to be transferred easily between different types of computers, operating systems, and applications without compatibility issues.

Efficiency: HDF5 provides efficient data I/O by allowing users to read and write subsets of data without having to access the entire dataset.

Rich Metadata Support: Users can store detailed metadata in attributes, making it easier to track and manage complex data.

Support for Parallel I/O: It's designed to support high-performance computing, allowing for parallel I/O which is essential in processing large-scale data efficiently.

Zarr

Zarr is a format for the storage of chunked, compressed, N-dimensional arrays inspired by hdf5

Highlights

- Create N-dimensional arrays with any NumPy dtype.
- **Chunk** arrays along any dimension.
- Compress and/or filter chunks using any NumCodecs codec.
- Store arrays in memory, on disk, inside a Zip file, on S3, …
- Read an array concurrently from multiple threads or processes.
- Write to an array concurrently from multiple threads or processes.
- Organize arrays into hierarchies via groups.

Feature	HDF5	Zarr
File Format	Binary format	Binary format
API Language Support	**Available in C, C++, Python, Java, etc.**	Available in Python, NumPy, and more
Scalability	High for single-node (single file)	**Designed for parallel, distributed systems**
Data Type Support	**Supports complex data types and metadata**	Supports NumPy-like data types and multi-dimensional arrays
Compression	Supports various compression algorithms (e.g. GZIP, LZW)	Supports NumPy’s built-in compression algorithms and custom ones
Chunking	Yes, allows efficient I/O access	**Yes, includes flexible chunking**
Storage Options	Typically file-based	File-based, cloud storage, and remote storage
Concurrency	Limited support for concurrent access	Optimized for multi-threading and distributed access
Ease of Use	More complex due to extensive features	**Generally simpler to use, especially with NumPy**
Data Integrity	Supports checksums for data integrity	Supports checksums; designed with data integrity in mind
Community Support	Well-established and widely-used	Growing community, particularly in the data science field
Versioning	No in-built versioning	Supports versioning, making it suitable for machine learning workflows
Compatibility	Supported by many scientific computing tools	Designed to be compatible with many data science tools (e.g., Dask, Xarray)
Interoperability	Good, many libraries support HDF5	Excellent, especially with modern data science libraries
Use Cases	Scientific computing, MATLAB interchange	Big data, cloud computing, machine learning, and AI

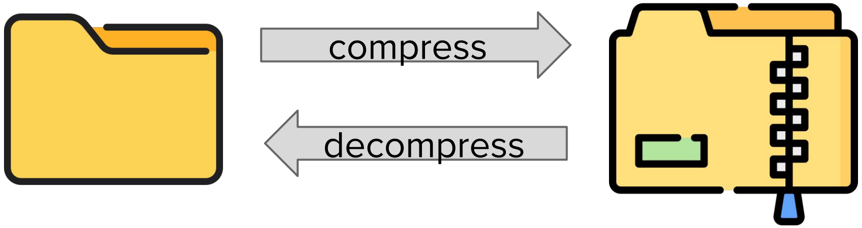
Compression

Numcodecs

Numcodecs is a Python package providing buffer compression and transformation codecs for use in data storage and communication applications. These include:

- Compression codecs, e.g., Zlib, BZ2, LZMA, ZFPY and Blosc.
- Pre-compression filters, e.g., Delta, Quantize, FixedScaleOffset, PackBits, Categorize.
- Integrity checks, e.g., CRC32, Adler32.

Compression



Factors:
Time to compress,
time to decompress/unpack,
compression ratio

$$\text{Compression ratio} = \frac{\text{Original file size}}{\text{File size packed}}$$

- Compression Ratio > 1: Indicates that the data has been compressed (i.e., the compressed size is smaller than the original size).
- Compression Ratio = 1: Indicates no compression has occurred (the size remains the same).
- Compression Ratio < 1: This would indicate the compressed size is larger than the original (unusual for lossless compression algorithms, but can occur in certain scenarios or with lossy compression).

Compressing redundant information

Run-length encoding (RLE)

```
aaaabbbaaaawwope
```

↓

```
a5b3a4w2o1p1e1
```

Run-Length Encoding (RLE): RLE compresses data by replacing sequences of repeating elements with a single element and a count, efficient for data with long runs of identical values.

LZ77: LZ77 uses a sliding window technique to replace repeated occurrences of data with references to the previous occurrence within a fixed-length window, ideal for text with repeating substrings.

LZW: LZW builds a dictionary of recurring data sequences, replacing repeated patterns with dictionary indices, effective for a variety of data types without needing a predefined dictionary.

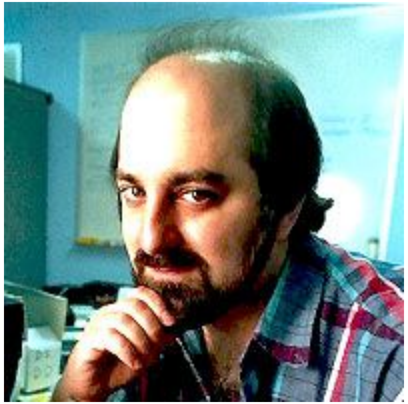
Huffmann encoding

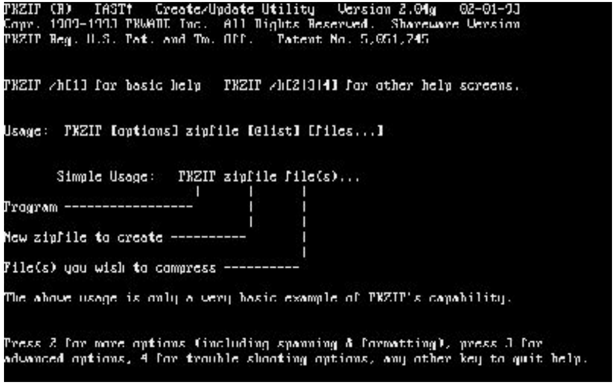
Huffman encoding is a **compression algorithm** that **assigns shorter binary codes to more frequent characters** and longer codes to less frequent ones, creating a variable-length encoding scheme that reduces overall file size. **By constructing a binary tree where each character's frequency determines** its position, Huffman encoding ensures no code is a prefix of another, enabling efficient and lossless data reconstruction.

David A. Huffmann; PhD student at MIT, published 1952

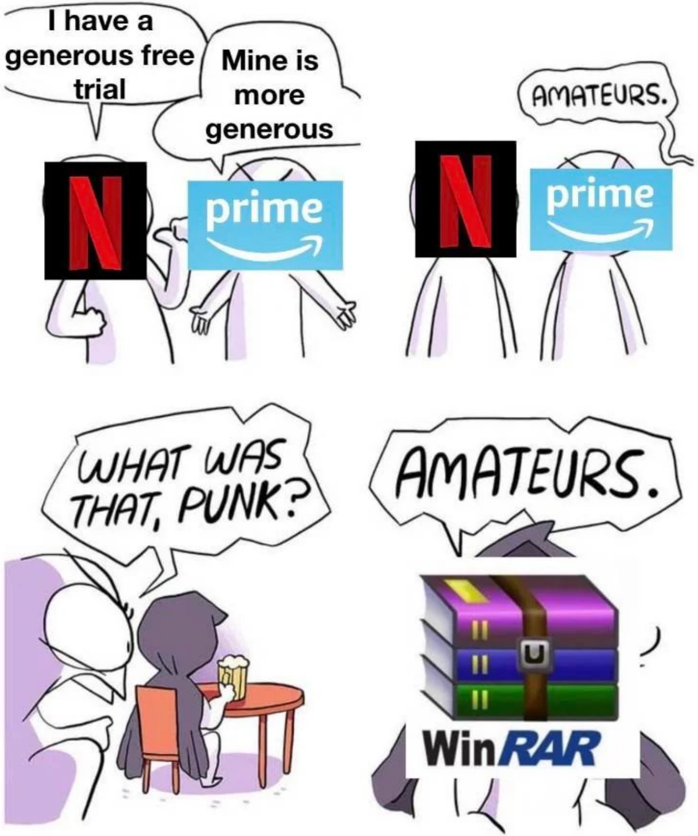
DEFLATE

- Developed by Phil Katz (PK)
- Used in ZIP implementation
- DEFLATE => LZ77 + Huffmann encoding

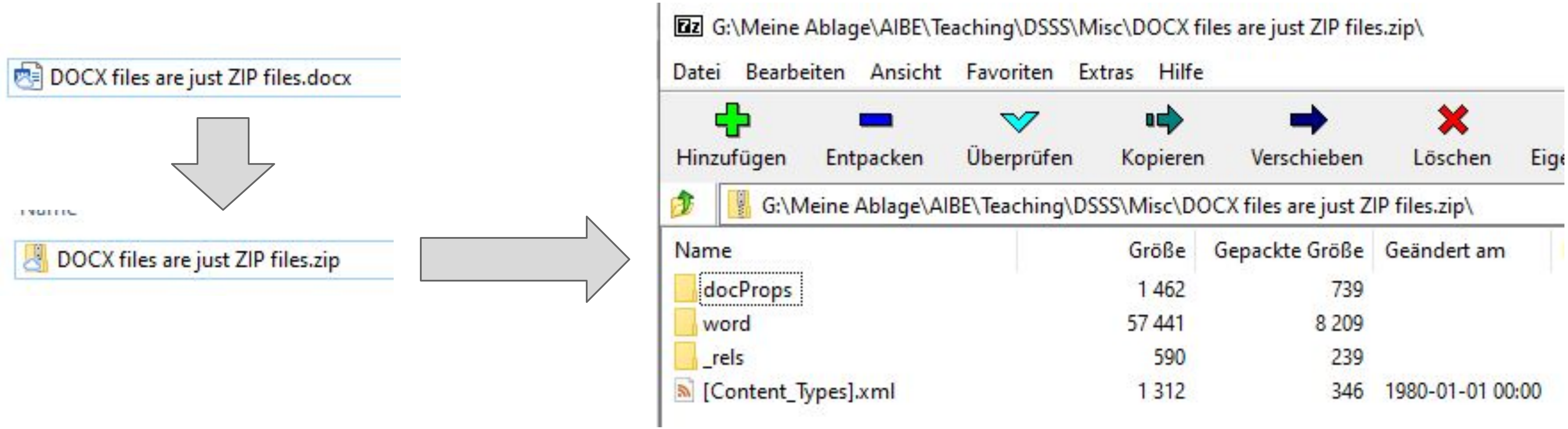




WinZip, WinRAR, 7zip



Fun fact: DOCX files are just ZIP files…



Homework

In this week's lecture, we covered some kinds of data files and talked about datasets. You will have to work now with a mini version of the Benchmark for Automatic Glottis Segmentation ([BAGLS]) dataset. After that, the task is to convert an image from RGB to grayscale.