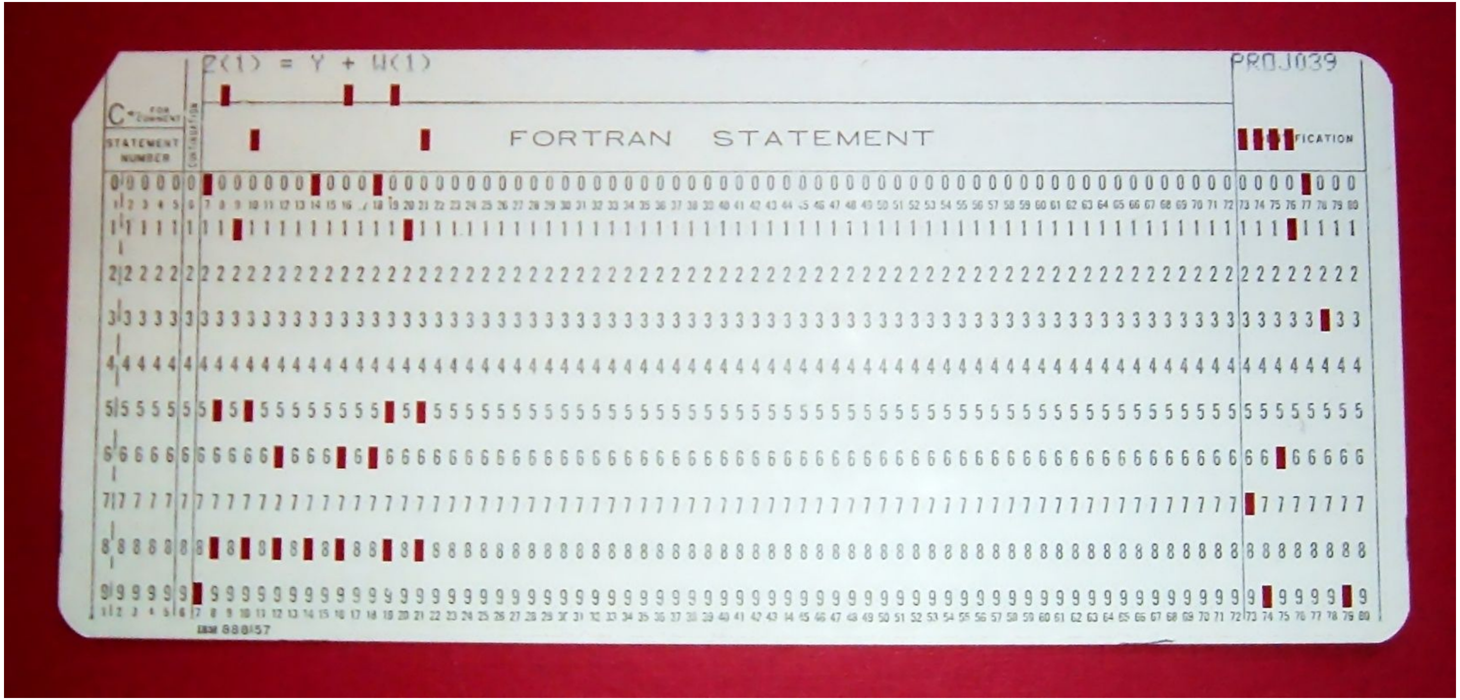
Data Science Survival Skills

What is actually data?

A file

A file

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



Storing information as bits and bytes

Number:

7 Binary: 111

Characters:

DATA Binary ➔

Pixel values:

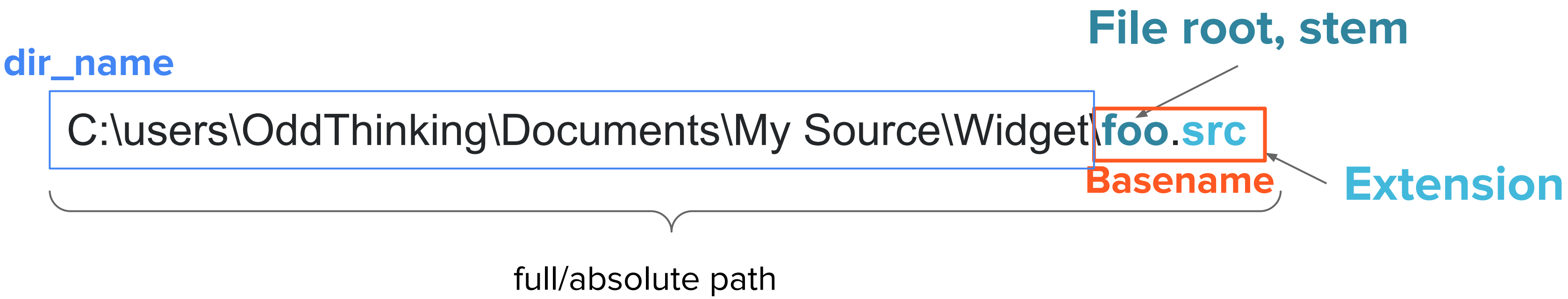
Are numbers!!

USASCII code chart

b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b_4	b_3	b_2	b_1	Column → Row ↓	0	1	2	3	4	5	6	7
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



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**?

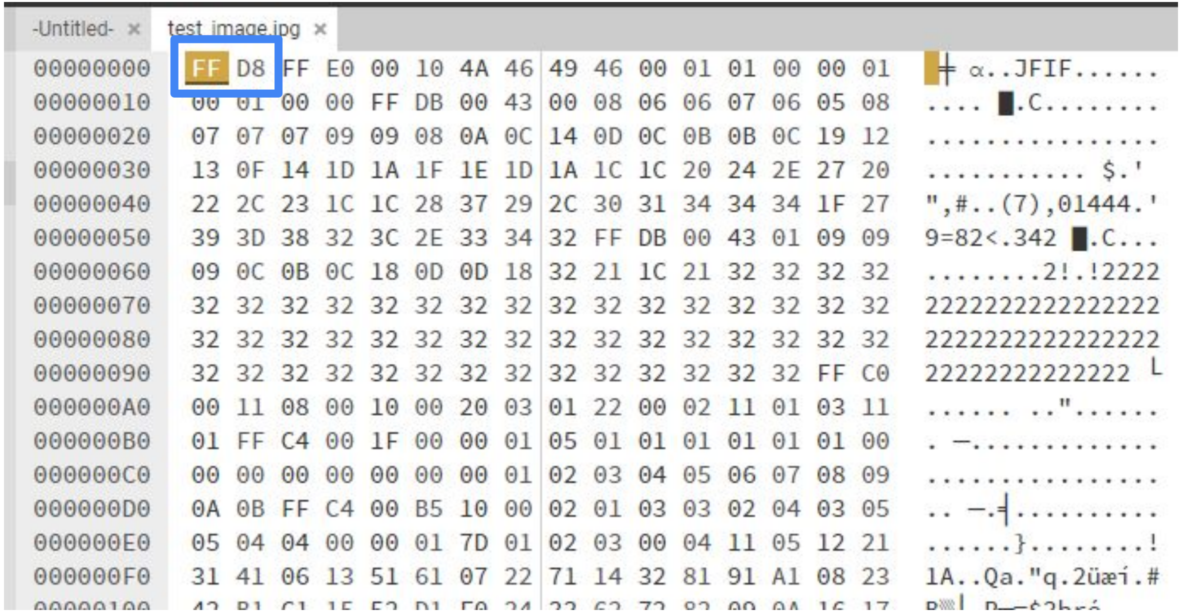
  2^32 - 1 ➜ 4,294,967,295 ($2^{32} - 1$) bytes, ca 4 GB max

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** does not convert the file to the JPG standard.
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

File extensions

Which ones do you know?

File types commonly used in Data Science

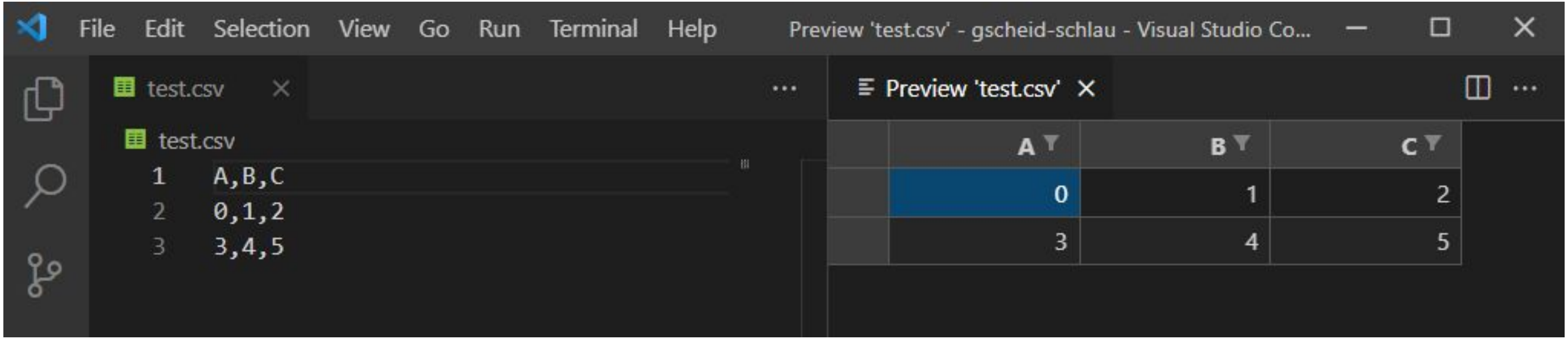
- Plain text (common extensions *.txt, *.csv, *.log, *.json, *.xml) - Python program code!
- Spreadsheets (*.xlsx)
- Word processing files (*.docx)
- Images (*.jpg -> Camera, *.png -> Scientific data, *.tif -> Microscopy)
- Videos (*.avi -> mostly raw data, *.mp4 almost everything, commonly h264 codec)
- 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

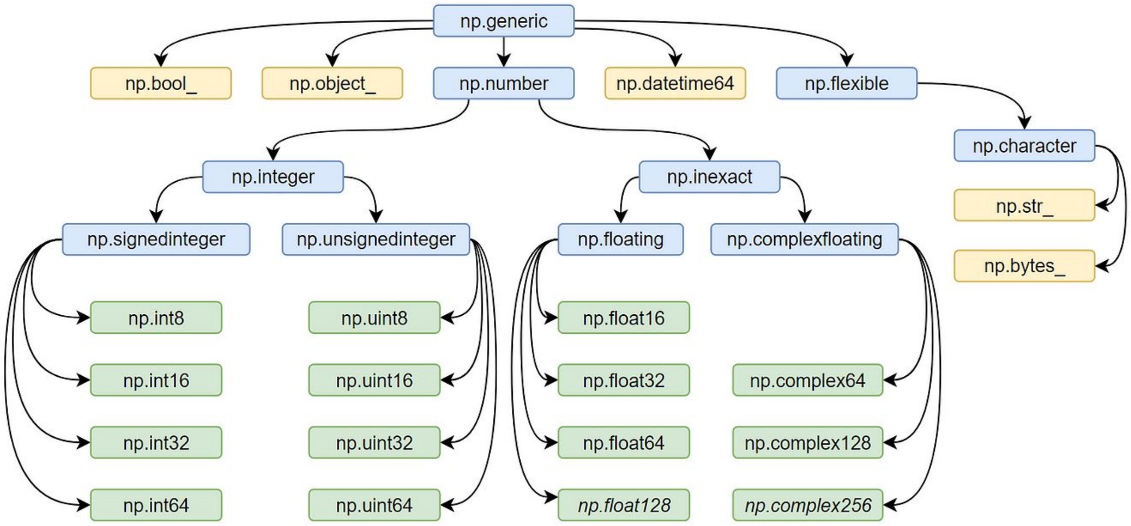
Tabular data

Table I

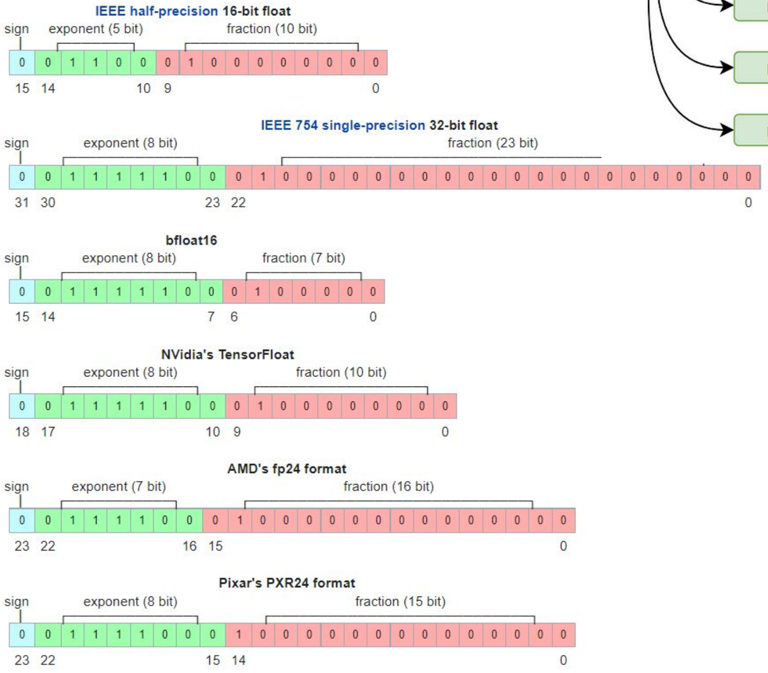
Iris setosa				*Iris versicolor*				*Iris virginica*			
Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width
5·1	3·5	1·4	0·2	7·0	3·2	4·7	1·4	6·3	3·3	6·0	2·5
4·9	3·0	1·4	0·2	6·4	3·2	4·5	1·5	5·8	2·7	5·1	1·9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

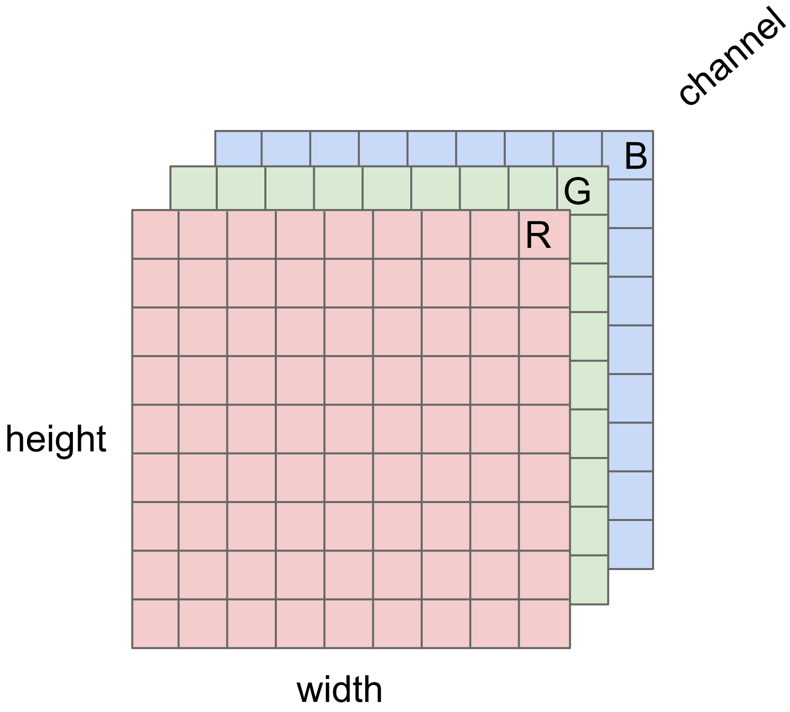
Data types



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



An image consists of many pixels



<u>Very common</u>:

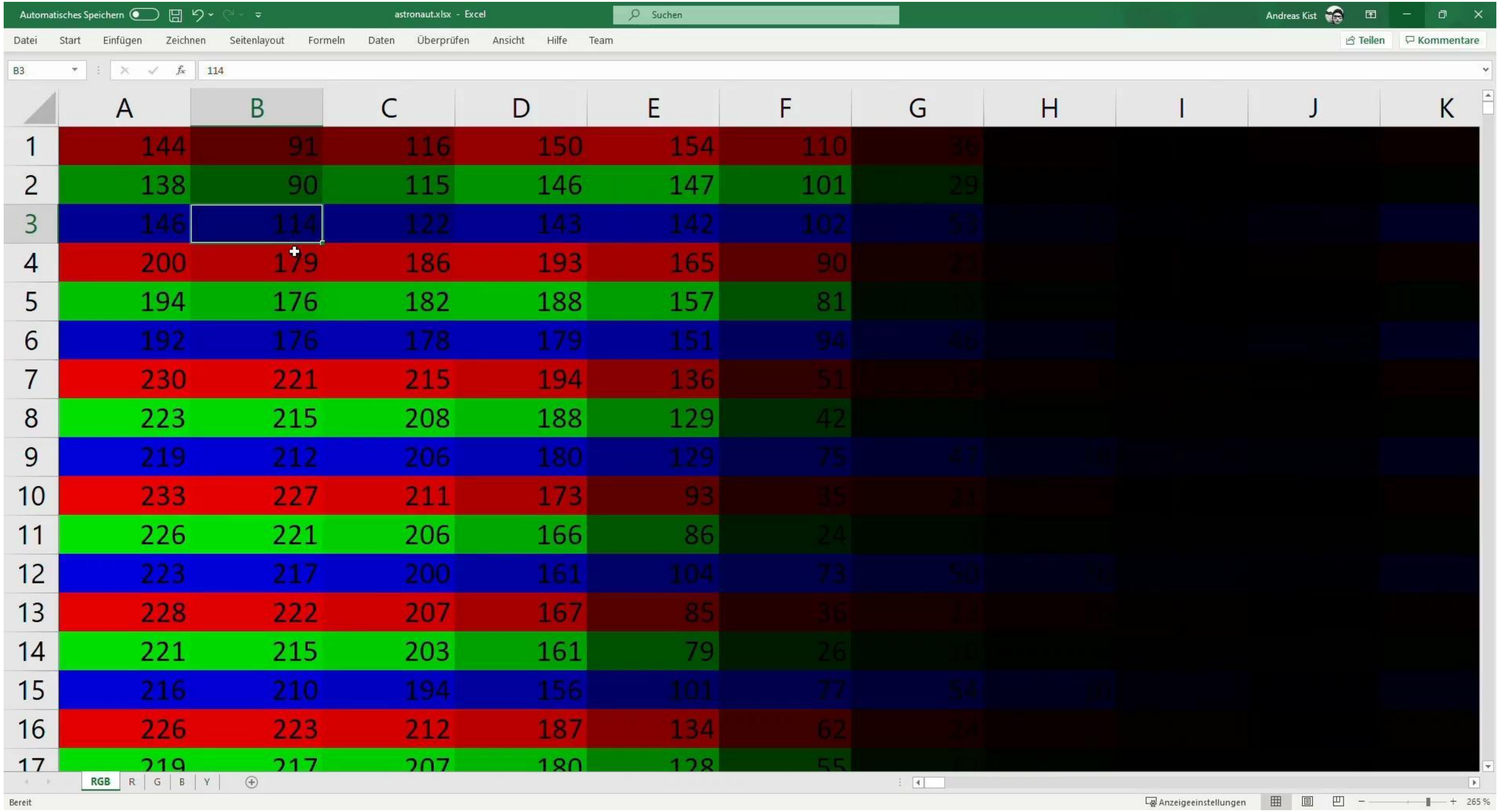
RGB (height x width x channels ⇒ HxWx3)
RGBA (HxWx4, last channel is alpha ⇔ transparency)
Monochrome (HxWx1 ⇒ HxW)

<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



Interacting with images in Python

OPENING/SAVING

imageio - Python library for reading and writing image data

scikit-image
image processing in python

OpenCV

PROCESSING

NumPy

Multi-dimensional image processing (scipy.ndimage)¶

scikit-image
image processing in python

OpenCV

PLOTTING

matplotlib

seaborn: statistical data visualization

PyQtGraph
Scientific Graphics and GUI Library for Python

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%

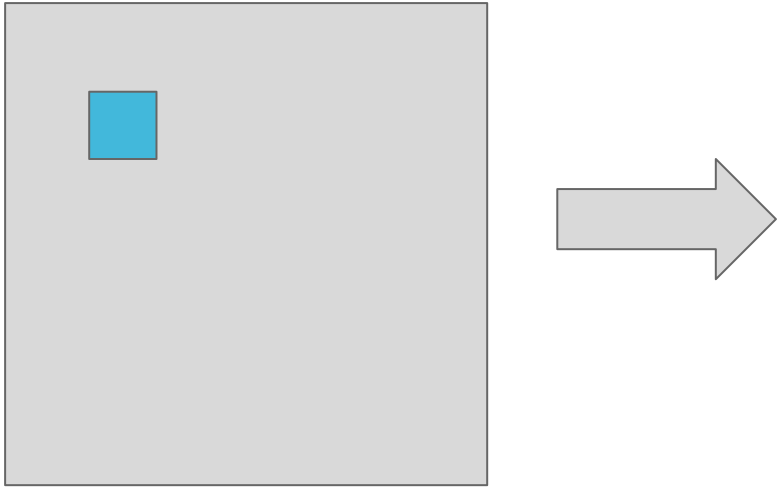
Do the math,... H(x) = 1.0881 bit

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

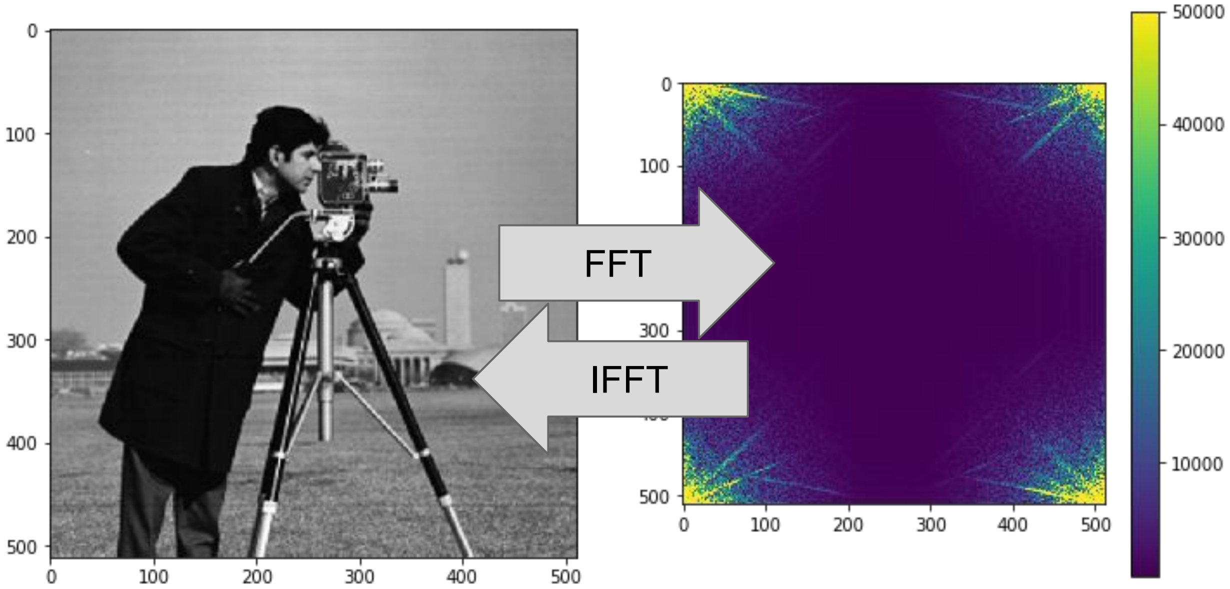
LOSSY

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

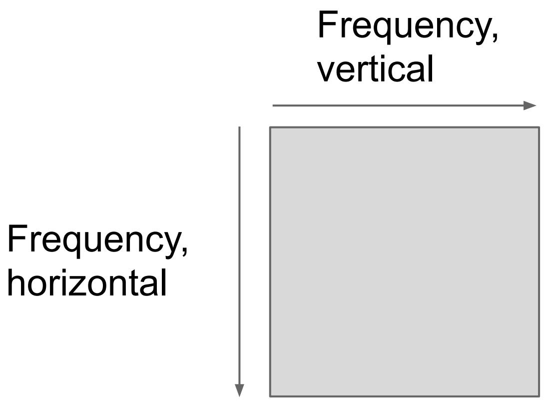
LOSSLESS

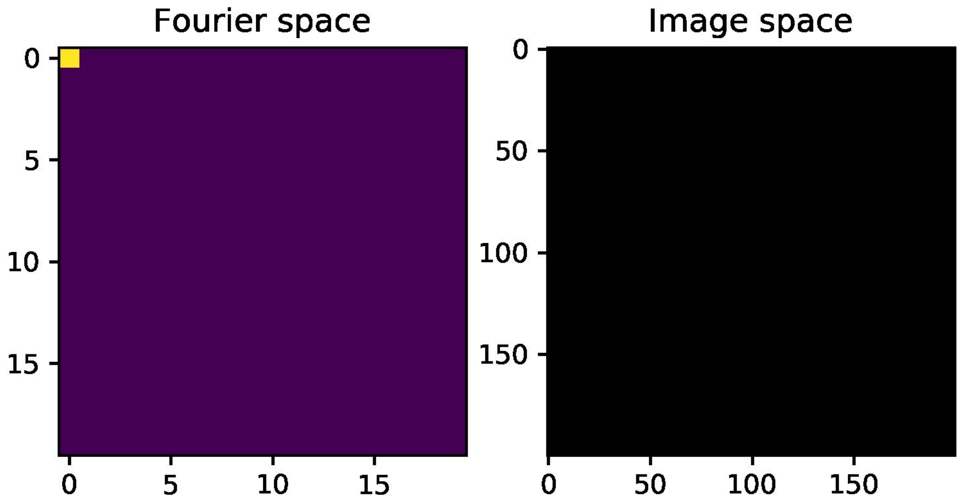
E.g. ZIP/7z files, PNG files

Images in Fourier space

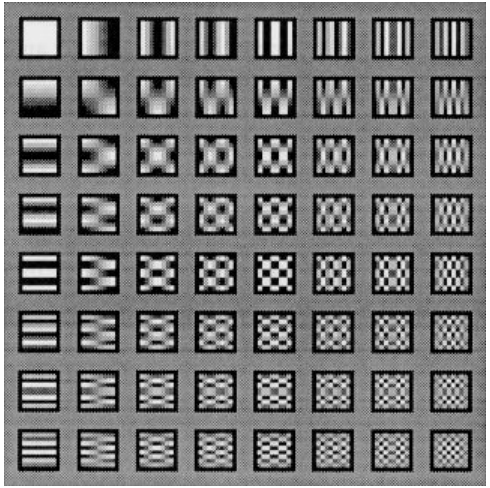


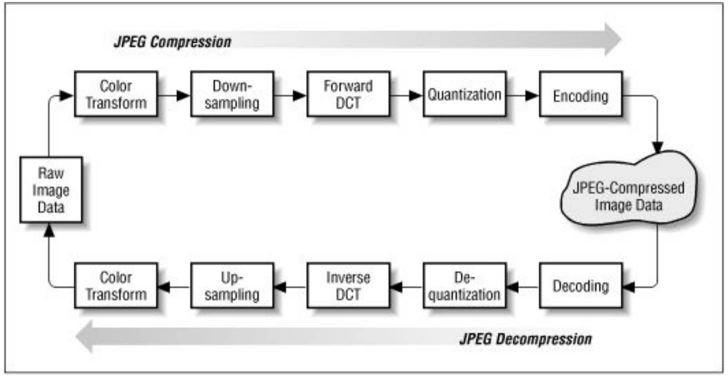
What is exactly happening?

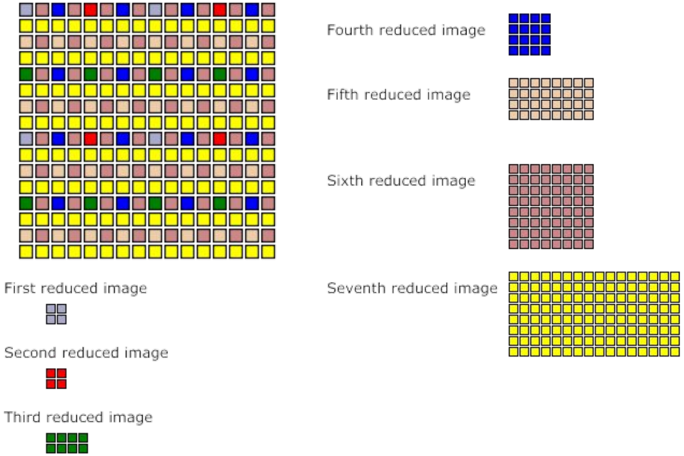




Lossy and lossless compression



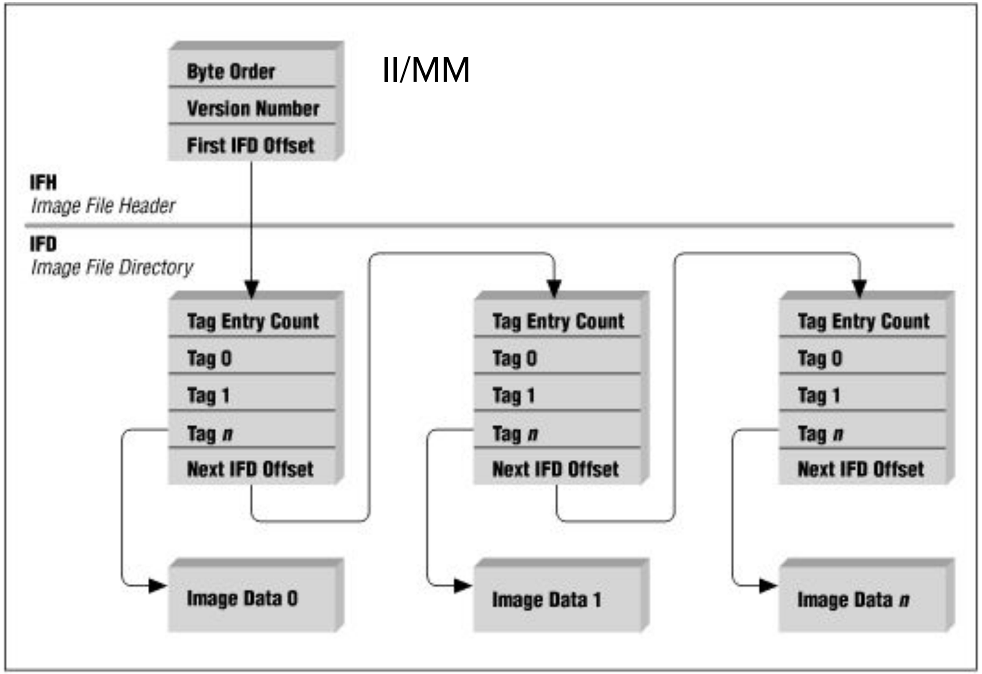




PNG: LZ77-based lossless compression (LUT, +Huffmann encoding)

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

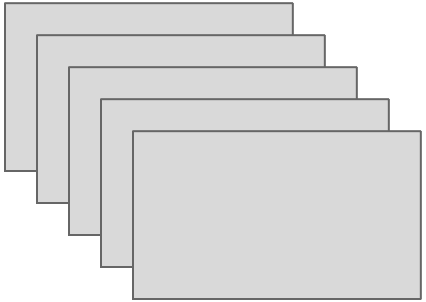
PNG

- Lossless compression
- Up to 4 channels (RGBA)

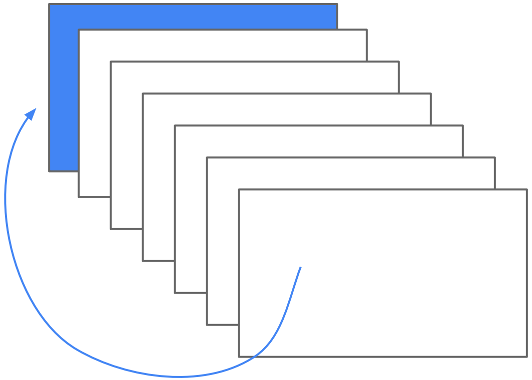
JPG

- Lossy compression
- Fine for photography
- Compression artifacts

Videos

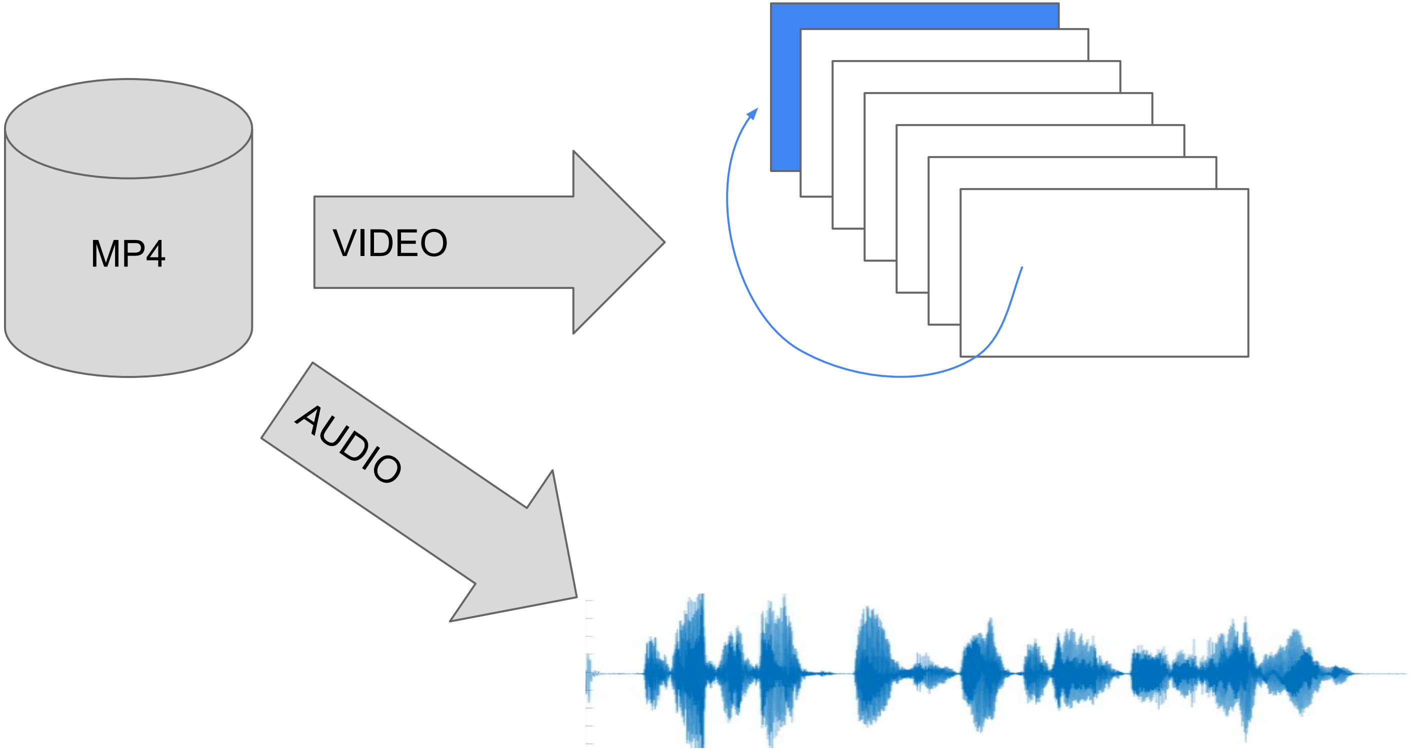


WAY 1: Store each frame one after another, each frame is independent



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

H264 codec in MP4 container



H264

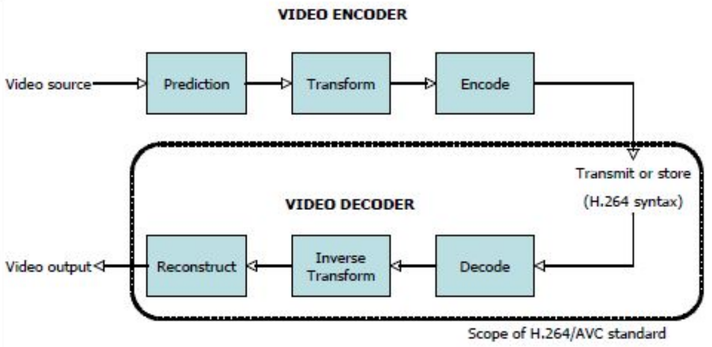


Figure 1 The H.264 video coding and decoding process

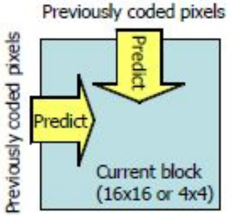


Figure 2 Intra prediction

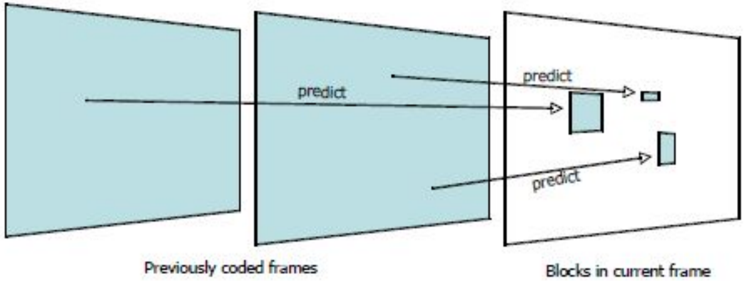


Figure 3 Inter prediction

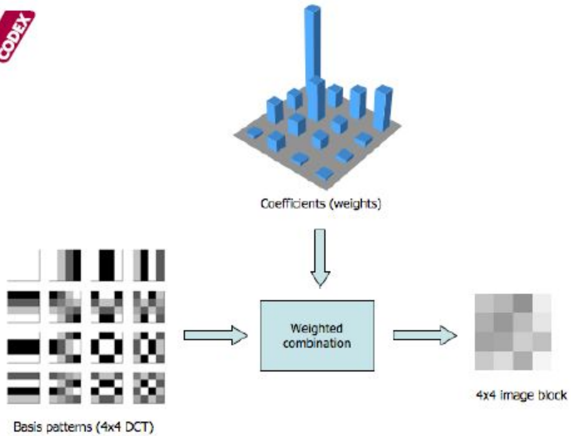


Figure 4 Inverse transform: combining weighted basis patterns to create a 4x4 image block

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

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.

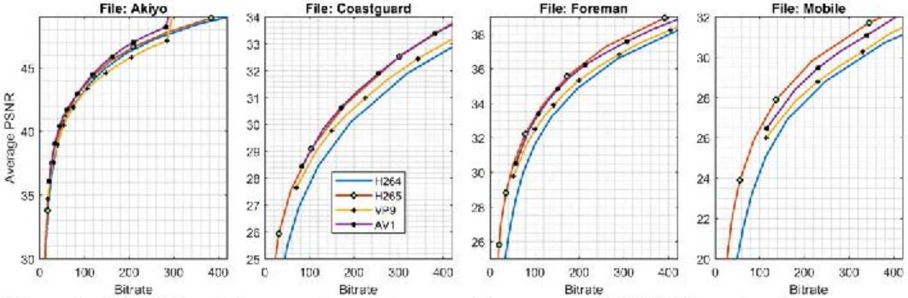


Fig. 8: PSNR with varying bitrates in case of CRF level adjustment (*placebo* presets for H.264 and H.265)

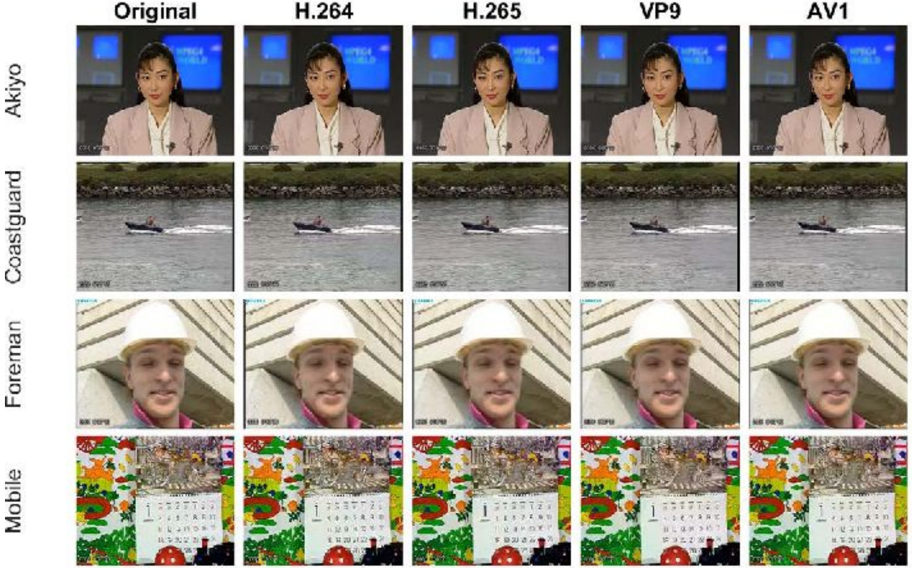
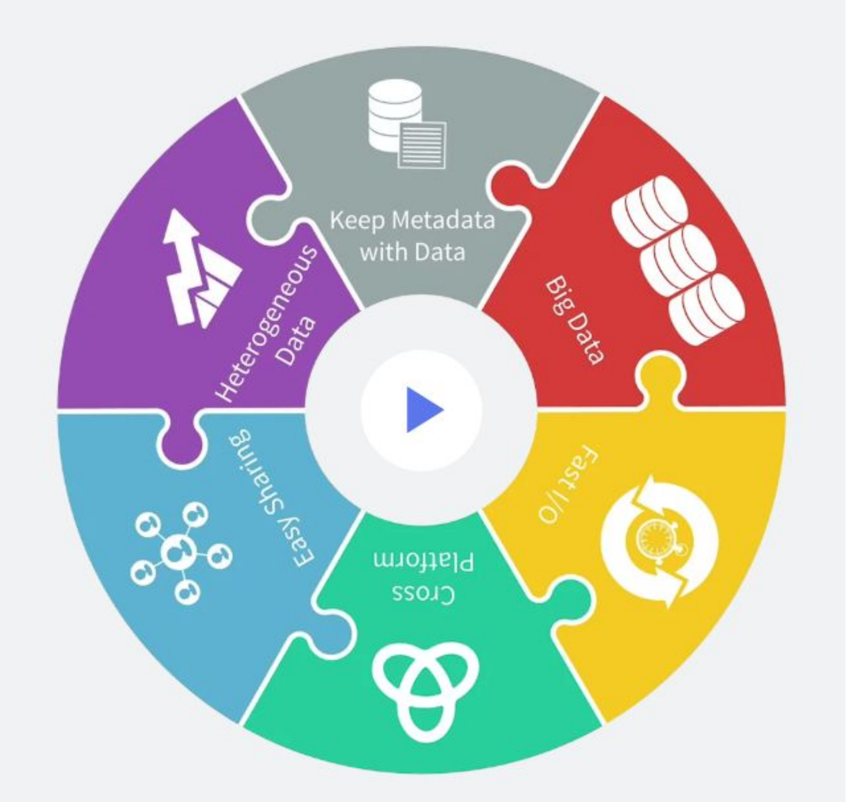
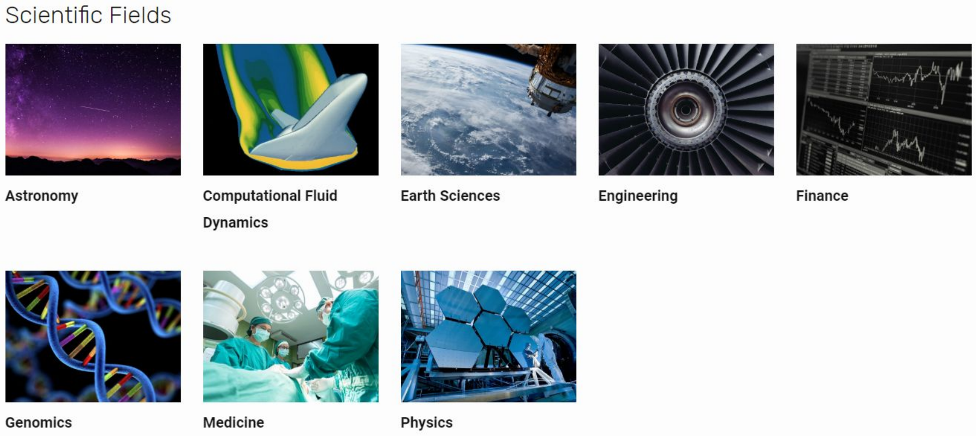


Fig. 9: First frames of the originals and the encoded videos at the

HDF5 - the universal file container





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.

How to handle/open/save HDF5?

```
pip install flammkuchen
```

```python
import flammkuchen as fl

d = {
    'foo': np.ones((10, 20)),
    'sub': {
        'bar': 'a string',
        'baz': 1.23,
    },
}
fl.save('test.h5', d)
```

Numpy ndarray
(e.g. multi-channel z-stack…)

E.g. some metadata...

Command line tool

Or, better yet, our custom tool `ddls` (or `python -m fl.ls`):

```
$ ddls test.h5
/foo                        array (10, 20) [float64]
/sub                        dict
/sub/bar                    'a string' (8) [unicode]
/sub/baz                    1.23 [float64]
```

Compression

Intelligent lossless compression,
A general feature of many libraries!

Check your data dtype!
You may save a lot of space!

Method	Compression	Space (MB)	Write time (s)	Read time (s)
scipy's mmwrite	N	145	79	40
numpy's save	N	134	1.36	0.75
pickle	N	115	0.63	0.17
deepdish (no compression)	N	115	0.52	0.17
numpy's savez_compressed	Y	32	8.88	1.33
pickle (gzip)	Y	29	5.19	0.86
deepdish (blosc)	Y	24	0.36	0.37
deepdish (zlib)	Y	21	9.01	0.83

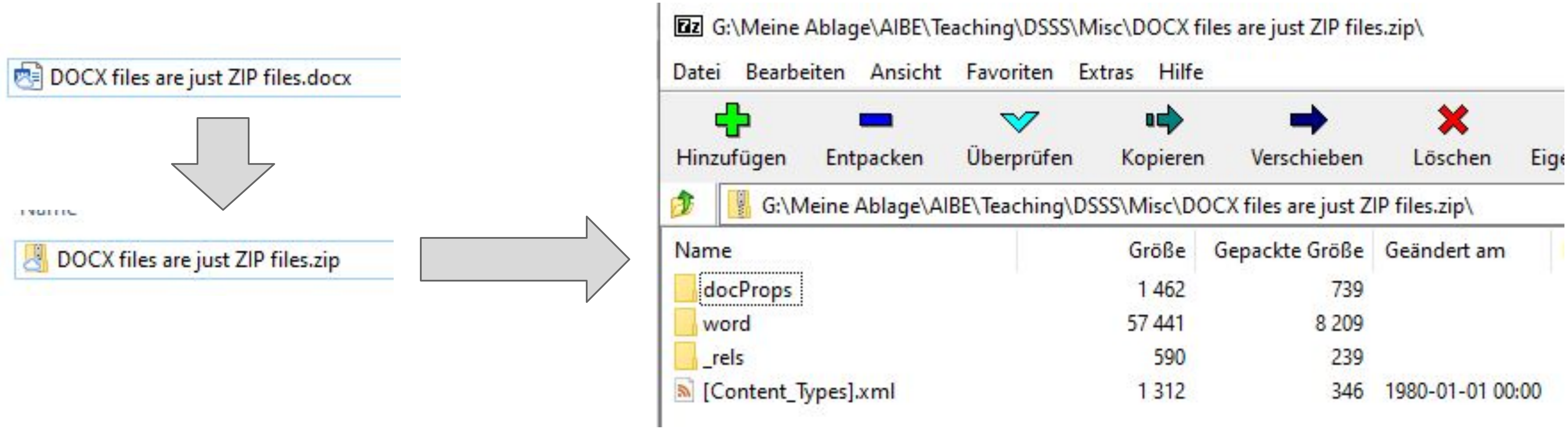
In [19]:
```
import flammkuchen as fl
import numpy as np
import os
```

In [20]:
```
x = np.random.randint(0, 2, (120, 512, 512, 3))  # int32!!
```

In [33]:
```
for i in range(10):
    %time fl.save("test_compression{}.h5".format(i), dict(x=x), compression=("blosc", i))
    print("compression level {}, file size: {:.2f} MB".format(i,
        os.path.getsize("test_compression{}.h5".format(i))/1048576))
```

```
Wall time: 283 ms
compression level 0, file size: 384.01 MB
Wall time: 496 ms
compression level 1, file size: 155.01 MB
Wall time: 704 ms
compression level 2, file size: 69.06 MB
Wall time: 855 ms
compression level 3, file size: 90.59 MB
Wall time: 825 ms
compression level 4, file size: 46.72 MB
Wall time: 805 ms
compression level 5, file size: 46.72 MB
Wall time: 789 ms
compression level 6, file size: 46.72 MB
Wall time: 782 ms
compression level 7, file size: 46.72 MB
Wall time: 763 ms
compression level 8, file size: 46.72 MB
Wall time: 772 ms
compression level 9, file size: 46.72 MB
```

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.