

# Was war zuerst: 🐱 oder 🍋? Emojis und ihre technischen Hintergründe

Workshop Tag der Philologie 25.06.2025 - [Oliver Deck](#)

In diesem Workshop wollen wir die technischen Grundlagen von Emojis kennenlernen und folgende Fragen beantworten:

1. Was sind Emojis?
2. Wie kommen sie in den Computer (und wieder raus)?
3. Wer entscheidet
  - A. welche Emojis es gibt?
  - B. wie Emojis aussehen?
  - C. was Emojis bedeuten?
4. Quatsch mit Emojis

Dazu werden wir auch einige kleine Experimente durchführen. In diesem interaktiven Jupyter-Notebook gibt es sowohl Text-Zellen (Markdown), als auch Code-Zellen (Python). Letztere kann man durch Klick auf den Play-Button oder mit Strg+Enter ausführen.

---

## Allgemeines

### Infrastruktur

Das Jupyter-notebook:

- Link zum Notebook: <https://github.com/ODeck/teaching/blob/main/EmojiWorkshop.ipynb>
- Link zum Notebook für den Direktimport in Jupyterhub/-lab oder Google Collab: <https://odeck.github.io/teaching/EmojiWorkshop.ipynb>

Wo kann man das Notebook ausführen:

- <https://www.jupyterhub.nrw/> (Für Studierende/Angestellte der RUB, Uni Münster, TUDo, H-BRS)
- <https://colab.research.google.com/> (Für Menschen mit Google-Account)
- <https://jupyter.org/try-jupyter/lab/> (Damit zumindest Teile funktionieren)
- Eigener PC (Für Leute, die sich mit Python auskennen)
- <https://github.com/ODeck/teaching/blob/main/ew.pdf> (PDF falls alle Stricke reißen)

Es dürfen gerne allen während des Workshops Fragen stellen und nebenbei in einem LLM ihrer Wahl selbst Emoji-Experimente durchführen:

- <https://gpt.ruhr-uni-bochum.de/app/chat>
- <https://www.chatgpt.com>
- <https://www.perplexity.ai/>
- <https://gemini.google.com/app>
- <https://copilot.microsoft.com>
- uvm

# Ultrakurzeinführung Python

- Text in Python (heißt `String`) steht entweder in einfachen ' oder doppelten " Anführungszeichen.
- Emojis sind Text, daher stehen sie auch in Anführungszeichen
- Funktionen und Methoden brauchen runde Klammern () in denen das steht, was in die Funktion oder Methode reinkommt
  - `print('Hallo')` -> gibt das Wort Hallo aus
  - `len('Hallo')` -> gibt die Länge des Wortes Hallo zurück
  - `sorted('Hallo')` -> gibt eine sortierte Liste der Buchstaben in Hallo zurück
  - weitere Funktionen werden im Laufe des Workshops erklärt oder sind nicht so wichtig
- Funktionen können verschachtelt werden und werden dann von innen nach außen abgearbeitet
  - `print(len(sorted('Hallo')))` -> sortiert erst, berechnet dann die Länge der Liste und zeigt diese dann an

Für Jupyter-Notebooks (wie dieses hier):

- Input steht in Python-Zellen
- Output steht darunter (nach Ausführen der Zelle)
- Mit Run All ganz oben im Notebook werden alle Zellen nacheinander ausgeführt
- alles was in `print()` s steht, wird ausgegeben
- das Ergebnis des letzten Befehls/Funktion/Berechnung wird auch immer ausgegeben

```
In [1]: print('Hello World')
```

```
Hello World
```

```
In [2]: sorted('Oliver')
```

```
Out[2]: ['O', 'e', 'i', 'l', 'r', 'v']
```

```
In [3]: sorted('🥚🐣')
```

```
Out[3]: ['🐣', '🥚']
```

Wenn wir mit der Programmiersprache Python die Liste der Emojis sortieren, sehen wir analog zur Sortierung der Buchstaben in Oliver, dass die Henne vor dem Ei da war!

Am Ende des Workshops wissen wir, warum das so ist, aber zunächst eine kleine Umfrage: <https://www.menti.com/alccvihjhiv7>



---

## Was sind Emojis?

### Emoji vs. Emoticon

Sowohl Emoticons als auch Emojis werden üblicherweise als digitale Phänomene gesehen.

- **Emoticons** (von engl. Emotion + Icon)
  - aus Buchstaben, Ziffern und Satzzeichen zusammengesetzte Piktogramme
  - wie :-), :D oder ;). ;)
- **Emojis** (japanisch 絵文字 "Bildschriftzeichen")
  - Bilder in Schriftzeichenform - d.h. man kann sie eintippen und muss nicht ein Bild in den Text einfügen wie z.B. bei GIFs
  - wie 😊, 😃, 😍, 🎉, 🎀, 🥚

[Frühe "Emoticons" von 1896](#)

## Telegraphische Zeichenkunst.

Wenn der „Hughſer“\*) früh erwacht,  
Er ein gähnend Antlitz macht:

Amtsgeſicht beim Dienstanfang,  
Wenn die „Karre“\*\*) ist im Gang:

Geht nach Wunsch die „Karre“ nicht,  
„sakrejou“†) dann dies Gesicht:

Iſt die Arbeit eine Haſz,  
Sieht man oft auch diese Fratz:

Doch läßt Dienſchluß ihn hinaus,  
Trägt er ſolch’ Geſicht nach Hauſ:

Fröhlich er bei „Muttern“ ſieht  
Und zum Kuß die Lippen ſpißt:

Bei der Mahlzeit lauernd dann  
Lieſt er das Verbandsorgan:

Denn Verbändler iſt gewiß  
Er; foſt wär’ ſein Antlitz dies:

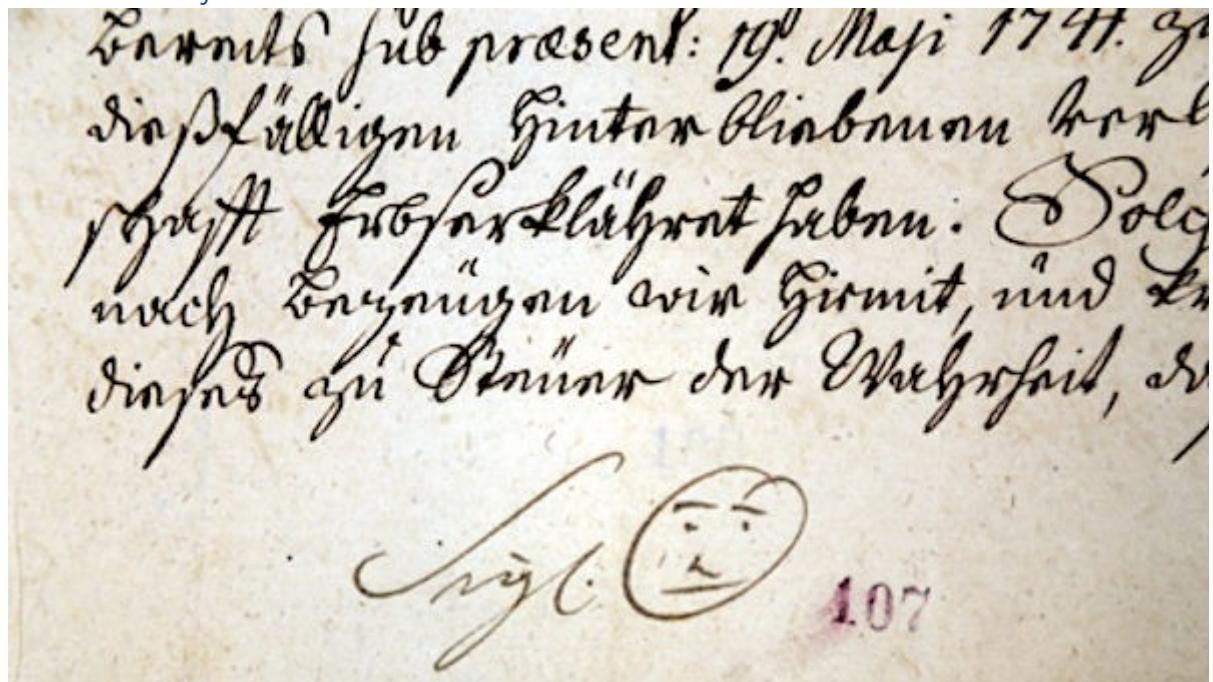
( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : )	Hält recht treu er zum Verband, Braucht wo’s ſein muß, er den „Mand“††)
( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : )	Stellenzulag’ macht zumeist Doppeltünig ihn und feift:
( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : )	Umgekehrt iſt’s, wenn ihn ſchlug Schicksal mit Gehaltsabzug:
( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : )	Ohne Gelb zu ultimo, Hält ihn doch ſein Pfeiſchen froh:
( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : )	Iſt der „erſte“ angerückt, Wichſt er ſeinen Bart entzückt:
( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : )	Bringt der Tod ihm endlich Ruh, Schließt er ſeine Augen zu:
( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : ) ( - : )	Und es klagt um ihn: „Wie ſchad!“ Mancher treue Kamerad.

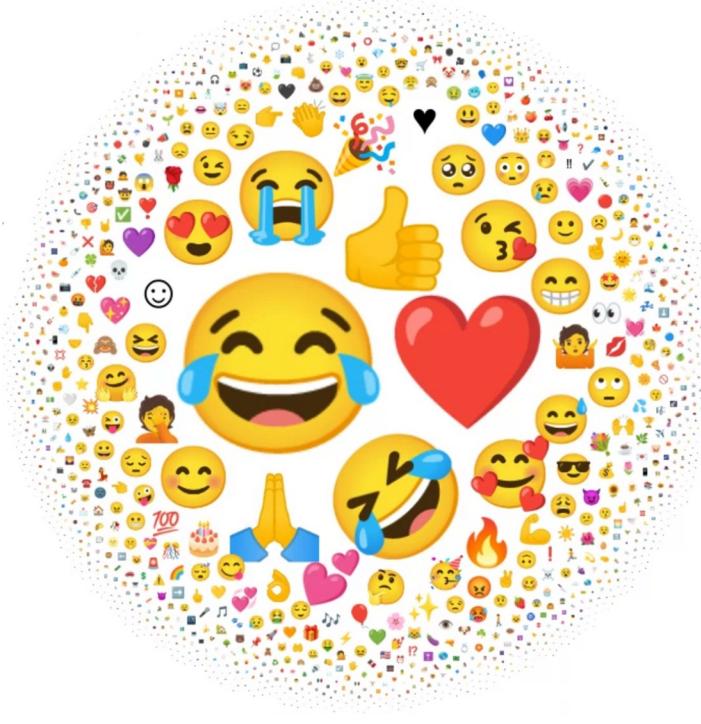
Streifen mit aufammengeſetzten Zeichen des Hughſe-Apparates.

Streifen mit aufammengeſetzten Zeichen des Hughſe-Apparates.

\*) In inneren Verkehr eingebürgerte Bezeichnung für den die Bedienung des Hughſe-Apparates wahrnehmenden Beamten.  
\*\*) Ebenſolche Bezeichnung für den Apparat.  
†) Ebenſolche, den Ausdruck der Ungebühr darſtellende Kombination von Buchſtaben.  
††) Berliner Ausdruck für „Mund“.

Ein frühes "Emoji" von 1741





Für diesen Workshop interessieren wir uns nur für Emojis, die man per Tastatur (unter Windows mit + . , Apple Control + Command + Space oder fn + e ) eingeben kann. Bilddateien, Gifs, etc funktionieren anders (und genau so wie alle nicht-Emoji-Bilddateien).

Mit Emoticons befassen wir uns nur am Rande, aber da sie aus Zeichen bestehen, funktionieren sie ähnlich wie Emojis.

---

## Wie kommen Emojis in den Computer (und wieder raus)?

Um diese Frage zu beantworten müssen wir ganz von vorne anfangen, nämlich bei der 0 und der 1 .

### Binärkode

Aktuelle<sup>[1]</sup> Computer (und alle sonstigen digitalen Geräte) basieren auf **Binärkode**, bzw. *binärer* oder *klassischer Logik*. Das heißt, alles was der Computer tut (und was in ihm drin ist) basiert auf einer Reihe von 1 en und 0 en.

Mit Binärkode kann man Zahlen darstellen. Die Dezimalzahl 3 entspricht der Binärzahl 11 , die 26 der 11010 . Mit Binärzahlen kann der Computer also alle<sup>[2]</sup> Zahlen verarbeiten.

1. ^ In Zukunft gibt es vermutlich auch nutzbare [Quantencomputer](#) und im kalten Krieg gab es auch Experimente mit [ternären Computern](#).

2. ^ Kommazahlen verhalten sich teilweise unerwartet. z.B. in Python-Code `print(0.1+0.2)`

```
In [4]: print(26)      # Zeige 26 in Dezimalzahlen an
          print(bin(26)) # Zeige 26 in Binärzahlen an (das '0b' davor ist nur ein Marker für binär)

          print(oct(26)) # Oktalzahlen
          print(hex(26)) # Hexadezimalzahlen
```

```
26
0b11010
0o32
0x1a
```

Jetzt wissen wir, wie Computer Zahlen speichern. Aber wir wollen Buchstaben (und Emojis)! Wie wird aus einer Zahl ein Buchstabe?

## ASCII

'American Standard Code for Information Interchange'

1. Wir ordnen bestimmten Zahlen bestimmte Buchstaben zu.
2. Außerdem gestalten wir die Rechnerarchitektur so, dass der Computer weiß: *Jetzt kommt eine Zahl* oder *Jetzt kommt ein Buchstabe*.

Wie genau das funktioniert geht zu weit (und unterscheidet sich zwischen Windows, Apple, Android, etc.) aber es genügt zu wissen, dass der Computer weiß wann eine Zahl eine Zahl ist, und wann ein Buchstabe (oder ein Bildpunkt oder eine Audiodatei etc).

Wie viele Zahlen brauchen wir für alle Buchstaben?

Speicherplatz war früher ein knappes Gut. Wie viele Bits (also Einsen und Nullen) braucht man für einen Buchstaben?

Antwort (der Amerikaner): Sieben Bits reichen! 0000000 bis 1111111 also 128 unterschiedliche Buchstaben

ASCII-Tabelle

## ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	1100000	60	0	96	60	1100000	140	'
1	1	1	1	[START OF HEADING]	49	31	1100001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	1100100	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[END OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	10000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	-
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

Mit der Tabelle kann man wunderbar englische Texte schreiben, und auch schon die klassischen Emoticons verfassen : - )

## Beyond ASCII

Irgendwann kamen dann die Deutschen und sagten "Äääääh, da fehlt doch was!" (Kein Anspruch auf historische Korrektheit). Und auch alle anderen Sprachen wollten Texte digital schreiben. Also wurden dutzende unterschiedliche Systeme entwickelt, wie man alle äöüß ø æ å 施氏食獅史 irgendwie unterbringen kann.

Zum Beispiel indem man einfach statt einer Tabelle jetzt 17 Tabellen zu je 7 Bit hat und zwischen denen umschaltet. ([ISO 8859](#))

Dann hat man gemerkt, dass  $17 \times 128$  auch nicht alle Schriftzeichen der Welt abdeckt (und man will ja irgendwann auch noch Platz für Emojis haben).

## Unicode

Eine einzige große Tabelle, bestehend aus 17 "Ebenen" zu je 16 Bit ( $2^{16} = 65.536$  Code Points)

- insgesamt also **Platz für 1.114.112 Zeichen**
  - Zeichen aus **161 modernen und alten Schriftarten** sowie **Steuerzeichen** und **Emojis**
  - Enthält aktuell 154.998 Zeichen
  - Unicode 16.0 definiert insgesamt 3790 Emojis, die sich auf 1431 Zeichen in 24 Blöcken verteilen

Jetzt ist auch Platz für seltene Schriftarten wie:



Und mit mehr Schriftzeichen auch mehr Potenzial für coolere Emoticons

- $(\square^j)^j \leq \square$  ( $\square \circ \square$ )  $\rightarrow (\square) /$

## Unicode Transformation Format (UTF)

Die Unicode-Tabelle weist nur Zeichen zu Zahlen zu, aber sagt nichts darüber aus, wie diese im Computer - in Einsen und Nullen - gespeichert werden.

Dafür gibt es unterschiedliche Formate:

- UTF-32:
    - Jedes Zeichen wird in 32 Bits (also 4 Bytes) gespeichert
    - Einfachstes Format, es gibt genügend Bits für jede Zahl
    - Benötigt am meisten Speicherplatz
    - Selten verwendet
    - Nicht ASCII-kompatibel
  - UTF-16:
    - Jedes Zeichen wird in **mindestens** 16 Bits (also 2 Bytes) gespeichert
    - Windows (und Java) Format
    - Nicht ASCII-kompatibel
  - **UTF-8** 
    - Jedes Zeichen wird in **mindestens** 8 Bit (also 1 Byte) gespeichert
    - Standard im Internet und sonst überall

- ASCII-kompatibel
- **Funktionsweise** etwas kompliziert, aber vereinfacht gesagt werden die ersten Bits jedes Bytes genutzt, um zu sagen wie viele Bytes zum aktuellen Zeichen gehören

Beispiele UTF-8

### Beispiele für UTF-8 Kodierungen

<b>Zeichen</b>	<b>Unicode</b>	<b>Unicode binär</b>	<b>UTF-8 binär</b>
Buchstabe y	U+0079	00000000 01111001	01111001
Buchstabe ä	U+00E4	00000000 11100100	11000011 10100100
Griechischer Großbuchstabe Omega Ω	U+03A9	00000011 10101001	11001110 10101001
Eurozeichen €	U+20AC	00100000 10101100	11100010 10000010 10101100
Violinschlüssel ♪	U+1D11E	00000001 11010001 00011110	11110000 10011101 10000100 10011110

### Zusammengesetzte Zeichen

Sowohl das A als auch das Ä haben im Unicode einen einzelnen Code Point.

Das A an Stelle 65, das Ä an Stelle 196.

```
In [5]: print('A')          # Zeige das A an
        print(ord('A'))    # Zeige die ASCII/UTF-8-Zahl von A an

        print()            # Gib eine Leerzeile aus (nur für bessere Lesbarkeit)

        print('Ä')          # s.o. nur für Ä
        print(ord('Ä'))
```

A  
65

Ä  
196

Aber man kann das Ä auch aus dem A und dem Umlaut-Code-Point \u0308 kombinieren. Die Schreibweise \u0308 bezeichnet hier einen genauen Code-Point, was vor allem bei nicht anzeigbaren Zeichen gemacht wird. Das Zahlenformat ist in dieser Schreibweise weder dezimal, noch binär, sondern hexadezimal.

```
In [6]: print('A')          # Zeige ein A an
```

```

print('\u0308')      # Zeige das Umlautzeichen am Code-Point \u0308 an
print('A\u0308')    # Zeige die Kombination beider Zeichen an
print()
print(ord('A'))
print(ord('\u0308'))
print()
print(hex(ord('\u0308'))) # Zeige das Umlautzeichen in hexadezimal an, was dann 308 ist (mit einem Präfix)

```

A

..

Ä

65

776

0x308

Das zusammengesetzte Ä ist "länger" als das normale "Ä", weil es aus 2 Code-Points besteht:

```
In [7]: print(len('Ä'))          # Das Ä mit der Tastatur getippt
        print(len('Ä'))          # Das vom Output oben rauskopierte Ä
        print(len('A\u0308'))     # Die Kombination wie oben
```

1

2

2

## Unicode Codepoints in UTF-8

Wie können wir die UTF-8 Bytesequenz für ein bestimmtes Zeichen sehen?

```
In [8]: print("ä")
        print(ord("ä"))
        print(bin(ord('ä'))) # funktioniert das?
```

ä

228

0b11100100

## Beispiele für UTF-8 Kodierungen

Zeichen	Unicode	Unicode binär	UTF-8 binär
Buchstabe y	U+0079	00000000 01111001	01111001
Buchstabe ä	U+00E4	00000000 11100100	11000011 10100100

Um die tatsächlichen Bytes zu sehen können wir die Funktion `bytes()` nutzen und müssen dabei das gewünschte Encoding (also für uns UTF-8) angeben. Die Funktion gibt uns die Bytes allerdings in hexadezimal. Wenn wir die Bits in binär sehen wollen, müssen wir das Ergebnis nochmal umwandeln.

```
In [9]: b = bytes('ä', encoding='UTF-8')      # wir speichern das Ergebnis der bytes()-Funktion in der Variable b
        print(b)                            # b anzeigen (hexadezimal)

        print([bin(byte) for byte in b])     # jedes Byte in b in binär umgewandelt anzeigen
```

b'\xc3\xaa'

[ '0b11000011', '0b10100100' ]

Die Bytesequenz ['0b11000011', '0b10100100'] stimmt jetzt mit der Tabelle von oben überein.

# Emojis in Unicode / UTF-8

```
In [10]: print('😊') # Zeige Zeichen an
          print(ord('😊')) # Zeige Unicode-Code-Point an
          print(len('😊')) # Länge des Zeichens
          print([bin(byte) for byte in bytes('😊', encoding='UTF-8')]) # Byte-Sequenz in UTF-8
```

😊  
128513  
1  
['0b11110000', '0b10011111', '0b10011000', '0b10000001']

```
In [11]: print('🦄') # Zeige Zeichen an
          print(ord('🦄')) # Zeige Unicode-Code-Point an
          print(len('🦄')) # Länge des Zeichens
          print([bin(byte) for byte in bytes('🦄', encoding='UTF-8')]) # Byte-Sequenz in UTF-8
```

🦄  
128020  
1  
['0b11110000', '0b10011111', '0b10010000', '0b10010100']

```
In [12]: print('🥚') # Zeige Zeichen an
          print(ord('🥚')) # Zeige Unicode-Code-Point an
          print(len('🥚')) # Länge des Zeichens
          print([bin(byte) for byte in bytes('🥚', encoding='UTF-8')]) # Byte-Sequenz in UTF-8
```

🥚  
129370  
1  
['0b11110000', '0b10011111', '0b10100101', '0b10011010']

```
In [13]: ord('🦄') < ord('🥚')
```

Out[13]: True

## Zusammengesetzte Emojis

Zitat von oben: Unicode 16.0 definiert insgesamt 3.790 Emojis, die sich auf 1.431 Zeichen in 24 Blöcken verteilen

Wie bei Buchstaben mit Umlauten, Diakritika, Zeichen für tonale Sprachen, etc. gibt es auch Emojis, die aus unterschiedlichen Code-Points zusammengesetzt sind. Dies sind vor allem die Hautfarben- und Gendervarianten, aber auch Berufe und einige nichtsichtbare technische Sonderzeichen wie \u200d (zero-width joiner) können zu Emojis kombiniert werden.

```
In [14]: print('😳', len('😳')) # Emoji und die Länge des Emojis anzeigen
          print('RGBO', len('RGBO'))
          print('👉', len('👉'))
          print('👉🏻', len('👉🏻'))
          print('👉🏿', len('👉🏿'))
```

😳 1  
RGBO 2  
👉 1  
👉🏻 4  
👉🏿 5

```
In [15]: # Familien

          print('👨‍👩', len('👨‍👩'))
          print('👨‍👩‍👧', len('👨‍👩‍👧'))
          print('👨‍👩‍👧‍👦', len('👨‍👩‍👧‍👦'))
```

```
print('👨‍👩‍👧‍👦', len('👨‍👩‍👧‍👦'))
print('👨‍👩‍👧‍👦', len('👨‍👩‍👧‍👦'))
```

```
👨‍👩‍👧‍👦 1
👨‍👩‍👧‍👦 3
👨‍👩‍👧‍👦 7
👨‍👩‍👧‍👦 7
👨‍👩‍👧‍👦 11
```

```
In [16]: # wir importieren die Python-Bibliothek 'emoji'
```

```
try:
    import emoji
except ImportError:
    %pip install emoji
    import emoji
```

```
In [17]:
```

```
em = '🐓'
print(em, emoji.demojize(em))
# mit der Funktion demojize zeigen wir die Namen der Emojis

em = '🥚'
print(em, emoji.demojize(em))

🐓 :chicken:
🥚 :egg:
```

```
In [18]:
```

```
def parts(testemoji):
    print(f'{testemoji} hat {len(testemoji)} Bestandteil(e):')
    print('\t',[part for part in testemoji])
    print('\t',[emoji.demojize(part) for part in testemoji])
    print()
```

```
parts('🐓')
parts('🥚')
parts('👨‍👩‍👧‍👦')
parts('👨‍👩‍👧‍👦')
parts('👨‍👩‍👧‍👦')
parts('👨‍👩‍👧‍👦')
parts('👨‍👩‍👧‍👦')
parts('👨‍👩‍👧‍👦')
```

```

ǚ hat 1 Bestandteil(e):
['ǚ']

ǚ hat 2 Bestandteil(e):
['👤', '🟧']
['👤', ':medium-light_skin_tone:']

ǚ hat 4 Bestandteil(e):
['🤷', '\u200d', '♂', '']
[':person_shrugging:', '\u200d', ':male_sign:', '']

ǚ hat 5 Bestandteil(e):
['🤷', '🟧', '\u200d', '♀', '']
[':person_shrugging:', ':medium-light_skin_tone:', '\u200d', ':female_sign:', '']

ǚ hat 4 Bestandteil(e):
['👤', '🟧', '\u200d', '🏡']
['👤', ':medium-light_skin_tone:', '\u200d', ':school:']

ǚ hat 3 Bestandteil(e):
['👤', '\u200d', '🏭']
[':woman:', '\u200d', ':factory:']

ǚ hat 11 Bestandteil(e):
['👤', '🟧', '\u200d', '👤', '🟧', '\u200d', '👤', '🟧', '\u200d', '👤', '🟧']
['👤', ':medium-dark_skin_tone:', '\u200d', ':woman:', ':medium-light_skin_tone:', '\u200d', ':girl:', ':medium_skin_tone:', '\u200d', ':boy:', ':dark_skin_tone:']

```

In [19]: # Emojis subtrahieren

```

em = 'ǚ'
print(em)
for i in range(len(em)-1): # solange es Zeichen im Emoji gibt
    em = em[:-1] # ziehen wir jeweils das letzte Zeichen ab
    print(em) # und geben das resultierende Emoji aus

```



In [20]:

```

em = 'ǚ'
print(em, emoji.demojize(em))
for i in range(len(em)):
    em = em[:-1]
    print(em, emoji.demojize(em))

```

```
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl_medium_skin_tone::boy_dark_skin_tone:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl_medium_skin_tone::boy:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl_medium_skin_tone:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl_medium_skin_tone:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone:  
:man_medium-dark_skin_tone::woman:  
:man_medium-dark_skin_tone:  
:man_medium-dark_skin_tone:  
:man:  
  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl_medium_skin_tone::boy_dark_skin_tone:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl_medium_skin_tone::boy:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl_medium_skin_tone:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl_medium_skin_tone:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone::girl:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone:  
:man_medium-dark_skin_tone::woman_medium-light_skin_tone:  
:man_medium-dark_skin_tone::woman:  
:man_medium-dark_skin_tone:  
:man_medium-dark_skin_tone:  
:man:
```

---

## Wer entscheidet welche Emojis es gibt?

### Unicode-Konsortium und der Entscheidungsprozess für neue Emojis

Das Unicode-Konsortium ist eine gemeinnützige Organisation, die den [Unicode-Standard](#) entwickelt und pflegt. Dieser Standard legt fest, wie Schriftzeichen und Emojis weltweit digital dargestellt werden.



Wer ein neues Emoji vorschlagen möchte, [kann einen offiziellen Antrag beim Unicode-Konsortium einreichen](#).

Es gibt bestimmte **notwendige und ausschließende Merkmale** für neue Emojis, wie zum Beispiel Mehrdeutigkeit (𩶻 for das Tier oder z.B. einen Immobilien-𩶻) oder Einzigartigkeit (unterschiedlich genug von existierenden Emojis).

Ältere Emojis verletzen teilweise Vorgaben (wie zum Beispiel "Emojis dürfen keinen Text enthalten" bei **NEW OK**, etc.). Sobald ein Code-Point im Unicode allerdings vergeben ist, kann das nicht mehr rückgängig gemacht werden, deshalb bleiben die Zeichen erhalten - und das Konsortium lässt sich viel Zeit um über neue Emojis zu entscheiden

---

## Wer entscheidet wie Emojis aussehen?

Der Unicode-Standard beschreibt nur welche Zeichen welchen Code-Points zugeordnet sind. Ähnlich wie bei Schriftarten entscheidet der Software-Hersteller über das finale Aussehen auf seiner Plattform. Daher sehen Emojis auch je nach Gerät und Software unterschiedlich aus.

[Salutierendes Emoji](#)



Apple



Google Noto Color Emoji



Samsung



WhatsApp



Microsoft 3D Fluent



Twitter / X



Facebook



Huawei



JoyPixels

Auf Websites wie [Emojipedia](#) kann man die unterschiedlichen Designs sehen, zum Beispiel [für das beliebteste Emoji 😊](#).





Beispiel Facebook



## Emojis die nicht angezeigt werden

Da die Anzeige von der Softwareplattform abhängig ist, kann es vorkommen, dass bestimmte Emoji auf bestimmten Plattformen noch nicht umgesetzt sind (Das gleiche kann natürlich bei allen Unicode-Zeichen passieren, zum Beispiel bei den exotischeren Schriftzeichen).

- Dann sieht man normalerweise (die systemabhängige Repräsentation von) ☐ (WHITE SQUARE).
- Wenn ein Unicode-Fehler auftritt, dann sieht man häufig ♠ (REPLACEMENT CHARACTER).
- Für Länderflaggen werden ggf. Ländercodes angezeigt, also DE statt 🇩. (Falls das zweite keine Flagge ist, dann trifft das auf der aktuellen Plattform zu.)

## Wer entscheidet was Emojis bedeuten?

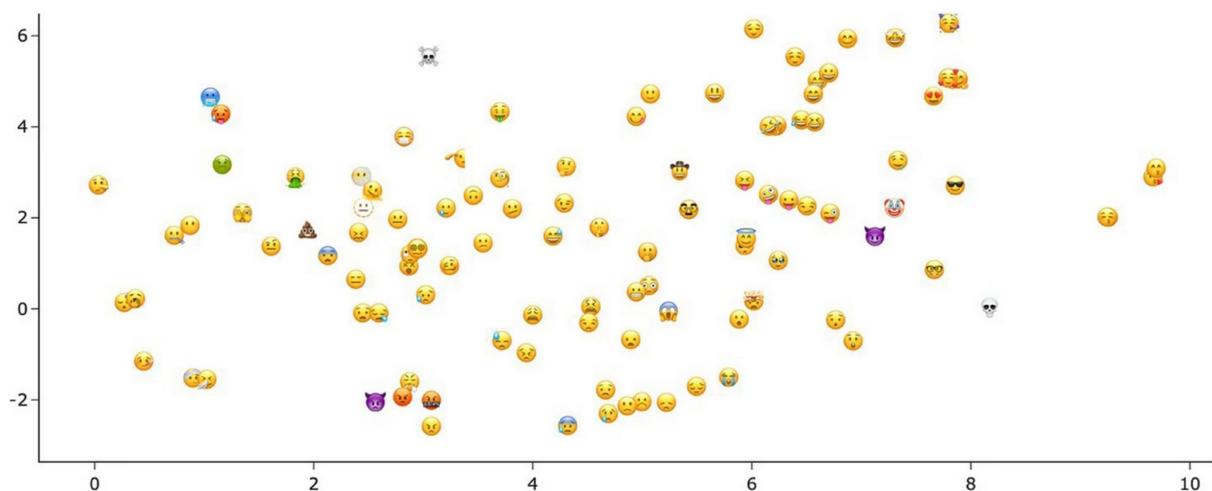
**Wir!**

Often, emojis also take on new meanings or connotations over time (examples are the 🍆 as a phallic symbol and the “slightly smiling” 😊 emoji, which has been reinterpreted as indicating “I’m dead inside”, particularly by younger users).

😂 vs. 💀

Scheffler & Nenchev 2024 - Affective, semantic, frequency, and descriptive norms for 107 face emojis

**From: Affective, semantic, frequency, and descriptive norms for 107 face emojis**



T-SNE plot of emoji description embeddings (average embeddings for our solicited descriptions for each emoji)

# Quatsch mit Emojis

**ACHTUNG:** Es folgt wahnsinniger Spaghetti-Code generiert von Copilot und angepasst von mir. Wer programmieren kann sollte die Augen schließen, wer nicht programmieren kann sollte keinesfalls denken, dass das normalerweise so aussieht!

## Random Emojis

Man könnte zum Beispiel zufällige Emojis in einen Text einfügen.

In [21]:

```
import random

def add_random_emoji(text, emoji_list=None):
    """Appends a random emoji from emoji_list (or all emojis) to the text."""
    if emoji_list is None:
        # Use all emoji characters from the emoji library
        emoji_list = list(emoji.EMOJI_DATA.keys())
    return f'{text} {random.choice(emoji_list)}'

def random_emoji_insertion(text, n=3, emoji_list=None): # uff
    """Randomly inserts n emojis into the text, preserving original line breaks."""
    if emoji_list is None:
        emoji_list = list(emoji.EMOJI_DATA.keys())
    lines = text.splitlines()
    # Flatten words and keep track of their (line_idx, word_idx) positions
    all_words = []
    word_indices = []
    for line_idx, line in enumerate(lines):
        words = line.split()
        for word_idx, word in enumerate(words):
            all_words.append(word)
            word_indices.append((line_idx, word_idx))
    # Insert n emojis at random positions
    for _ in range(n):
        idx = random.randint(0, len(all_words))
        emoji_char = random.choice(emoji_list)
        all_words.insert(idx, emoji_char)
        word_indices.insert(idx, (-1, -1)) # Mark as emoji, not a word from original text
    # Reconstruct lines
    new_lines = [[] for _ in lines]
    ptr = 0
    for line_idx, line in enumerate(lines):
        words = line.split()
        for _ in range(len(words)):
            if ptr < len(word_indices) and word_indices[ptr][0] == line_idx:
                new_lines[line_idx].append(all_words[ptr])
                ptr += 1
            else:
                break
        # Insert any emojis that were inserted at this line position
        while ptr < len(word_indices) and (word_indices[ptr][0] == -1 or word_indices[ptr][0] != line_idx):
            new_lines[line_idx].append(all_words[ptr])
            ptr += 1
    # If there are any remaining emojis (e.g. inserted at the end), add them to the last line
    while ptr < len(all_words):
        new_lines[-1].append(all_words[ptr])
        ptr += 1
    return '\n'.join(' '.join(line) for line in new_lines)
```

In [22]:

## # Beispiel:

```
print(add_random_emoji("Das ist ein Test"))
print(random_emoji_insertion("Heute ist ein schöner Tag zum Feiern!", 2))      # fügt 2 Emojis ein
print(random_emoji_insertion("Heute ist ein schöner Tag zum Feiern!", 20))     # fügt 20 Emojis ein
```

Das ist ein Test

Heute ist ein schöner Tag zum Feiern

Heute ist  ein schö

Feiern!

Die Emojis sollen nicht zufällig sein, sondern aus 'sinnvollen' Listen ausgewählt werden.

Copilot Prompt: Give me a list of the most boomer emojis and zoomer emojis

```
In [23]: # List of "boomer" emojis, often considered old-school or used by older generations
```

```
boomer_emojis = [
    "😂", "# Face with Tears of Joy",
    "👍", "# Thumbs Up",
    "😊", "# Smiling Face with Smiling Eyes",
    "😎", "# Winking Face",
    "😃", "# Smiling Face with Open Mouth",
    "👏", "# Clapping Hands",
    "🤝", "# Folded Hands",
    "🎉", "# Party Popper",
    "😎", "# Smiling Face with Sunglasses",
    "😅", "# Smiling Face with Sweat",
    "😆", "# Grinning Face with Smiling Eyes",
    "😋", "# Winking Face with Tongue",
    "😇", "# Smiling Face with Halo",
    "😎", "# Smirking Face",
    "💯", "# Hundred Points",
    "👌", "# OK Hand",
    "😍", "# Face Savoring Food",
    "🎂", "# Birthday Cake",
    "🎶", "# Musical Notes",
    "💃", "# Woman Dancing"
]
```

# List of "zoomer" emojis, often considered trendy or used by younger generations

```
zoomer_emojis = [
    "😭", "# Loudly Crying Face",
    "💀", "# Skull",
    "🧢", "# Billed Cap (used for \"cap/no cap\"),",
    "融化", "# Melting Face",
    "🤝", "# Heart Hands",
    "🥺", "# Pleading Face",
    "✨", "# Sparkles",
    "😩", "# Weary Face",
    "☁️", "# Face in Clouds",
    "🦋", "# Butterfly",
    "👤", "# Person Standing",
    "ଓ", "# Dotted Line Face",
    "ଓ", "# Smiling Face with Horns",
    "😳", "# Flushed Face",
    "😔", "# Pensive Face",
    "💨", "# Face with Steam From Nose",
    " thở", "# Face Exhaling",
    "👉", "# Saluting Face",
    "🤰", "# Pregnant Man"
]
```

Tn [24]: # *boomify*

```
print(random emoji insertion)
```

```
"Heute ist ein schöner Tag zum Feiern!", 10, boomer_emojis)) # fügt 10 "boomer" Emojis
```

Heute 🎉 😊 🙏 🎉 🙏 ist 😊 ein schöner Tag zum 👍 🎉 😊 🎉 Feiern!

In [25]: `# zoomify`

```
print(random_emoji_insertion(  
    "Heute ist ein schöner Tag zum Feiern!", 10, zoomer_emojis)) # fügt 10 "zoomer" Emojis
```

Heute 🧑‍💨 😊 🐱 🐱 ist 😊 🧑 ein schöner 🧑 Tag zum 🧑 Feiern! 🧑 😊

Wir wollen seriöseren Text!

Copilot Prompt: get the first chapter of Goethe's faust in German from project gutenberg or something as a string

Musste nur weniges anpassen 🤸

In [26]: `from urllib.request import urlopen`

```
def get_faust_chapter_1():  
    url = "https://www.gutenberg.org/cache/epub/2229/pg2229.txt"  
    with urlopen(url) as f:  
        text = f.read().decode("utf-8")[1000:]  
        # Find the start and end of the first chapter (Zueignung)  
        start = text.find("Zueignung")  
        end = text.find("Vorspiel auf dem Theater")  
        if start != -1 and end != -1:  
            chapter_1 = text[start:end].strip()  
            return chapter_1  
        else:  
            return "Kapitel nicht gefunden."  
  
faust_chapter_1 = get_faust_chapter_1()  
faust_absatz_1 = faust_chapter_1[15:381]
```

In [27]: `# Boomer-Faust`

```
print(random_emoji_insertion(faust_absatz_1, 50, boomer_emojis)) # fügt 50 "boomer" Emojis
```

Ihr 🎉 😊 naht euch wieder, schwankende Gestalten, 😊  
Die 😊 🎉 😊 🎉 🎉 fröhlich sich 🎉 einst dem trüben Blick gezeigt.  
Versuch ich 😊 😊 wohl, 😊 euch diesmal festzuhalten?  
Fühl 😊 ich mein 💯 Herz noch jenem 😊 🙏 🎉 😊 Wahn 😊 😊 😊 geneigt? 🎉 😊  
Ihr drängt euch zu! 😊 nun gut, 😊 💯 so 🎶 😊 🎶 mögt ihr 💯 walten, 🎉  
Wie 😊 😊 😊 🎉 ihr aus Dunst 🎉 und 😊 Nebel um 😊 🎉 mich steigt; 😊  
Mein 😊 🙏 Busen 😊 fühlt sich jugendlich erschüttert  
Vom 🎶 Zauberhauch, der 😊 euren Zug umwittert. 😊 🎶

In [28]: `# Zoomer-Faust`

```
print(random_emoji_insertion(faust_absatz_1, 50, zoomer_emojis)) # fügt 50 "zoomer" Emojis
```

Ihr 🎉 😊 naht euch 🤝 wieder, 🤝 schwankende Gestalten,  
Die früh sich einst dem trüben Blick 😢 gezeigt.  
Versuch 🦋 😊 🐠 ich wohl, euch 🤪 😊 😊 😊 😊 😊 🧑 diesmal festzuhalten? 😢 😊  
Fühl ich 🎉 🤪 mein Herz noch jenem 🎉 😊 Wahn geneigt?  
Ihr 🦋 drängt 😊 😊 euch zu! nun 😊 🌟 🤪 gut, so 😢 mögt 🐠 ihr walten,  
Wie 😢 🤪 😊 🐠 ihr 😊 aus Dunst 🌟 😊 😊 😊 und 🤪 😊 🐠 😊 Nebel um mich steigt; 😢  
Mein Busen 🧑 fühlt 🤖 sich jugendlich 😊 erschüttert  
Vom 😊 Zauberhauch, der euren 🎉 Zug 😢 🐠 umwittert.

## "Sinnvollere" Emojis

Wir Wörter ersetzen durch *passende* Emojis. Wir berechnen hier nur die Wortähnlichkeit auf Zeichenebene, also ist Maus ähnlich zu Haus aber nicht zu Ratte .

Es ist prinzipiell nicht schwer, semantische Ähnlichkeit (mit Word-Embeddings, etc.) zu berechnen, aber die Installation der Bibliotheken und Modelle dauert zu lange für einen Workshop.

```
In [29]: # Create a dictionary mapping German words to emojis based on emoji.EMOJI_DATA
# Remove the colons from the emoji names

german_to_emoji = {}

for emj in emoji.EMOJI_DATA:
    # Get the emoji short name (e.g., ":chicken:")
    short_name = emoji.EMOJI_DATA[emj]['de'].replace(':', '').replace('_', ' ').replace('flag', '')
    german_to_emoji[short_name] = emj

# Example: print a few entries
for k, v in list(german_to_emoji.items())[-10:]:
    print(f"{k}: {v}")
```

```
gelbes herz: ❤️
gelbes quadrat: 🟡
yen-banknote: 💰
yin und yang: ☽
jo-jo: 🎭
irres gesicht: 🤡
zebra: 🦓
gesicht mit reißverschlussmund: 😢
zombie: 🧟
flagge ålandinseln: 🇫🇮
```

```
In [30]: from difflib import get_close_matches
```

```
def get_closest_emojis(word, word_dict=german_to_emoji, n=1, cutoff=0.55):
    candidates = list(word_dict.keys())
    matches = get_close_matches(word, candidates, n=n, cutoff=cutoff)
    emojis = [word_dict[word] for word in matches]
    return word, matches, emojis
```

```
In [31]: get_closest_emojis("Maus", n=10)
```

```
Out[31]: ('Maus',
['maus', 'haus', 'pause', 'kaktus', 'auster', 'abakus', 'bus'],
['MouseEvent', 'House', 'Pause', 'Cactus', 'Asterisk', 'Abacus', 'Bus'])
```

```
In [32]: import random
```

```
def emojify_text(text, n=1, percentage=25):
    # Split text into lines to preserve line breaks
    lines = text.splitlines()
    all_words = []
    word_indices = []
    # Flatten words and keep track of their (line_idx, word_idx) positions
    for line_idx, line in enumerate(lines):
        words = line.split()
        for word_idx, word in enumerate(words):
            all_words.append(word)
            word_indices.append((line_idx, word_idx))
    num_to_replace = int(len(all_words) * percentage / 100)
    indices = random.sample(range(len(all_words)), num_to_replace)
    # Replace selected words with emojis
    for idx in indices:
        word = all_words[idx]
        _, _, emojis = get_closest_emojis(word, n=n)
        if emojis:
            all_words[idx] = random.choice(emojis)
    # Reconstruct lines with replaced words
    new_lines = []
```

```

word_ptr = 0
for line in lines:
    words = line.split()
    for i in range(len(words)):
        words[i] = all_words[word_ptr]
        word_ptr += 1
    new_lines.append(' '.join(words))
return '\n'.join(new_lines)

```

In [33]: test\_text = "Ich bin sportlich, zuverlässig, attraktiv, schlank, zärtlich, liebenvoll, einfühlsam, treu, ehrlich und vor allen Dingen kein Mensch, der alleine leben will.."

```

print(emojify_text(test_text, percentage=0))
print(emojify_text(test_text, percentage=25))
print(emojify_text(test_text, percentage=50))
print(emojify_text(test_text, percentage=100))

```

Ich bin sportlich, zuverlässig, attraktiv, schlank, zärtlich, liebenvoll, einfühlsam, treu, ehrlich und vor allen Dingen kein Mensch, der alleine leben will..

– bin sportlich, zuverlässig, attraktiv, schlank, zärtlich, liebenvoll, einfühlsam, treu, ehrlich und vor allen Dingen 🧑‍🦰 🌎 der alleine leben will..

Ich 🏃‍♂️ sportlich, zuverlässig, attraktiv, schlank, zärtlich, 🍊 einfühlsam, treu, ehrlich und 📖 🦑 🥈 kein Mensch, der alleine 🏃‍♂️ will..

– 🏃‍♂️ 🌎 🌸 zuverlässig, 🌟 💬 zärtlich, 🍊 einfühlsam, 🍰 🦌 💋 📖 🦑 🥈 🌟 🌎 🌸 ~ 🏃‍♂️ wi...

In [34]: print(emojify\_text(faust\_chapter\_1))

Zueignung

Ihr naht euch wieder, schwankende 🤷‍♂️  
 🐾 früh sich einst 🐱 trüben Blick gezeigt.  
 🐴 ich wohl, euch diesmal festzuhalten?  
 Fühl 🌟 mein Herz noch jenem 💔 geneigt?  
 Ihr drängt 🦌 zu! nun gut, so mögt ihr walten,  
 Wie 🌞 🐿 Dunst und Nebel um 🐠 steigt;  
 Mein Busen fühlt sich jugendlich erschüttert  
 Vom Zauberhauch, 🌺 🚶 Zug umwittert.

Ihr bringt mit euch die Bilder froher Tage,  
 Und manche liebe Schatten 🧑‍🦰 auf;  
 Gleich einer 🗓 halbverklungenen 🏹  
 ⚡ erste Lieb und Freundschaft mit 📜 RBE  
 ♥ Schmerz wird neu, es wiederholt die Klage  
 🍪 Lebens labyrinthisch 🚶 Lauf,  
 Und 🌿 die Guten, die, um schöne Stunden  
 Vom Glück 🦄 vor mir hinweggeschwunden.

hog 🚶 🧑‍🦰 die 🇮🇳 Gesänge,  
 Die Seelen, denen 🐠 die ersten sang;  
 Zerstoben ist 🎵 freundliche Gedränge,  
 Verklungen, ach! 🌺 🎵 Widerklang.  
 Mein Lied ertönt der unbekannten Menge,  
 Ihr Beifall 🎉 macht 🎵 Herzen bang,  
 Und was 🙅 sonst an 🎵 Lied erfreuet,  
 Wenn es 🕸 lebt, irrt in der Welt 🧑‍🦰

hog 🐠 🐾 🐱 ein 🎵 entwöhntes 🍀  
 Nach 🌿 stillen, ernsten Geisterreich,  
 Es schwebet nun in 🎵 🍀  
 Mein lispelnd Lied, der Äolsharfe gleich,  
 Ein Schauer faßt mich, Träne folgt den Tränen,  
 Das 🧑‍🦰 Herz, es 🌼 sich mild und weich;  
 Was ich besitze, seh ich 🌟 im Weiten,  
 Und was 🐦 wird mir zu 🧑‍🦰

```
In [35]: print(emojify_text(faust_chapter_1, percentage=100))
```

1

```
In [36]: emoji_text = '✋ 🎉 🐂 🐃 🐑 🐄 🐆 🐈 '
          emoji.demojize(emoji_text, language='de')
```

```
Out[36]: ':ohr: :dna: :elch: :widder: :schwan: :statue:'
```

```
In [37]: for idx, line in enumerate(emojify_text(faust_absatz_1, percentage=100).splitlines()[:5]):  
    print(line)  
    print(emoji.demojize(line, language='de'))  
    print(faust_absatz_1.splitlines()[idx])  
    print()
```



:ohr: :dna: :elch: :widder: :schwan: :statue:  
Ihr naht euch wieder, schwankende Gestalten,



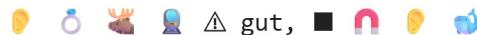
:igel: :frosch: :schal: :fenster: :oden: :trauben: Blick :geige:  
Die fröhlich sich einst dem trüben Blick gezeigt.



:hirsch: :fisch: :wolf: :elch: :lineal: :stuhl:  
Versuch ich wohl, euch diesmal festzuhalten?



Fühl :fisch: :bein: :herz: :loch: :ente: :zahn: :geige:  
Fühl ich mein Herz noch jenem Wahn geneigt?



:ohr: :ring: :elch: :zug: :warnung: gut, :stopp: :magnet: :ohr: :wal:  
Ihr drängt euch zu! nun gut, so mögt ihr walten,

```
In [38]: print(emojify_text(test_text, percentage=100))
print(emoji.demojize(emojify_text(test_text, percentage=100), language='de'))
print(test_text)
print()
```

- 🌞 🌟 🌸 zuverlässig, 🌸 🌈 zärtlich, 🍂 einfühlsam, 🎉 🦌 🍁 🍀 🍃 🍅 🍇 🍈 🍉 wi  
ll..  
:loch: :bein: :sportschuh: zuverlässig, :radioaktiv: :schlange: zärtlich, :zwiebel: einfühlsa  
m, :torte: :elch: :mund: :tor: :quelle: :ingwer: :kind: :tropenfisch: :feder: :wellenlinie: :b  
ein: will..  
Ich bin sportlich, zuverlässig, attraktiv, schlank, zärtlich, liebevoll, einfühlsam, treu, ehr  
lich und vor allen Dingen kein Mensch, der alleine leben will..

## Geheimschrift

"Professionell" programmierte Emoji-Enigma-Maschine! Solange beide Parteien die richtige Geheimzahl haben (und den Code), können Nachrichten in Emojisequenzen übersetzt und rückübersetzt werden.

```
In [39]: class emojigema:
    """
    Encode and decode text as a sequence of emojis using a fixed emoji alphabet and length marker
    """

    LENGTH_MARKERS = ["😊", "😊", "😊", "🤣", "😊", "😊", "😊", "😊"] # up to 8 emoji-length markers

    def __init__(self, Geheimzahl):
        self.seed = Geheimzahl
        # Only use single-codepoint, non-flag, non-compound emojis
        self._emoji_list = [
            e for e in emoji.EMOJI_DATA
            if len(e) == 1 and not emoji.EMOJI_DATA[e]['en'].startswith(':flag_')
        ]
        self._emoji_list = [e for e in self._emoji_list if e not in self.LENGTH_MARKERS]
        self.base = len(self._emoji_list)

    def _get_rotated_emoji_list(self):
        offset = self.seed % self.base
        return self._emoji_list[offset:] + self._emoji_list[:offset]

    def _int_to_emojis(self, codepoint, emoji_list):
        n = 1
        max_val = self.base
        while codepoint >= max_val:
            n += 1
```

```

        max_val *= self.base
    val = codepoint
    emojis = []
    for _ in range(n):
        emojis.append(emoji_list[val % self.base])
        val //= self.base
    return self.LENGTH_MARKERS[n-1] + ''.join(reversed(emojis))

def _emojis_to_int(self, emoji_seq, emoji_list):
    n = self.LENGTH_MARKERS.index(emoji_seq[0]) + 1
    val = 0
    for e in emoji_seq[1:n+1]:
        val = val * self.base + emoji_list.index(e)
    return val, n+1

def encode(self, text):
    emoji_list = self._get_rotated_emoji_list()
    return ''.join(self._int_to_emojis(ord(c), emoji_list) for c in text)

def decode(self, emoji_text):
    emoji_list = self._get_rotated_emoji_list()
    i, decoded = 0, []
    while i < len(emoji_text):
        marker = emoji_text[i]
        if marker not in self.LENGTH_MARKERS:
            raise ValueError(f"Invalid marker emoji at position {i}: {marker}")
        n = self.LENGTH_MARKERS.index(marker) + 1
        seq = emoji_text[i:i+n+1]
        cp, consumed = self._emojis_to_int(seq, emoji_list)
        decoded.append(chr(cp))
        i += consumed
    return ''.join(decoded)

```

In [40]: verfassermaschine = emojigma(Geheimzahl=42)  
text = "Oliver"

```

Geheimtext = verfassermaschine.encode(text)          # Text in Geheimschrift übersetzen
print(Geheimtext)
print()

```

In [41]: übersetzt = verfassermaschine.decode("😊ビル😊 биз😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊") # Text aus Geheimschrift übersetzen  
print(übersetzt)

Oliver

In [42]: empfängermaschine = emojigma(Geheimzahl=20) # Hier wird die Geheimschrift nicht richtig entziffern
print(empfängermaschine.decode("😊ビル😊 biz😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊 😊"))

e{

## LLMs

Experimentiert gerne mit LLMs!

- <https://www.chatgpt.com>
- <https://www.perplexity.ai/>
- <https://gemini.google.com/app>
- <https://copilot.microsoft.com>
- uvm

## Beispielprompts:

- Ziehe vom Emoji 🧑‍🤝‍🧑 schrittweise jeweils eine UTF-8 Bytesequenz ab
- Erkläre den Klimawandel nur in Emojis
- Verschönere meinen Text mit passenden Emojis
- Was wären die Lieblingsemojis von Friedrich Schiller

### Der **Klimawandel**



\_\_\_\_\_ Die **Lieblingsemojis** von Dschingis und Oliver K[ah]ha[n]

## Dschingis Khan (1162–1227) – Mongolischer Eroberer

1. 🐾 (Pferd) – Symbolisiert die mongolische Reitkunst und Mobilität.
2. ✨ (Sword) – Steht für Krieg, Eroberung und Macht.
3. 💯 (Bogen und Pfeil) – Die Mongolen waren berühmte Bogenschützen.
4. 🌎 (Weltkugel Asien/Australien) – Für sein riesiges, erobertes Reich.
5. 🏰 (Krone) – Als Herrscher und „Khan“.
6. 🏜 (Zelt) – Nomadisches Leben der Mongolen.
7. 🐾 (Galoppierendes Pferd) – Nochmals die Bedeutung des Pferdes.

## Oliver Kahn (geb. 1969) – Deutscher Fußballtorwart und TV-Experte

1. ⚪ (Torwarthandschuh) – Symbol für seine Position als Torwart.
2. ⚽ (Fußball) – Klar, als Fußballlegende.
3. 🦁 (Löwe) – Sein Spitzname war „Der Titan“ oder „Der Löwe“.
4. 🗣 (Sprechblase) – Für seine markanten Interviews und Sprüche.
5. 🏆 (Pokal) – Für seine vielen Titel und Erfolge.
6. 💪 (Starker Arm) – Für seine kämpferische Einstellung.
7. 😡 (Wütendes Gesicht) – Für seine legendären Emotionen auf dem Platz.

## RUB-GPT API

- Für erfahrene Anwender\*innen
- Zugriff nur aus dem Uninetz oder per VPN
- API-Token nötig
- API\_TOKEN und API\_URL können auch durch die von OpenAI ersetzt werden

```
In [43]: API_TOKEN = "API TOKEN UNTER https://gpt.ruhr-uni-bochum.de/app/settings generieren"
API_URL = "https://gpt.ruhr-uni-bochum.de/external/v1/chat/completions"
MODEL = "gpt-4.1-2025-04-14"
```

```
In [44]: import subprocess
import json

def send_to_gpt(prompt, model=MODEL, timeout=30):
    headers = [
        "Content-Type: application/json",
        f"Authorization: Bearer {API_TOKEN}"
    ]
    data = {
        "model": model,
        "messages": [
            {"role": "user", "content": prompt}
        ]
    }
    curl_command = [
        "curl", "-s", API_URL,
        "-H", headers[0],
        "-H", headers[1],
        "-d", json.dumps(data)
    ]
    try:
```

```

result = subprocess.run(curl_command, capture_output=True, text=True, timeout=timeout)
try:
    response_dict = json.loads(result.stdout)
except json.JSONDecodeError:
    response_dict = {"error": "Invalid JSON response", "raw": result.stdout}
except subprocess.TimeoutExpired:
    response_dict = {"error": "Request timed out after {} seconds".format(timeout)}
return response_dict

# Example usage:
# response = send_to_gpt("Hello, GPT! How are you?")

```

## Beispielsätze

In [45]:

```

from urllib.request import urlopen
from random import choice

def download_sents(url):
    with urlopen(url) as f:
        sents = [sw.decode("UTF-8").strip()[1].upper() +
                 sw.decode("UTF-8").strip()[1:] + "." for sw in f.readlines()]
    return sents

männersätze = download_sents(r"https://raw.githubusercontent.com/gambolputty/textstelle/master/männer_sätze.txt")
frauensätze = download_sents(r"https://raw.githubusercontent.com/gambolputty/textstelle/master/frauen_sätze.txt")

def gib_satz(gender="all"):
    if gender == "m":
        return choice(männersätze)
    elif gender == "f":
        return choice(frauensätze)
    else:
        return choice(männersätze+frauensätze)

```

In [46]:

```
sents = [gib_satz() for _ in range(5)]
```

Out[46]:

```

['Ich bin ein lebensfroher, lustiger und immer gut gelaunter Mensch..',
 'Ich bin ein positiver und lebenslustiger Mensch, der gute Gespräche zu schätzen weiß..',
 'Ich bin ein leicht zu begeisternder Mensch..',
 'Ich bin ein positiver und herzlicher Mensch, der auch schwierigen Situationen stets das Lachen behält..',
 'Ich bin auf jeden Fall respektvoll und Du kannst Dir (als vertrauensbildende Maßnahme) auch gern vorher meine öffentliche Mitarbeiterseite im Netz anschauen..']

```

In [47]:

```

from urllib.request import urlopen

prompt = "Ergänze den Satz mit passenden Emojis, antworte nur mit dem Satz: {}"
# in die Klammern wird jeweils der Satz aus der Liste von oben eingefügt

responses = [send_to_gpt(prompt.format(s)) for s in sents]
if not responses[0].get('error') == 'Invalid JSON response':
    print('Response from GPT API:\n')
    for i, r in enumerate(responses):
        if 'choices' in r:
            pair = f'{sents[i]}\n{r["choices"][0]["message"]["content"]}\n'
            print(pair)
else:
    print('Could not access GPT API (Did you save your API Key above?). Displaying fallback example')
    url = "https://raw.githubusercontent.com/ODEck/teaching/main/emergency_backup.txt"
    with urlopen(url) as f:
        print(f.read().decode("utf-8"))

```

Could not access GPT API (Did you save your API Key above?). Displaying fallback example:

Ich bin interessiert an Menschen, die tolerant, nicht auf den Kopf und auf den Mund gefallen sind und die meine Interessen teilen..

Ich bin interessiert an Menschen, die tolerant 🤝, nicht auf den Kopf 😮 und auf den Mund 🍻 gefallen sind und die meine Interessen teilen 🎨🎶.

Ich bin eine Frau mit Hirn, Herz und Verstand und suche einen Partner, der das Herz am richtigen Fleck hat und nicht auf den Kopf gefallen ist..

Ich bin eine Frau mit 💁‍♀️💖 und Verstand und suche einen Partner, der das Herz am richtigen Fleck ❤️ hat und nicht auf den Kopf gefallen ist. 🥰

Ich bin Papa von 2 wundervollen Kindern, die mir alles bedeuten..

Ich bin Papa von 2 wundervollen Kindern 👶👶, die mir alles bedeuten ❤️.

Ich bin charakterfest, weiß was ich will und würde mich nie für jmd..

Ich bin charakterfest 💪, weiß was ich will 💪 und würde mich nie für jmd. 🙅 ändern.

Ich bin sehr lebensfroh und weiß was ich vom Leben möchte! 🎯🌟

Ich bin sehr lebensfroh ☀️😊 und weiß, was ich vom Leben möchte! 🎯🌟