NEUER BEFEHL: PWD

Befehl

Kompletter Pfad des aktuellen Ordners

```
Leonie@Ubuntu:~/Python/Übungsblatt1 $ pwd
/home/leonie/Python/Übungsblatt1
Leonie@Ubuntu:~/Python/Übungsblatt1 $
```

PYTHON VERSIONEN

- Es gibt zwei Python Versionen, Python 2 und Python 3
- Python 2 und Python 3 sind nicht kompatibel!
- In der Konsole wird `python` zu `python2` vervollständigt, deswegen Python3-Programme immer mit **`python3`** aufrufen
- Bei Internetrecherchen aufpassen, ob sich die Quelle auf Python 2 oder 3 bezieht

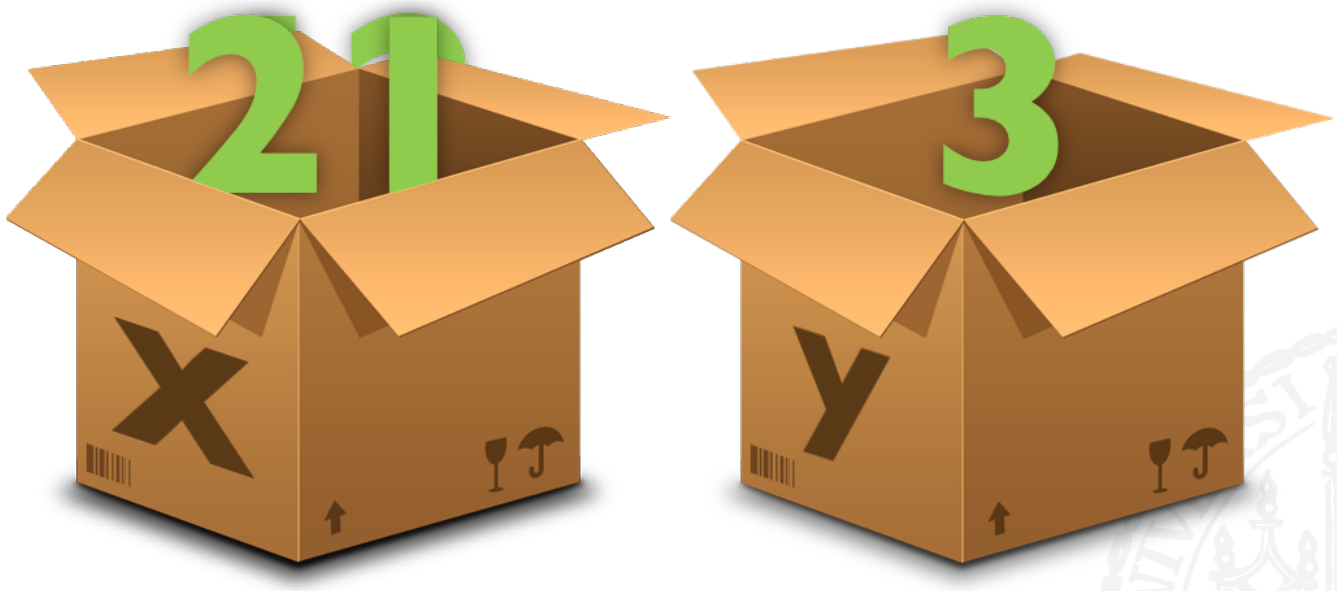
SHEBANG

```
#!/usr/bin/python3
#Aufgabe 1-2
#WS 2016/17
#Autorin: Leonie Weißweiler
print ('Hello World')
```

- Beispielprogramm 1-4.py wie beim letzten Mal einfügen
- #!/usr/bin/python
- Falls wir das Programm ausführbar machen wollen damit es ohne python3 aufgerufen werden kann
- Die Shebang line sagt dem Betriebssystem, wo es den Interpreter für python finden kann

VARIABLEN

```
>>> x = 42
>>> print(x)
42
>>> x = 21
>>> print(x)
21
>>> y = 3
>>> print(y)
3
>>> print(x)
21
```



VARIABLEN

```
>>> straße = "Oettingenstraße"
>>> hausnummer = 67
>>> print(straße + hausnummer)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

DATENTYPEN

- Jeder Wert hat einen Typ, zum Beispiel:
  - Integer (3, 42, -100)
  - String (“Hallo Welt”, “CIS”)
- Typen verhalten sich unterschiedlich:
  - 3+3=6,
  - “hallo” + “welt” = “hallo welt”
  - “3” + “3” = “33”

DATENTYPEN

Datentyp	Inhalt	Operatoren
`integer`	Ganze Zahl	`+ - * / > < <= >=`
`float`	Kommazahl	
`string`	Text	`+ *`
`boolean`	Wahrheitswert (`True` oder `False`)	`&& || !`

VERGLEICHE

- Man kann Werte mit passenden Typen vergleichen und erhält boolean Werte
- ```
  >>> 3 < 5
  True
  ```
- ```
  >>> 3 < 5.4
  True
  ```
- ```
  >>> "a" < "b“
  True
  ```
- ```
  >>> 3 < “3“
  Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: unorderable types: int() < str()
  ```

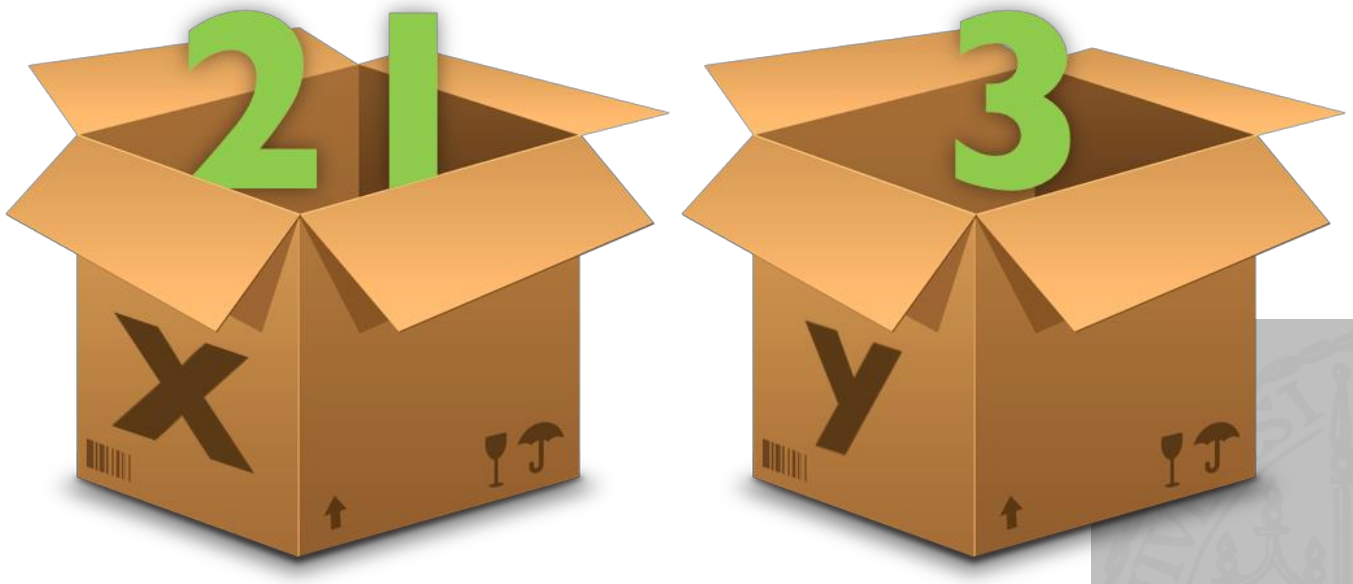
TYPE CASTING

- Man kann manche Werte zwischen Typen konvertieren (“Casten”)
- ```
  >>> int(5.6)
  5
  ```
- ```
  >>> str(4)
  ‘4’
  ```
- ```
  >>> int(“54”)
  54
  ```
- ```
  >>> int("Max")
  Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  ValueError: invalid literal for int() with base 10: 'Max'
  ```

TYPE CASTING

- Man kann manche Werte zwischen Typen konvertieren (“Casten”)
- ```
  >>> bool(0)
  False
  ```
- ```
  >>> bool(1)
  True
  ```
- ```
  >>> bool(-42.5)
  True
  ```
- ```
  >>> bool(“”)
  False
  ```
- ```
  >>> bool(“abc”)
  True
  ```
  ```

WIEDERHOLUNG - VARIABLEN



WIEDERHOLUNG - VARIABLEN

```
Clemens = ‘Simone’
Simone = ‘Felix’
Annabelle = Clemens + Simone
print(Felix)
```

ERROR: Die Variable Felix existiert nicht

```
print(Annabelle)
```

SimoneFelix

WIEDERHOLUNG - DATENTYPEN

Datentyp	Inhalt	Operatoren
`integer`	Ganze Zahl	`+ - * / %` `> < <= >= ==`
`float`	Kommazahl	
`string`	Text	`+ * == < > <= >=`
`boolean`	Wahrheitswert (**`True`** oder **`False`**)	`&& || ! ==`

WIEDERHOLUNG - DATENTYPEN

```
x = 4
y = 5.0
z = ‘6’
a = ‘7’
print(x+y)
```

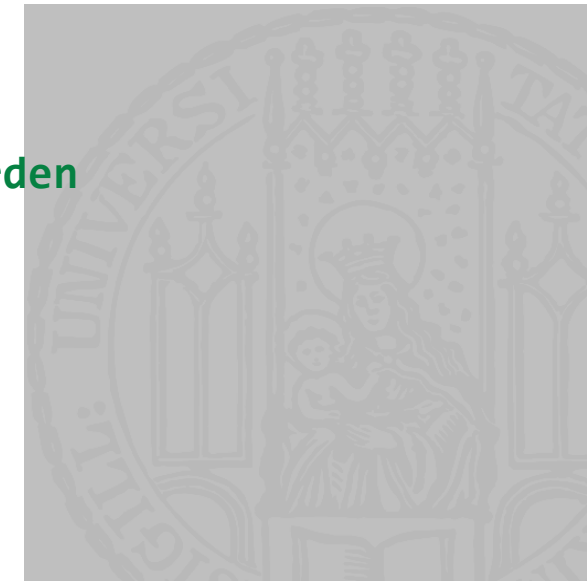
9.0

```
print(y+z)
```

FEHLER: Float und String können nicht addiert werden

```
print(z+a)
```

67



UNIX - PFADE

- Ein Pfad gibt den Ort einer Datei oder eines Ordners an.
- Ein Pfad kann **absolut** oder **relativ** sein.
- `cd ../privat/sicherung`
- `kate hello.py`
- `kate ./hello.py`
- `python3 /home/weissweiler/programme/weltherrschaft.py`

UNIX - PFADE

```
Leonie@Laptop:Seminar $ python3
../programme/lmu.py
```

```
Leonie@Laptop:Seminar $ pwd
/home/leonie/Seminar
Leonie@Laptop:Seminar $ cd ..
Leonie@Laptop:~ $ pwd
/home/leonie/
Leonie@Laptop:~ $ cd programme
Leonie@Laptop:programme $ pwd
/home/leonie/programme
Leonie@Laptop:Seminar $ ls
lmu.py
Leonie@Laptop:Seminar $ python3 lmu.py
```

UNIX - PFADE

- Relative Pfade starten beim aktuellen Verzeichnis
- Absolute Pfade starten im Root-Verzeichnis (“ganz oben”) und beginnen deswegen mit einem **/**
- Wir befinden uns im Ordner `/home/leonie/seminar`
- Relativer Pfad: `../privat/lmu.py`
- Absoluter Pfad: `/home/leonie/seminar/../privat/lmu.py = /home/leonie/privat/lmu.py`

UNIX - MAN PAGES

- man steht für "Manual", also "Anleitung"
- Man kann zu jedem Befehl eine Anleitung aufrufen, in der Befehl und Optionen erklärt werden
- `man ls`
- `man pwd`
- `man python3`
- In einer man page bewegt man sich mit den Pfeiltasten und beendet mit Q

UNIX - WILDCARDS

- Beim Angeben von Dateinamen kann man Platzhalter, sog. Wildcards verwenden
- * steht für beliebig viele beliebige Zeichen
- ? steht für genau ein beliebiges Zeichen

UNIX - WILDCARDS

```
Leonie@Laptop:Seminar $ ls
Hallo.py  Hello.py  Halloho.txt

Leonie@Laptop:Seminar $ ls *
Hallo.py  Hello.py  Halloho.txt

Leonie@Laptop:Seminar $ ls *.py
Hallo.py  Hello.py

Leonie@Laptop:Seminar $ ls ?????.py
Hallo.py  Hello.py
```

```
Leonie@Laptop:Seminar $ ls H?llo.py
Hallo.py  Hello.py

Leonie@Laptop:Seminar $ ls Hall*
Hallo.py  Halloho.txt

Leonie@Laptop:Seminar $ ls H?llo*
Hallo.py  Hello.py Halloho.txt

Leonie@Laptop:Seminar $ ls a*
```

FEHLERKORREKTUR

- Es stellt sich heraus, dass == für alle Datentypen definiert ist
- Es gibt für inkompatible Datentypen (wie `String` und `int`) **`false`** zurück
- Für `<, >` und ähnliches wird trotzdem ein Fehler ausgegeben

PIPING

- Mit "|" wird in Bash der output eines Befehls als Eingabe einem anderen Befehl übergeben
- Es kopiert die Ausgabe des linken Programmes und "tippt" sie als Eingabe in das recht ein
- Unendlich verkettbar

```
Leonie@Laptop $ cat inhalt.txt
c
a
f
b

Leonie@Laptop $ cat inhalt.txt | sort
a
b
c
f

Leonie@Laptop $ cat inhalt.txt | sort | ...
```

REDIRECTING

- Mit “>” wird in Bash die Ausgabe des Programmes abgefangen und in eine Datei geschrieben
- Eine evtl. extistierende Datei wird **überschrieben**
- `cat datei.txt > datei2.txt`
- Mit “>>” wird in Bash die Ausgabe des Programmes abgefangen und an eine Datei angehängt
- `echo “Hallo” >> begrüßungen.tx`
- `echo “Hi” >> begrüßungen.txt`

PIPING VS REDIRECTING

Befehl	Piping	Redirecting	Appending
Operator	|	>	>>
Quelle	Programm	Programm	Programm
Ziel	Programm	Datei (überschreibend)	Datei (anhängend)

WHILE-SCHLEIFE

- While-Schleifen kann man benutzen, um Anweisungen zu wiederholen
- ```
  x = 0
  while (x < 5):
      print(x)
      x = x + 1
  ```
- ```
  y = 5
  while (y > 0):
      print(y)
      y = y + 1
  ```

LISTEN

- Bis jetzt kennen wir normale Variablen
- Sie können sich genau eine Sache merken, und wenn wir ihnen eine andere übergeben, vergessen sie die erste.
- Wenn wir jetzt aber mehrere zusammengehörige Werte gleichzeitig in einer Variable speichern wollen, brauchen wir eine **Liste**
- ```
  x0 = 0
  x1 = 0
  x2 = 0
  x3 = 0
  x...
  ```


  ```

LISTEN

```
zahlen = [4,9,42]

print (zahlen[0])
    4

zahlen[0] = 11

print (zahlen[3])
    Fehler

zahlen[3] = 0
    Fehler
```

0	1	2
4	9	42

0	1	2
11	9	42

LISTEN

```
zahlen = []
```

```
zahlen.append(3)
```

0
3

```
zahlen.append(-4)
```

0	1
3	-4

```
zahlen.append(90)
```

0	1	2
3	-4	90

LISTEN

```
len(zahlen)
    3

print(zahlen[-1])
    90

zahl = 3

zahl.append(4)
    Fehler
```

0	1	2
3	-4	90

3

3

SLICING

- Mit slicing kann man sich eine "Scheibe" aus einem Array "schneiden"
- Der linke Index wird hierbei "eingeschlossen", der rechte "ausgeschlossen"
- `zahlen = [1,2,4,8,16,32]`
- `zahlen[2:4]`
- `zahlen[1:-1]`
- `zahlen[3:]`
- `zahlen[:-3]`

0	1	2	3	4	5
1	2	4	8	16	32

MEHR SLICING

- `zahlen = [1,2,4,8,16,32]`
- `zahlen[::2]`

“jedes zweite”

0	1	2	3	4	5
1	2	4	8	16	32

MEHR SLICING

- `zahlen = [1,2,4,8,16,32]`
- `zahlen[::2]`
- `zahlen[::-1]`

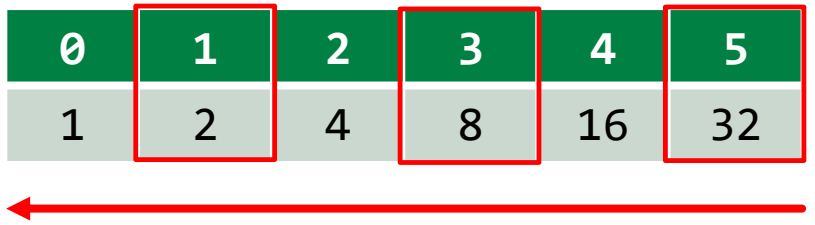
“Jedes erste von hinten”

0	1	2	3	4	5
1	2	4	8	16	32

MEHR SLICING

- `zahlen = [1,2,4,8,16,32]`
- `zahlen[::2]`
- `zahlen[::-1]`
- `zahlen[::-2]`

“jedes zweite von hinten”



FOR SCHLEIFEN

- Mit For Schleifen kann man die Elemente einer Liste durchgehen (iterieren)
- Man vermeidet komplizierte, unlesbare while schleifen

```
zahlen = [5,6,7,8]
position = 0
while (position < len(zahlen)):
    z = zahlen[position]
    print(z)
```

```
zahlen = [5,6,7,8]

for z in zahlen:
    print(z)
```

FOR SCHLEIFEN

- ```
  for z in zahlen:
      print(z)
  ```
- **`for`** leitet die Schleife ein
- **`z`** ist die Variable die nacheinander alle Werte annimt
- **`in`** kündigt die Liste an
- **`zahlen`** ist die Liste aus der die Werte kommen
- **`:`** kündigt den codeblock an
- *„Führe den folgenden Code immer wieder aus und lass z jedes mal einen anderen Wert aus der Liste sein“*
  ```

FOR SCHLEIFEN

```
zweierpotenzen = [1,2,4,8,16]
for a in zweierpotenzen:
    print(a)
```

```
1
2
4
8
16
```

RANGES

- Mit `range(x)` kann man sich automatisch eine Range von Zahlen generieren lassen
- Ranges verhalten sich wie Listen, sind aber keine Listen **(!)**
- ```
  range(5)
  [0,1,2,3,4]
  ```
- ```
  range(10,15)[4]
  14
  ```
- ```
  for x in range(5)
      print(x, end=' ')
  0 1 2 3 4
  ```
  ```

RANGES

- Die Parameter verhalten sich wie die Parameter beim Slicing
- `range(3)` ~ *`[0,1,2]`*
- `range(4,9)` ~ *`[4,5,6,7,8]`*
- `range(10,0)` ~ *`[]`*
- `range(0,10,2)` ~ *`[0,2,4,6,8]`*

ACHTUNG SELTSAM!

- Um eine Range "rückwärts" zu definieren (oder einen Teil einer Liste rückwärts zu slicen) muss im dritten Argument ein Minus stehen **und** die ersten beiden Argumente vertauscht sein!
- `range(0,10,-1)`
- `range(10,0)`
- `range(10,0,-1) [10,9,8,7,6,5,4,3,2,1]`

IMMUTABLES VS MUTABLES

- Es gibt zwei Arten von Typen in Python: Mutables und Immutables
- Nur Variablen deren Typ mutable ist, können durch Methoden verändert werden
- Variablen deren Typ immutable ist, können nur neu belegt werden
- Zahlen sind immutable:
- ```
  i = 3
  i = i + 4
  ```
- `Listen sind mutable:`
- ```
  l = [0,7,2,5,1]
  l.sort()
  ```

IMMUTABLES VS MUTABLES

Immutable	Mutable
• int • float • string • boolean • range	• list

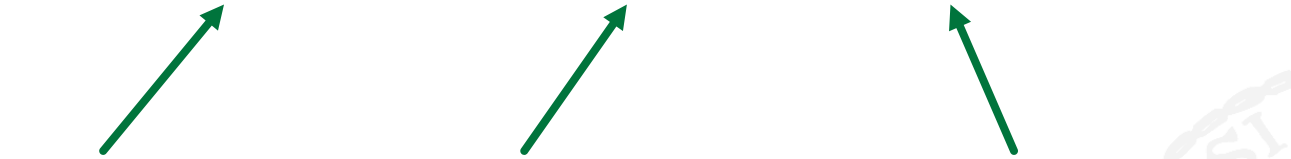
DATEIEN

- Man kann aus einem Python Programm Dateien lesen und schreiben
- Wir kennen Dateien schon aus Bash
- `echo "We demand a shrubbery!" > knight.txt`

FILEHANDLES

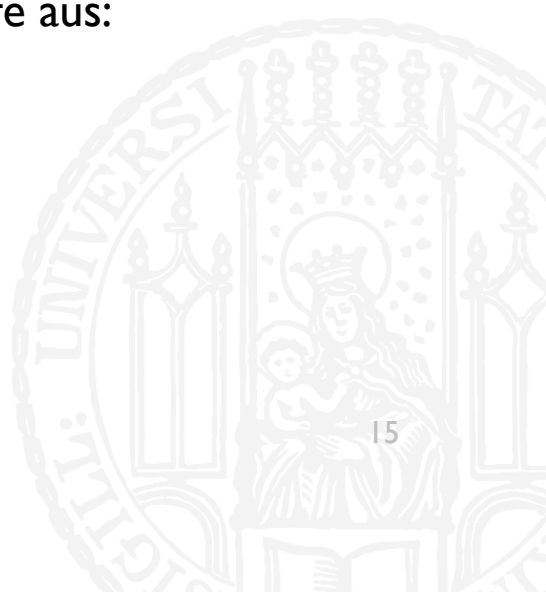
- Man greift auf eine Datei zu, indem man ein **Filehandle** öffnet
- Ein Filehandle ist wie eine Variable über die man mit der Datei kommuniziert

```
datei = open('romanes.txt', 'r')
```



FILEHANDLES

- Aus einem Filehandle kann man dann den gesamten Dateiinhalt in einen String lesen
- `text = datei.read()`
- Man kann auch eine einzelne Zeile anfordern
- `text = datei.readline()`
- Am besten liest man die Datei zeilenweise mit einer for Schleife aus:
- ```
  for line in datei:
      print(line)
  ```
- Am ende schließt man die Datei
- `datei.close()`


  ```

FILEHANDLES


```
Leonie@Laptop:~ $ python test.py
Dies
ist
eine
Testdatei

Leonie@Laptop:~ $
```

- Liest man eine Datei so aus, hängt an jedem String noch eine Newline
- Dadurch ergeben sich beim Ausgeben doppelte Newlines und Probleme beim vergleichen
- Man entfernt diese mit strip:
- ```
  for line in datei:
      line = line.strip()
      print(line)
  ```
  ```

FILEHANDLES

- Man kann mit Filehandles auch schreiben
- **'w'** (*Write*) lässt einen in die Datei schreiben, und überschreibt sie ggf.
- `ausgabe = open('export.txt', 'w')`
- **'a'** (*Append*) lässt einen an eine bestehende Datei anhängen
- `ergebnis = open('results.txt', 'a')`

FILEHANDLES

- In ein beschreibbares Filehandle kann man Strings schreiben
- Im Gegensatz zu print() muss Newlines manuell einfügen
- Mehrere Strings müssen mit + zusammengefügt werden
- int o.ä. müssen zuerst umgewandelt werden

```
file = open('knights.txt', 'w')
file.write('Knights of the Coconut\n')
file.write('Knights who ' + 'say Ni\n')
file.write(str(5000) + ' other kinds of Knights\n')
file.close()
```

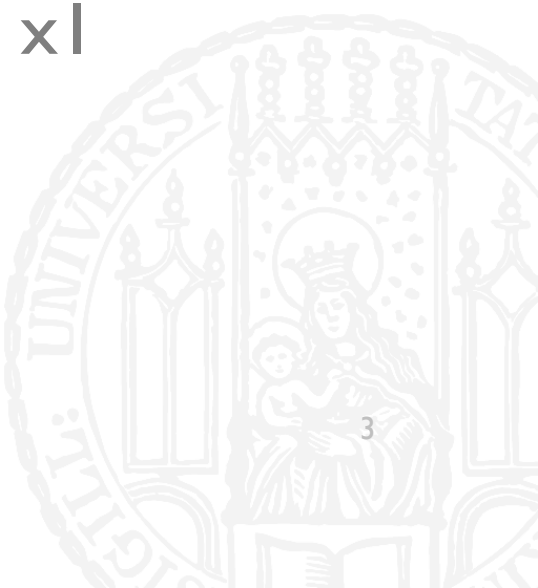
ZAHLENSYSTEME

- Normalerweise schreibt man Zahlen im sog. Dezimalsystem

4 2 6 9

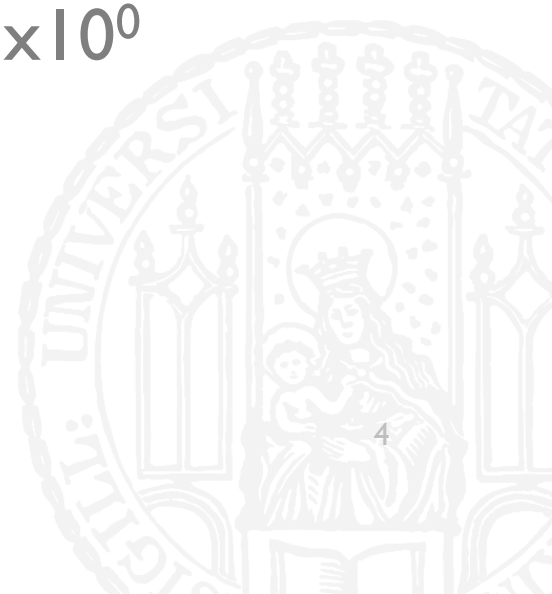
ZAHLENSYSTEME

4	**2**	**6**	**9**
x100	x100	x10	x1



ZAHLENSYSTEME

4	**2**	**6**	**9**
$\times 10^3$	$\times 10^2$	$\times 10^1$	$\times 10^0$

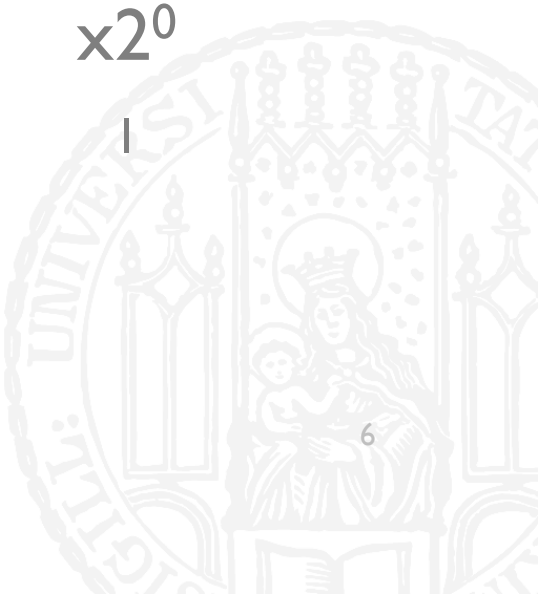


2 ANSTATT 10

- Computer können nur mit Booleans Rechnen
- True oder False
- Kekse oder keine Kekse
- Spam oder kein Spam
- Strom oder kein Strom
- Ladung oder keine Ladung
- 1 oder 0

BINÄRSYSTEM

0	1	0	1	0	1	0
$x2^6$	$x2^5$	$x2^4$	$x2^3$	$x2^2$	$x2^1$	$x2^0$
64	32	16	8	4	2	1



BINÄRSYSTEM

0 1 0 1 0 1 0

$\times 2^6 \quad \times 2^5 \quad \times 2^4 \quad \times 2^3 \quad \times 2^2 \quad \times 2^1 \quad \times 2^0$

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

BINÄRSYSTEM

0 1 0 1 0 1 0

$\times 2^6$ $\times 2^5$ $\times 2^4$ $\times 2^3$ $\times 2^2$ $\times 2^1$ $\times 2^0$

$= \mathbf{0}*2^6 + \mathbf{1}*2^5 + \mathbf{0}*2^4 + \mathbf{1}*2^3 + \mathbf{0}*2^3 + \mathbf{1}*2^1 + \mathbf{0}*2^0$

$= 2^5 + 2^3 + 2^1 = 32 + 8 + 2 = \mathbf{42}$

DEZIMAL NACH BINÄR

- Um eine Dezimalzahl in das Binärsystem umzuwandeln, muss man die enthaltenen Zweipotenzen finden
- Hierzu zieht man immer wieder die größte enthaltene Zweierpotenz ab und notiert diese
- Am Ende geht man alle existierenden Zweierpotenzen von der 1 an durch und notiert für jede gefundene eine 1 und für jede andere eine 0

DEZIMAL NACH BINÄR

- 54
- Größte Zweierpotenz in 54: **32**
- 54 – 32 = 22
- Größte Zweierpotenz in 22: **16**
- 22 – 16 = 6
- Größte Zweierpotenz in 6: **4**
- 6 – 4 = 2
- Größte Zweierpotenz in 2: **2**
- 2 – 2 = 0
- *Fertig*

64	**32**	**16**	8	**4**	**2**	1
0	1	1	0	1	1	0

OKTALSYSTEM

- Neben dem Binär (2) und dem Dezimalsystem (10) kann man auch beliebige andere Basen betrachten
- Im Oktalsystem rechnet man in Basis 8
- Dadurch sind die Ziffern 0 1 2 3 4 5 6 und 7 vorhanden

OKTALSYSTEM

3	**0**	**6**	**4**
$x8^3$	$x8^2$	$x8^1$	$x8^0$
256	64	8	1

OKTALSYSTEM

- Das Oktalsystem kann sehr einfach ins Binärsystem- und zurück umgewandelt werden
- Eine Ziffer im Oktalsystem hat den gleichen Wertebereich wie drei Ziffern im Binärsystem
- 0 – 7 ⇔ 000 – 111
- Deswegen kann man jede Oktalziffer einzeln in 3 Binärziffern umwandeln

OKTALSYSTEM

- Oktal 74 = 111 100
- Oktal 412 = 100 001 010
- Oktal 36123 = 011 110 001 010 011
- Binär 010101 = Oktal 25
- Binär 111000 = Oktal 70
- Binär 101110 = Oktal 56

Oktal	Binär
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

HEXADEZIMALSYSTEM

- Ein sehr beliebtes System zum Darstellen von Computerdaten ist das Hexadezimalsystem (Basis 16)
- Um die Basis 16 zu verwenden sind noch 6 weitere Ziffern nötig
- 0 1 2 3 4 5 6 7 8 9 A B C D E F
- Es wird gesetzt A = 10, B = 11 usw…
- Die Umwandlung funktioniert wie beim Oktalsystem, jedoch in Blöcken von vier
- Im Hexadezimalsystem kann man ein Byte (8 Bit/Binärstellen) in zwei Ziffern darstellen

HEXADEZIMALSYSTEM

- Hex A4 = Binär 1010 0100
- Hex FF = Binär 1111 1111
- Hex D00F = Binär 1101 0000 0000 1111
- Hex D1EB = Binär 1101 0001 1110 1011

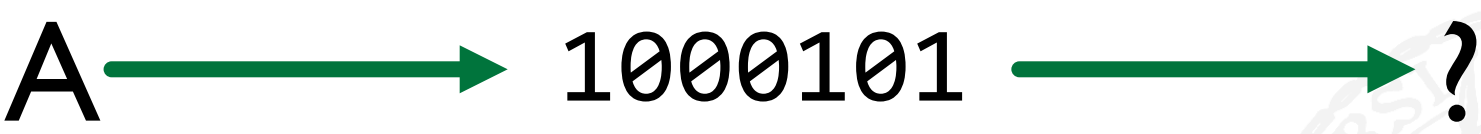
Hex	Binär
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A (10)	1010
B (11)	1011
C (12)	1100
D (13)	1101
E (14)	1110
F (15)	1111

SPLIT

- Mit string.split('x') kann man einen String in Einzelteile zerlegen
- Man übergibt ein Trennzeichen und erhält eine Liste aller Teile dazwischen zurück
- Das Trennzeichen verschwindet dabei
- ```
  "Hallo Welt".split(' ')
      ["Hallo", "Welt"]
  ```
- ```
  "Dra Chanasa mat da Kantrabass".split('a')
      ['Dr', ' Ch', 'n', 's', ' m', 't d', ' K', 'ntr', 'b', 'ss']
  ```

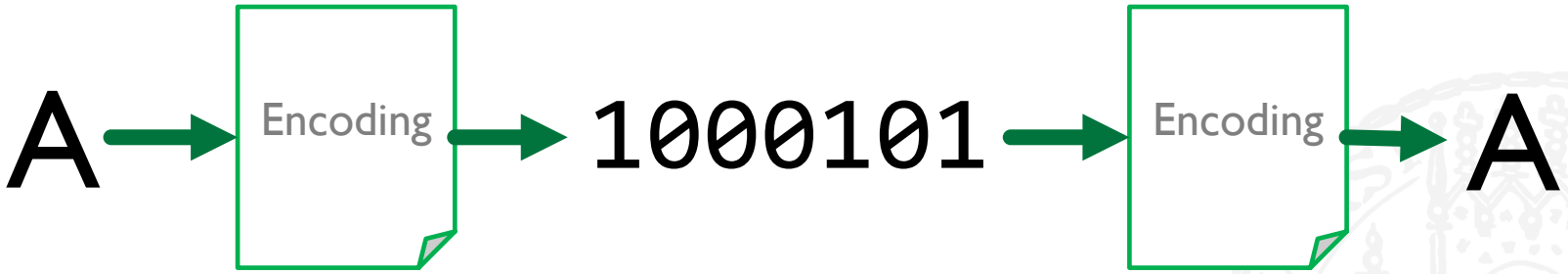
ENCODINGS: ASCII

- Computer können keine Buchstaben speichern, nur Zahlen
- Man braucht eine Vereinbarung welche Zahl zu welchem Buchstaben gehört



ENCODINGS: ASCII

- Computer können keine Buchstaben speichern, nur Zahlen
- Man braucht eine Vereinbarung welche Zahl zu welchem Buchstaben gehört



ENCODINGS: ASCII

- **ASCII**: 1963 u.a. für Fernschreiber entwickelt, 128 Zeichen auf 7 Bit
- `$ = 0100100`
- `A = 1000001`
- `z = 1111010`

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
1	1	SOH	17	11	DC1	33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
2	2	STX	18	12	DC2	34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
3	3	ETX	19	13	DC3	35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
4	4	EOT	20	14	DC4	36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
5	5	ENQ	21	15	NAK	37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
6	6	ACK	22	16	SYN	38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
7	7	BEL	23	17	ETB	39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
8	8	BS	24	18	CAN	40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
9	9	TAB	25	19	EM	41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
12	C	FF	28	1C	FS	44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	|
13	D	CR	29	1D	GS	45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
14	E	SO	30	1E	RS	46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
15	F	SI	31	1F	US	47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	

ENCODINGS: ISO 8859

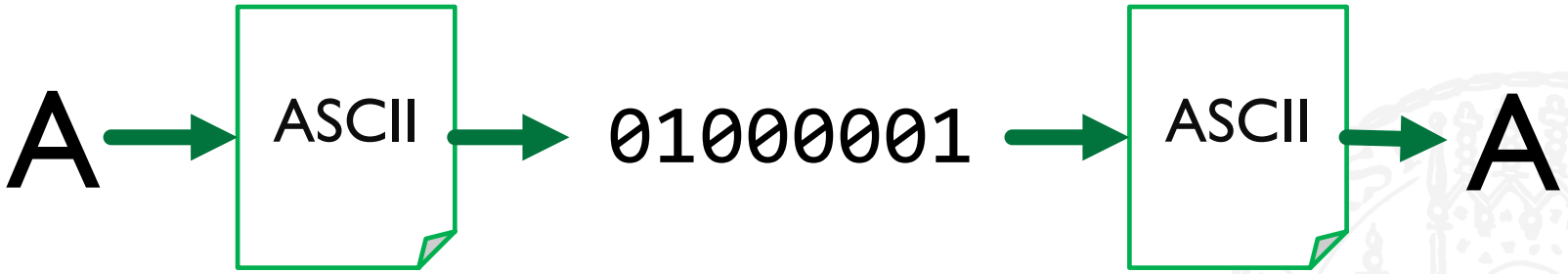
- ASCII enthält nur englische Buchstaben und Sonderzeichen
  - Was ist mit anderen Sprachen? äüÂøáË ĪKËÅõ
- Computer arbeiten mit 8-Bit ➔ Es sind noch 128 Möglichkeiten übrig
- $ = **0**0100100
- A = **0**1000001
- z = **0**1111010
- Ö = **1**??????
- å = **1**??????

ENCODINGS: ISO 8859

- ISO 8859 enthält 15 verschiedene Belegungen für die übrigen Plätze
  - ISO 8859-1 (Westeuropäisch)
  - ISO 8859-5 (Kyrillisch)
  - ISO 8859-11 (Thai)
- A = 01000001
- z = 01111010
- Ä = 11000100
- ü = 11111011

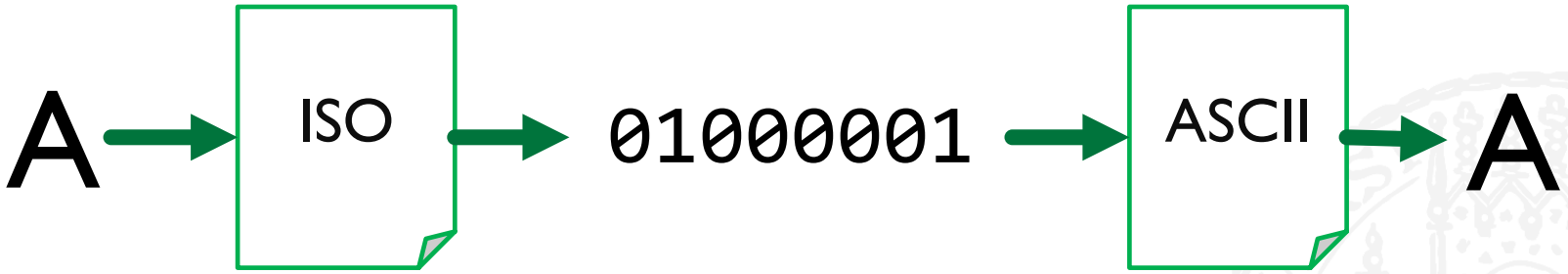
ENCODINGS: ISO 8859-1

- Durch das Ergänzen Funktioniert ISO „mit“ ASCII zusammen



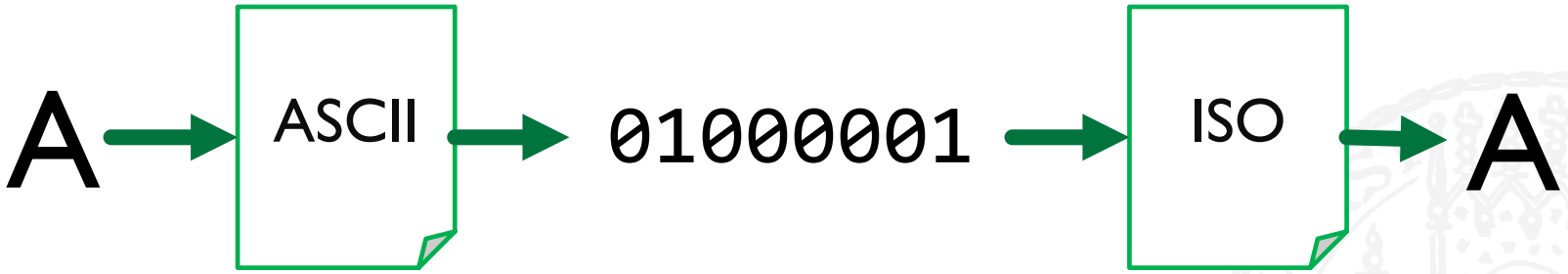
ENCODINGS: ISO 8859-1

- Durch das Ergänzen Funktioniert ISO „mit“ ASCII zusammen



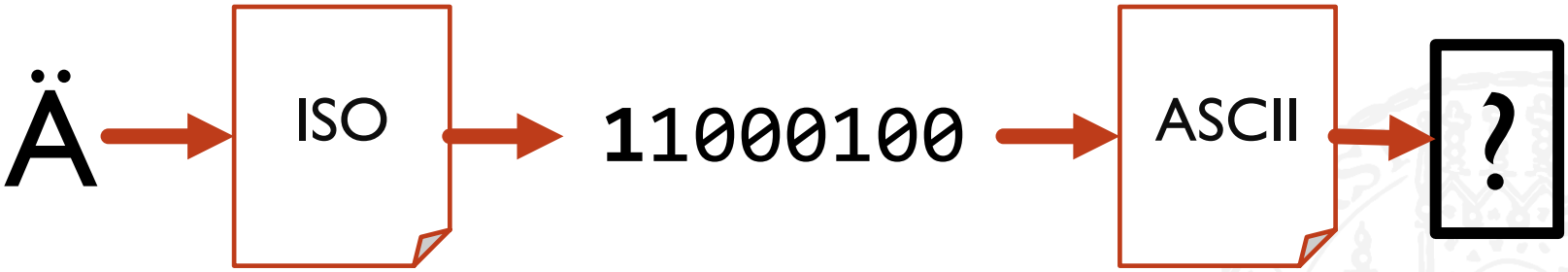
ENCODINGS: ISO 8859-1

- Durch das Ergänzen Funktioniert ISO „mit“ ASCII zusammen



ENCODINGS: ISO 8859-1

- Durch das Ergänzen Funktioniert (fast immer) ISO „mit“ ASCII zusammen



ENCODINGS: UNICODE

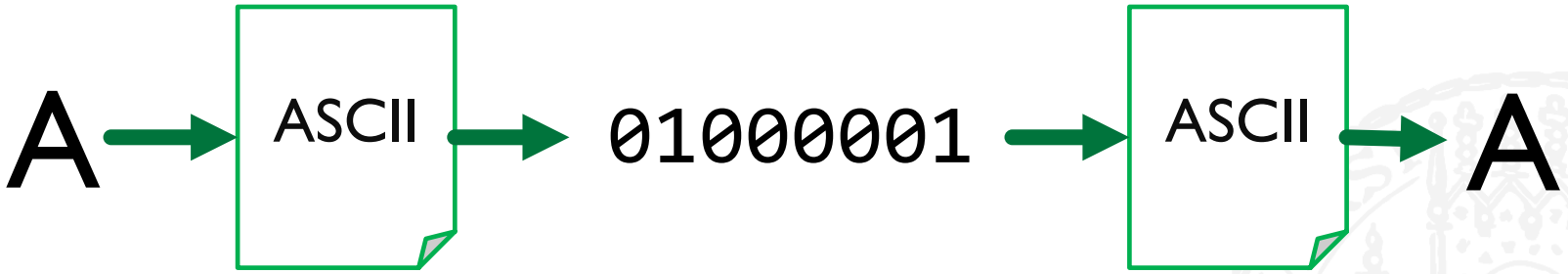
- ISO 8859 enthält jeweils nur 256 Zeichen
  - Was ist mit asiatischen Sprachen? ごみ 废话 🎄❤️🐧🦄💻
  - Was ist mit Dokumenten mit kyrillischen **und** deutschen „Sonderbuchstaben“?
- Es gibt mehr als $2^8 = 256$ Zeichen auf der Welt
- Es werden zwei Bit benötigt um alle Zeichen abzubilden
- In $2^{16} = 65.536$ ist genügend Platz für (fast) alle Zeichen

ENCODINGS: UTF 8

- Immer zwei Byte verwenden ist keine optimale Lösung
  - Platzverschwendung
  - Inkompatibel zu ASCII
  - Was ist wenn noch mehr Emojis erfunden werden...
  - ➢ Variable Länge
- Die ersten 127 Zeichen sind identisch zu ASCII und werden so gespeichert
  - **0**xxxxxxx = 00000000 0xxxxxxx
- Zeichen die mehr Platz benötigen werden in zwei/drei... Byte codiert
  - **110**xxxxx **10**xxxxxx = 00000xxx xxxxxxxx
  - **1110**xxxx **10**xxxxxx **10**xxxxxx = xxxxxxxx xxxxxxxx

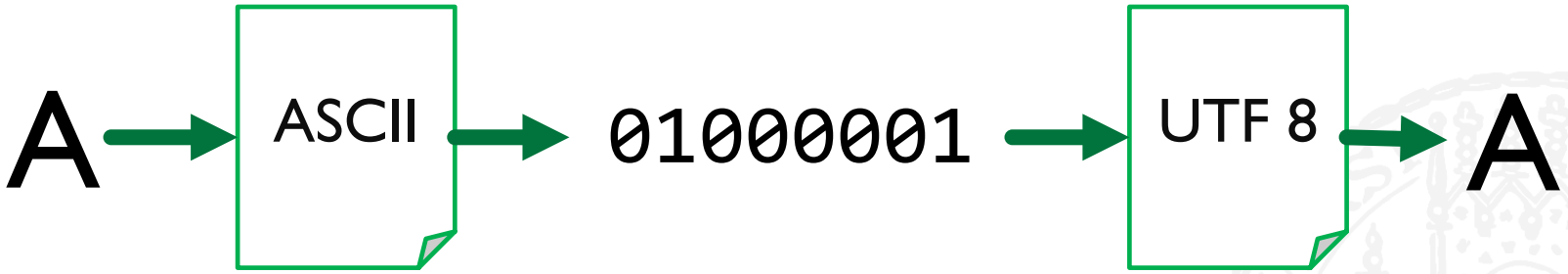
ENCODINGS: UTF 8

- Durch das Ergänzen Funktioniert UTF 8 „mit“ ASCII zusammen



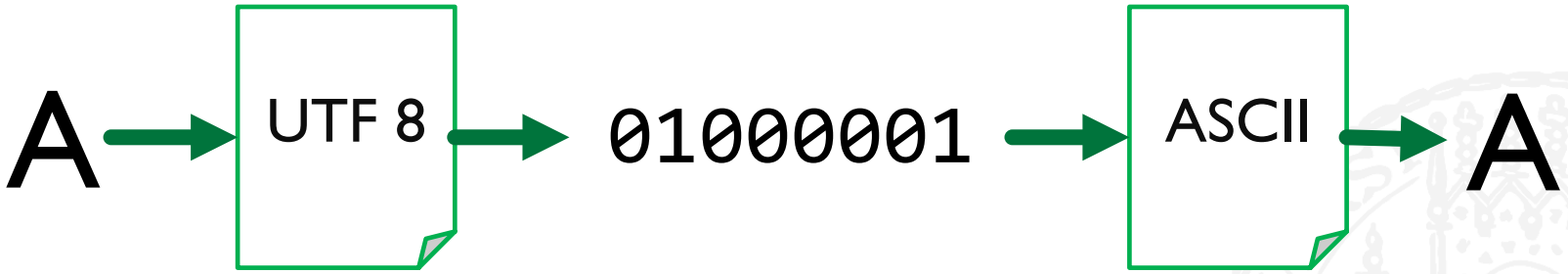
ENCODINGS: UTF 8

- Durch das Ergänzen Funktioniert UTF 8 „mit“ ASCII zusammen



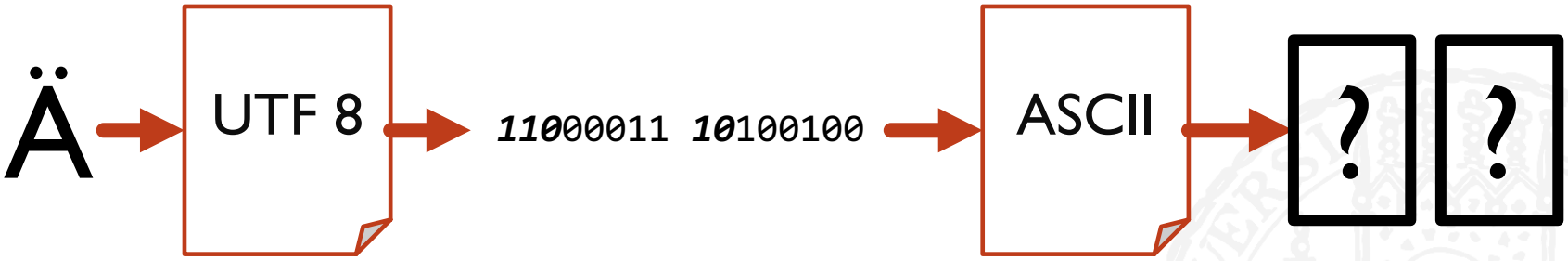
ENCODINGS: UTF 8

- Durch das Ergänzen Funktioniert UTF 8 „mit“ ASCII zusammen



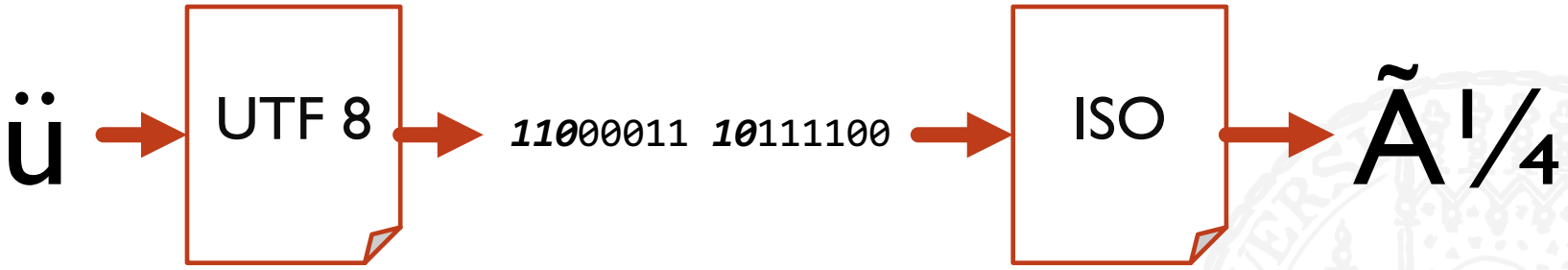
ENCODINGS: UTF 8

- Durch das Ergänzen Funktioniert UTF 8 „mit“ ASCII zusammen



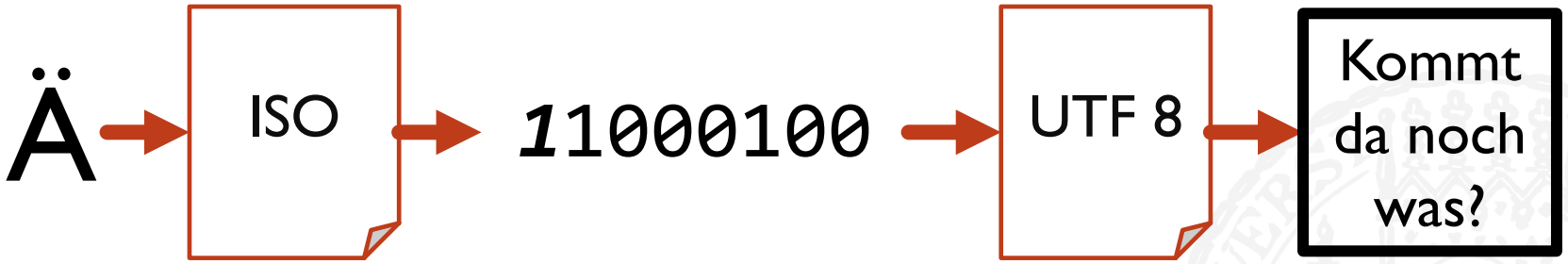
ENCODINGS: UTF 8

- Durch das Ergänzen Funktioniert UTF 8 **nicht** mit ISO Latin zusammen



ENCODINGS: UTF 8

- Durch das Ergänzen Funktioniert UTF 8 **nicht** mit ISO Latin zusammen



ENCODINGS: UTF-16

- UTF-16 belegt pauschal 2 Byte (16 Bit) pro Zeichen
- Inkompatibel zu allen anderen Encodings
- Programmierer sind sich bis heute nicht einig welches Byte zuerst kommt
- Es gibt deswegen zwei „Varianten“ von UTF-16:
  - **UTF-16 L**ittle**E**ndian (zuerst das “hintere“/“niederwertige“ Byte)
  - **UTF-16 B**ig**E**ndian (zuerst das „vordere“/“hochwertige“ Byte)
- Manchmal wird als erstes ein **B**yte**O**rder**M**ark gespeichert: `11111111 11111110` (LE)
- Sonst muss man raten, aber da die meisten Texte größtenteils aus englischen Buchstaben bestehen ist das hochwertige Byte sehr häufig 00000000

RECODE

- Man muss **häufig** Encodings umwandeln
- Hierzu gibt es das Tool **recode** `ENCODING..ENCODING DATEI`

```
Leonie@Laptop $ recode -l
ASCII-BS BS
ASMO_449 arabic7 iso-ir-89 ISO_9036
AtariST
baltic iso-ir-179
Bang-Bang
Leonie@Laptop $ recode UTF-8..ISO-8859-1 file.txt

Leonie@Laptop $ recode ISO-8859-1..ASCII-BS file.txt

Leonie@Laptop $ recode ASCII-BS..UTF-8 file.txt
```

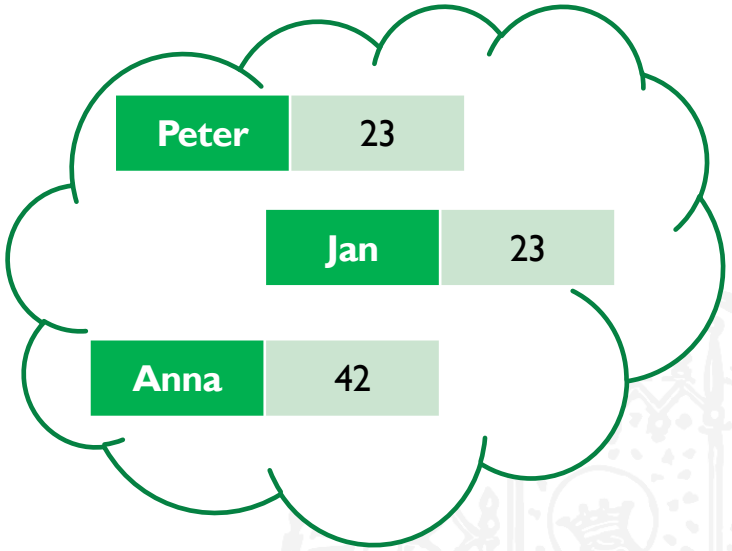
DICTIONARIES

- Beispiel: Alter von Personen speichern:
  - „Jan“ ➔ 23
  - „Anna“ ➔ 42
  - „Peter“ ➔ 23
- Lösung mit zwei Listen:
- ```
  namen = [„Jan“, „Anna“, „Peter“]
  alter = [27,    42,     23]
  ```
- Schwierig zu verwalten!
  - Dopplungen vermeiden
  - Werte finden (einmal durchsuchen?)


  ```

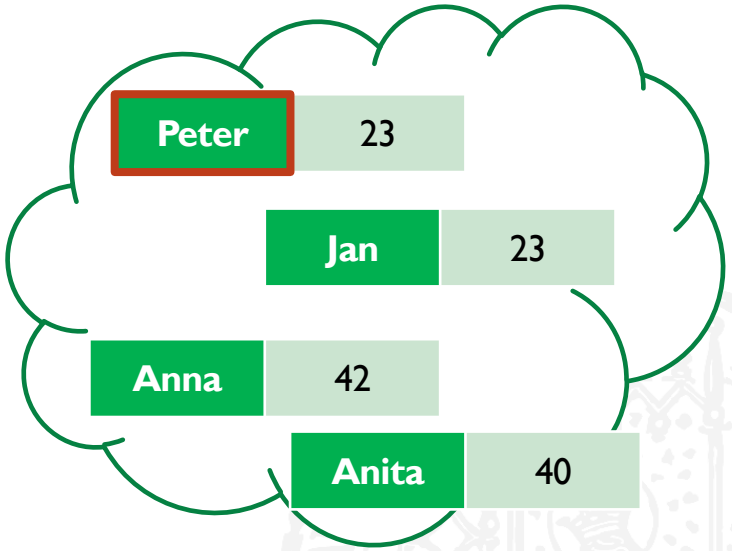
DICTIONARIES

- Es gibt in Python sog. **Dictionaries**
- In einem Dictionary kann man Werte unter **Schlüsseln** speichern
- Hier ist **Peter** ein Schlüssel und **23** der Wert
- Jeder Schlüssel der vorkommt hat genau einen Wert
- Jeder Wert kann beliebig oft vorkommen



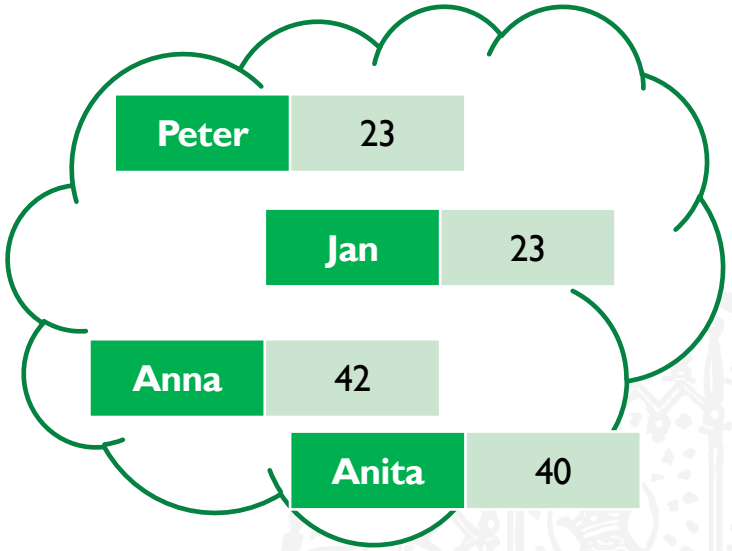
DICTIONARIES

- Man kann die Werte über die Schlüssel erreichen
- `>>> print( dict["Peter"] )`
  23
- `>>> dict["Anita"] = 40`



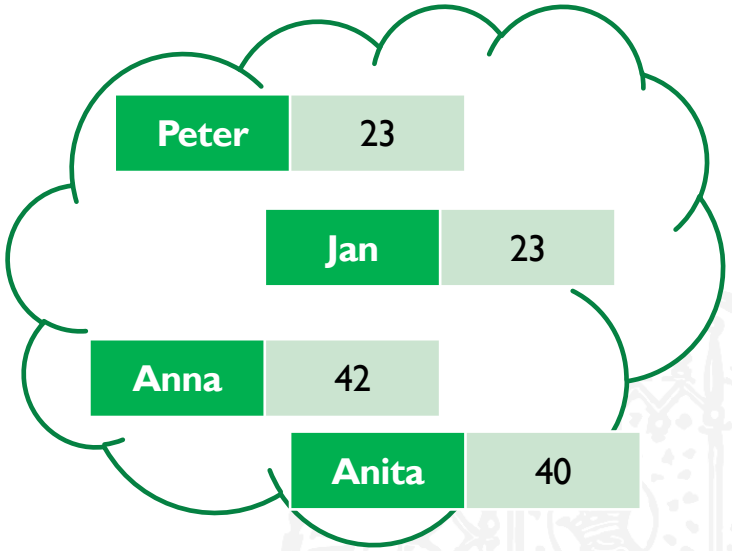
DICTIONARIES

- Dictionaries müssen wie Listen initialisiert werden
  - `dict = {}`
- Mit eckigen Klammern kann man einfügen/auslesen
  - `dict[„Peter“] = 23`
  - `print( dict[„Peter“] )`
- Mit `print` wird eine Textdarstellung ausgegeben
  - `{‘Peter‘: 23, ‘Jan‘: 23, ‘Anna‘: 42, ‘Anita‘: 40}`



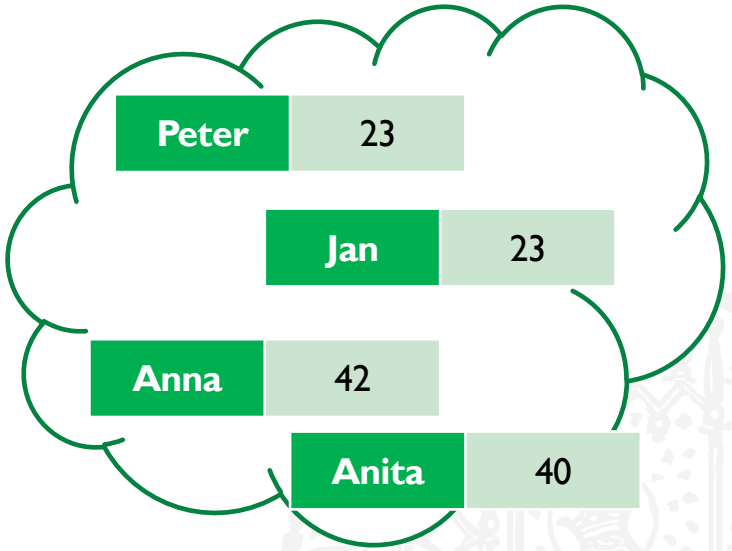
DICTIONARIES

- Mit einer for schleife kann man iterieren
- ```
  for key, value in dict.items():
      print(key, value)
  ```
- ```
  for key in dict.keys():
      print(key, dict[key])
  ```
- ```
  for value in dict.values():
      print(value)
  ```


  ```

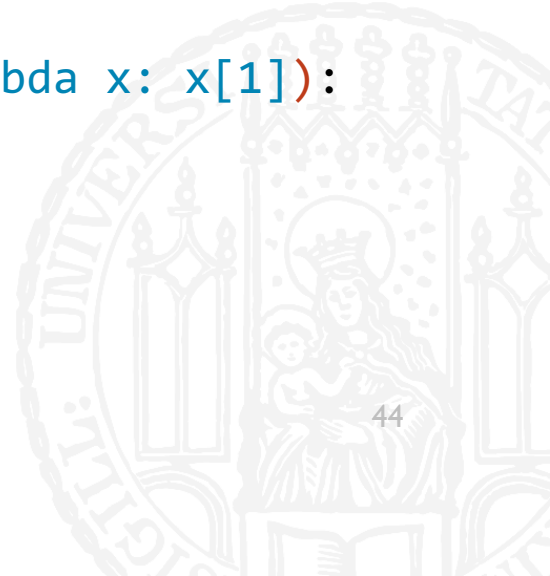
DICTIONARIES

- Dictionaries haben **keine Reihenfolge**
- Beim Ausgaben oder iterieren ist die Reihenfolge **zufällig**
- Man kann sich die Werte aber sortieren lassen



DICTIONARIES


- ```
  #nach Keys sortieren
  for key, value in sorted(dict.items()):
      print(key, value)
  ```
- ```
  #nach Value sortieren
  for key, value in sorted(dict.items(), key=lambda x: x[1]):
      print(key, value)
  ```



WILDCARDS

GZZ

- Mit einer sog. Wildcard kann man Dateien nach ihrem Namen auswählen
- ```
  ls
  abc.txt
  defgh.txt
  xyz.py
  ```
- ```
  ls *.txt
  abc.txt
  defgh.txt
  ```
- ```
  ls ???.*
  abc.txt
  xyz.txt
  ```


  ```

REGEXES

GZZ

- In Python kann man Text ähnlich wie mit Wildcards durchsuchen
- RegExes (RegularExpression) können noch mehr als Wildcards
- Dies sind die möglichen Zeichen
  - `.` (Punkt) = ein beliebiges Zeichen
  - a = ein kleines a
  - `[a-z]` = ein kleiner Buchstabe
  - `[A-Z]` = ein großer Buchstabe
  - `[äöüÄÖÜ]` = ein Umlaut
  - `[^z]` = ein beliebiges Zeichen das kein kleines z ist

REGEXES

GZZ

- Man kann für Zeichen auch eine Anzahl festlegen
  - `*` = beliebig viele (0-∞)
  - `?` = eins oder keins (0-1)
  - `+` = ein oder mehr (1-∞)
  - `{1,3}` = eins bis drei (1-3)
  - `{5,}` = fünf bis beliebig (5-∞)
  - `{,2}` = keine bis zwei (0-2)

REGEXES

GZZ

- Es gibt besondere Zeichen für bestimmte Zeichen
  - `^` = Anfang des Strings
  - `$` = Ende des Strings
  - `\w` = ein „Word Character“ (Buchstabe)
  - `\s` = ein „Whitespace Character“ (Leerzeichen, Tab, etc.)
  - `\S` = kein „Whitespace Character“ (alles außer Leerzeichen, Tab, etc.)
  - `\d` = ein „digit“ (Zahl)
  - `\\` = ein tatsächliches „\“
  - `\.` = ein tatsächlicher “.“

REGEXES

GZZ

- Um in Python eine Regex zu benutzen muss man sie zuerst erzeugen
- `meine_regex = re.compile(r'b+.*')`
- Man Schreibt ein **r** vor den String um einen raw-string zu erzeugen, um Backslashes verwenden zu können
- Danach kann man sie mit verschiedenen Funktionen anwenden
  - `re.match(meine_regex, "Ein beliebig blöder String")`
    Überprüft ob die Regex am Anfang des Strings matcht
  - `re.search(meine_regex, "Ein beliebig blöder String")`
    Überprüft ob die Regex irgendwo im String matcht
  - `re.findall(meine_regex, "Ein beliebig blöder String")`
    Gibt eine Liste aller Teile des Strings zurück die matchen (ohne Überlappung)

REGEX ALS SPLIT

GZZ

Dies ist ein Text. Er enthält, manche, so 12, besondere Zeichen	
`for word in text.split(' ')`	`split_regex = re.compile(r'\w+'),` `for word in re.findall(split_regex, text)`
Dies ist ein Text. Er enthält, manche, so 12, besondere Zeichen	**Dies ist ein Text. Er enthält, manche, so 12, besondere Zeichen**

ERGÄNZUNG: REVERSE SORT

GZZ

- Man kann beim Sortieren einen „Rückwärtsgang“ einlegen:
- ```
  #nach Keys sortieren
  for key, value in sorted(dict.items(), reverse=True):
      print(key, value)
  ```
- ```
  #nach Value sortieren
  for key, value in sorted(dict.items(), key=lambda x: x[1], reverse=True):
      print(key, value)
  ```

NUTZERRECHTE IN LINUX

G35

- In Linux kann jeder Nutzer verschiedenen Gruppen angehören
- Jede Datei hat einen User als Besitzer und eine Gruppe
- Der Zugriff auf die Datei wird in drei Ebenen kontrolliert:
  - Besitzer
  - Alle in der Gruppe
  - Alle
- Jede dieser Ebenen kann folgende Rechte haben:
  - Lesen (r)
  - Schreiben (w)
  - Ausführen (x)

NUTZERRECHTE IN LINUX

G35

```
rwxrwxrwx
rw-r-----
```

- Die Rechte einer Datei werden in einem langen ‘Wort’ dargestellt
- Jedes Trippel aus “rwx” steht für eine Ebene (Besitzer, Gruppe, Alle)
- In einer Ebene kann das r, w und x stehen, oder durch einen Strich ersetzt sein
- Buchstabe steht für “Recht vorhanden”, Strich für “nicht vorhanden”

NUTZERRECHTE IN LINUX

G35

```
rwxrwxrwx "alle dürfen alles"
rw-r----- "Nutzer: Lesen/Schreiben, Gruppe lesen"
```

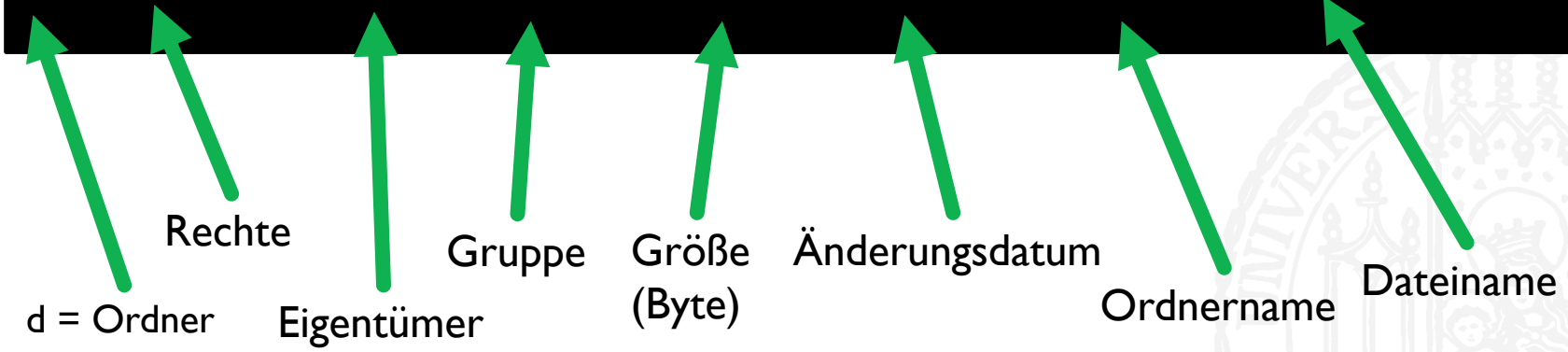
- Die Rechte einer Datei werden in einem langen 'Wort' dargestellt
- Jedes Trippel aus "rwx" steht für eine Ebene (Besitzer, Gruppe, Alle)
- In einer Ebene kann das r, w und x stehen, oder durch einen Strich ersetzt sein
- Buchstabe steht für "Recht vorhanden", Strich für "nicht vorhanden"

NUTZERRECHTE IN LINUX

G35

- Über den Command ls -l können diese Eigenschaften alle betrachtet werden

```
Leonie@Laptop $ ls -l
-rwxrw-r-- 1 Leonie EinfProg  191 Jan 12 23:14 programm.py
--w--w-rw- 1 Leonie Tutoren    28 Jun 02 16:33 schwarzes_brett.txt
drw-r----- 2 Leonie Tutoren     0 Dez 22 09:05 Website/
```



GREEDY / NON-GREEDY

G35

- Welchen Match eine Regex finden soll, ist nicht immer eindeutig
- Ein * oder + kann verschieden "weit" gehen
- `.*a` + `"Hey anna!"`
- `"Hey anna!"` oder `"Hey anna!"` ?
- Durch *Greedy* oder *Non-Greedy* kann der Unterschied spezifiziert werden!
- *Greedy* (englisch) **gierig**: wie gierig matcht der Regex?

GREEDY / NON-GREEDY

G35

- **Greedy**
  - Normales Verhalten
  - Matcht so weit wie **möglich**
  - `.*ein`
  - `Was für eine einsame Brücke?`
- **Non-Greedy**
  - Durch ein angehängtes Fragezeigen ausgelöst
  - Matcht nur so weit wie **nötig**
  - `.*?ein`
  - `Was für eine einsame Brücke?`

GROUPING

G35

- Manchmal sind die einzelnen Teile einer Regex interessant.
- `\d \w+`
- `Das sind die 8 Ritter!`
- Man kann sie mit Groups einschließen und danach auf diese zugreifen
- `(\d) (\w+)`
- `Das sind die 8 Ritter!`
- Es ist sogar möglich später in der Regex auf vorherige Gruppen zuzugreifen
- `(\d) (\w+) und \1 (\w+)`
- `Das sind die 8 Ritter und 8 Zauberer!`

FUNKTIONEN

G35

- Es ist möglich mehrfach verwendete Programmabschnitte zu gruppieren
- Man nennt diese Gruppen Funktionen und kann sie danach beliebig oft wieder aufrufen
- Eine Funktion wird mit dem Keyword **`def`**, einem Namen und `():` eingeleitet
- Danach kann sie durch `name()` beliebig aufgerufen werden
- ```
  def hallo_sagen():
      print('Hallo')

  hallo_sagen()
  ```
- **Faustregel:** anstatt Copy+Paste eine Funktion schreiben


  ```

FUNKTIONEN

G35

- Mit dem Keyword **return** kann eine Funktion auch etwas zurückliefern
- ```
  def fünf_fakultät():
      return 5*4*3*2*1

  ergebnis = fünf_fakultät()
  ```
- ```
  def aktuelles_jahr():
      return 2017

  jahr = aktuelles_jahr()
  ```

FUNKTIONEN

G35

- In den Klammern können die Funktionen Parameter erhalten
- In der Definition müssen hierzu der Reihe nach Namen vergeben werden
- Beim Aufrufen können dann entsprechend viele Parameter übergeben werden
- ```
  def vielfache_ausgeben(n):
      print(n, 2*n, 3*n, 4*n, 5*n)

  vielfache_ausgeben(2)
  >>> 2 4 6 8 10
  ```
- ```
  def multiplizieren(n,m):
      return n*m

  ergebnis = multiplizieren(5,6)
  print (ergebnis)
  >>> 30
  ```

FUNKTIONEN: ARGUMENTE

GY7

- Letztes Mal haben wir gesehen, dass man Funktionen Argumente übergeben kann:

```
def funktion(a,b):
    print(a+b)
```

- Nachteil: Die Funktion kann nur genau zwei Zahlen addieren
- Defaultwerte machen es möglich, einer Funktion unterschiedlich viele Argumente zu übergeben, weil für eventuell weggelassene Argumente ein Defaultwert hinterlegt ist:

```
def funktion(a,b,c=0,d=0):
    print(a+b+c+d)
```

IMMUTABLES VS MUTABLES

GY7

- Es gibt zwei Arten von Typen in Python: Mutables und Immutables
- Nur Variablen deren Typ mutable ist, können durch Methoden verändert werden
- Variablen deren Typ immutable ist, können nur neu belegt werden
- Zahlen sind immutable:
  - ```
    i = 3
    i = i + 4
    ```
- Listen sind mutable:
  - ```
    l = [0,7,2,5,1]
    l.sort()
    ```

IMMUTABLES VS MUTABLES

GY7

Immutable	Mutable
• int • float • string • boolean • range	• list • dict

CALL BY REFERENCE/ CALL BY VALUE GY7

- Es gibt zwei unterschiedliche Arten, wie ein Argument an eine Funktion übergeben werden kann:
- Call by reference
  - Das übergebene Objekt kann von der Funktion verändert werden
  - "Ich leihe dir meine Aufzeichnungen zum Abschreiben" → Du kannst auf mein Blatt schreiben
- Call by value
  - Es wird eine Kopie übergeben
  - "Ich kopiere dir meine Aufzeichnungen zum Abschreiben" → Meine Blatt bleibt wie es ist

CALL BY REFERENCE/ CALL BY VALUE GY7

- Man kann sich Call by Reference / Call by Value **nicht** aussuchen
- Immutables werden mit Call by Value aufgerufen
  - Weil sie nicht geändert werden können
- Mutables werden mit Call by Reference aufgerufen
  - Weil sie geändert werden können
  - Wenn man bei Mutables das Call by Reference umgehen will, kann man mit `list[:]` eine Kopie der Liste erstellen und mit `dict(frequenzliste)` eine Kopie des dictionarys

LOKALE UND GLOBALE VARIABLEN GY7

- Problem:
- ```
  x = 5
  def funktion():
      x += 1
  ```
- `UnboundLocalError: local variable 'x' referenced before assignment`


  ```

LOKALE UND GLOBALE VARIABLEN GY7

- Ein sog. *Namespace* ist der Bereich, in dem eine Variable existiert
- Es gibt dabei den sog. Globalen Namespace in dem "normale" Variablen existieren
- Eine Funktion hat aber ihren eigenen Namespace
  - Sie hat erstmal keinen Zugriff auf Variablen aus dem globalen Space
  - Sie kann nur durch ihre Argumente und "return" kommunizieren
- Mit `global x` baut man sich eine Verbindung zum globalen namespace auf
- **In 99% der Fälle braucht man keine solche Verbindung**

GLOBAL

GY7

```
x = 5
def funktion():
    x += 1
UnboundLocalError: local variable 'x' referenced before assignment

def richtige_funktion():
    global x
    x += 1

richtige_funktion()
x = 6

richtige_funktion()
x = 7
```


LAMBDA

GY7

- Lambda, diesesmal vernünftig erklärt:
- Ein Lambda-Ausdrucke definiert eine anonyme Funktion
- Beispiel:
- `for wort, frequenz in sorted(dict.items(),key=lambda x:x[1]):`
- Übersetzt:

```
def anonym(x):
    return x[1]
```

- Wird erstellt und an key übergeben

REKURSION

GVA

- Eine Funktion kann sich auch selbst aufrufen!
- z.B. Fakultät berechnen
- `5! = 5*4*3*2*1`
- `10! = 10*9*8*7*6*5*4*3*2*1`
- oder:
- `5! = 5 * 4!`
- `10! = 10 * 9!`
- Eine Rekusion braucht auch ein defininiertes Ende
- `1! = 1`

REKURSION

GVA

- Eine Rekursion lässt sich in zwei Fälle teilen:
  - Rekursion
  - Abbruch
- Ohne Abbruch würde die Rekursion ewig laufen
- Rekursion ist ideal um ein Problem in einfachere Teilprobleme zu zerlegen bis der einfachste Fall eintritt

REKURSION

GVA

- 4! =
- 4 * 3! =
- 4 * 3 * 2! =
- 4 * 3 * 2 * 1! =
- 4 * 3 * 2 * 1 =
- 24

REKURSION

GVA

```
def fakultät(n):
    if n == 1:
        return 1
    else:
        return n * fakultät(n-1)
```

REKURSION

GVA

```
fakultät(4) =
if 4 == 1:
    return 1
else:
    return 4 * fakultät(3)
```

REKURSION

GVA

```
fakultät(4) =
if 4 == 1:
    return 1
else:
    return 4 * if 3 == 1:
                   return 1
               else:
                  return 3 * fakultät(2)
```

REKURSION

GVA

```
fakultät(4) =
if 4 == 1:
    return 1
else:
    return 4 * if 3 == 1:
                   return 1
               else:
                  return 3 * if 2 == 1:
                                 return 1
                             else:
                                return 2 * fakultät(1)
```

REKURSION

GVA

```
fakultät(4) =
if 4 == 1:
    return 1
else:
    return 4 * if 3 == 1:
                   return 1
               else:
                  return 3 * if 2 == 1:
                                 return 1
                             else:
                                return 2 * if 1 == 1:
                                               return 1
                                           else:
                                              return 1 * fakultät
```

REKURSION

GVA

```
fakultät(4) =
if 4 == 1:
    return 1
else:
    return 4 * if 3 == 1:
                   return 1
               else:
                  return 3 * if 2 == 1:
                                 return 1
                             else:
                                return 2 * 1
```

REKURSION

GVA

```
fakultät(4) =
if 4 == 1:
    return 1
else:
    return 4 * if 3 == 1:
                   return 1
               else:
                  return 3 * 2
```

REKURSION

GVA

```
fakultät(4) =
if 4 == 1:
    return 1
else:
    return 4 * 6
```

REKURSION

GVA

```
fakultät(4) = 24
```