13. ÜBUNG

ZUR EINFÜHRUNG IN DIE PROGRAMMIERUNG FÜR COMPUTELRINGUISTEN

Leonie Weißweiler

TWEEDBACK

65Y

- lmu.twbk.de
- Lesson ID: **65Y**

WIEDERHOLUNG

65Y

- Was gibt der Code aus?

```
def magic(t):
    if t >= 16:
        return 1
    else:
        return magic(t*2) * 2
print( magic(4) )
```

a) 1

b) 2

c) 4

d) 8

WIEDERHOLUNG

65Y

- Was gibt der Code aus?

```
def magic(t):
    if t >= 16:
        return 1
    else:
        return magic(t*2) * 2
print( magic(4) )
```

a) 1

b) 2

c) 4

d) 8

GROSSE WIEDERHOLUNG!

UNIX - PFADE

65Y

- 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`

PIPING VS REDIRECTING

65Y

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

BINÄRSYSTEM

65Y

0 1 0 1 0 1 0

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

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

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

DEZIMAL NACH BINÄR

65Y

- 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

65Y

- 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

HEXADEZIMALSYSTEM

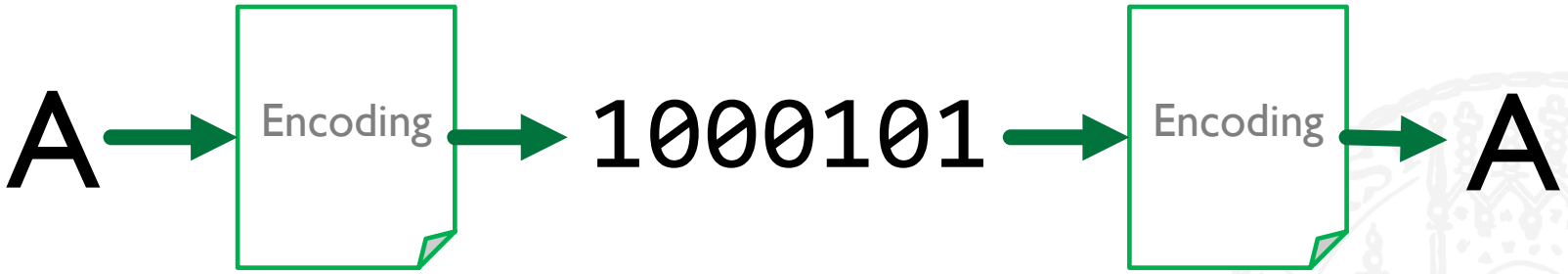
65Y

- 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

ENCODINGS: ASCII

65Y

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



ENCODINGS: ASCII

65Y

- **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

65Y

- 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: UNICODE

65Y

- 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

65Y

- 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-16

65Y

- 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

NUTZERRECHTE IN LINUX

65Y

```
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"

DATENTYPEN

65Y

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

TYPE CASTING

65Y

- 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'
  ```

LISTEN

65Y

```
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

65Y

```
len(zahlen)
    3

print(zahlen[-1])
    90

zahl = 3

zahl.append(4)
    Fehler
```

0	1	2
3	-4	90

3

3

SLICING

65Y

- 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

WHILE-SCHLEIFE

65Y

- 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
  ```

FOR SCHLEIFEN

65Y

- ```
  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“*
  ```

RANGES

65Y

- 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
  ```
  ```

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]`

FILEHANDLES

65Y

- 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')
```

Filehandle — Dateiname — Modus, hier *read* = nur lesen

FILEHANDLES

65Y

- 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')`

SPLIT

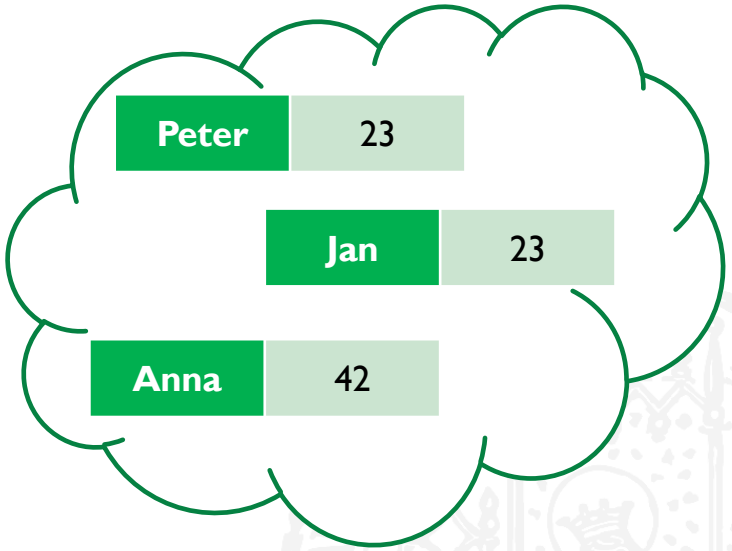
65Y

- 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']
  ```

DICTIONARIES

65Y

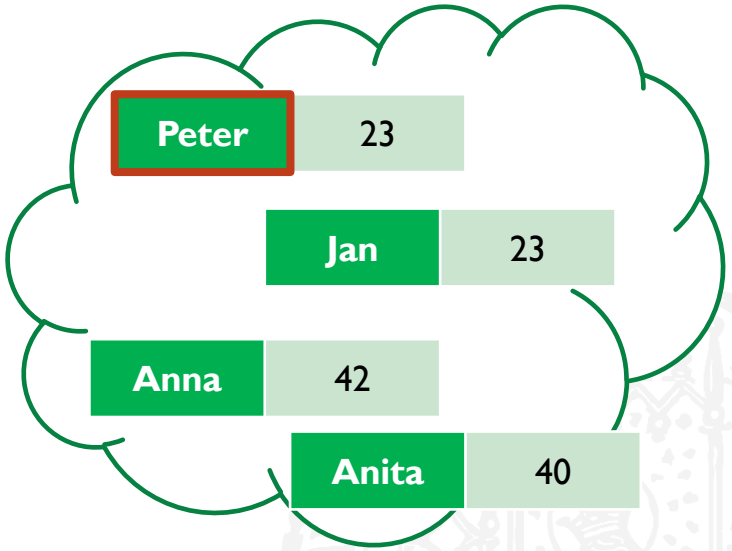
- Es gibt in Python sog. **Dictionaries**
- In einem Dictionary kann man Werte unter **Schlüsseln** speichern
- Hier ist **Peter** ein Schlüssel und **23** der Wert
- Jeder Schlüssel der vorkommt hat genau einen Wert
- Jeder Wert kann beliebig oft vorkommen



DICTIONARIES

65Y

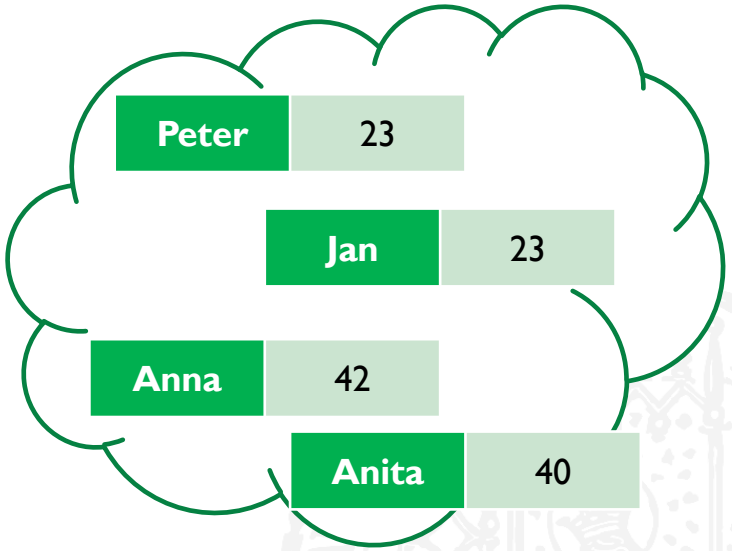
- Man kann die Werte über die Schlüssel erreichen
- `>>> print( dict["Peter"] )`
  23
- `>>> dict["Anita"] = 40`



DICTIONARIES

65Y

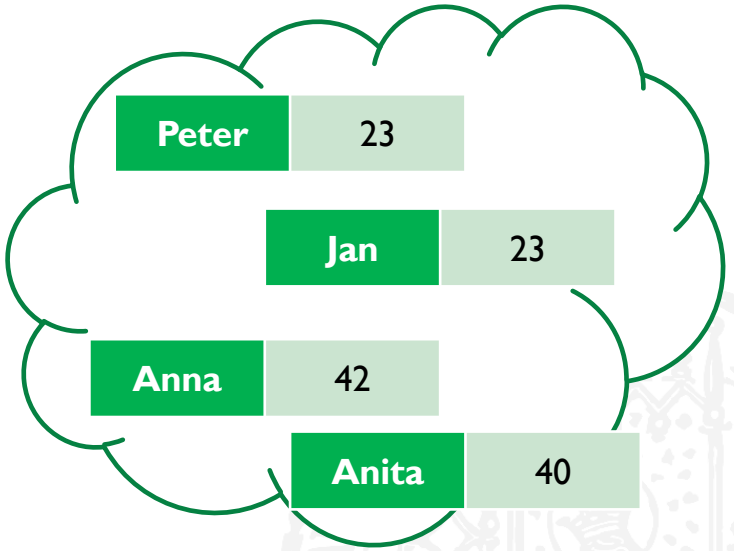
- Dictionaries müssen wie Listen initialisiert werden
  - `dict = {}`
- Mit eckigen Klammern kann man einfügen/auslesen
  - `dict[„Peter“] = 23`
  - `print( dict[„Peter“] )`
- Mit `print` wird eine Textdarstellung ausgegeben
  - `{‘Peter‘: 23, ‘Jan‘: 23, ‘Anna‘: 42, ‘Anita‘: 40}`



DICTIONARIES

65Y

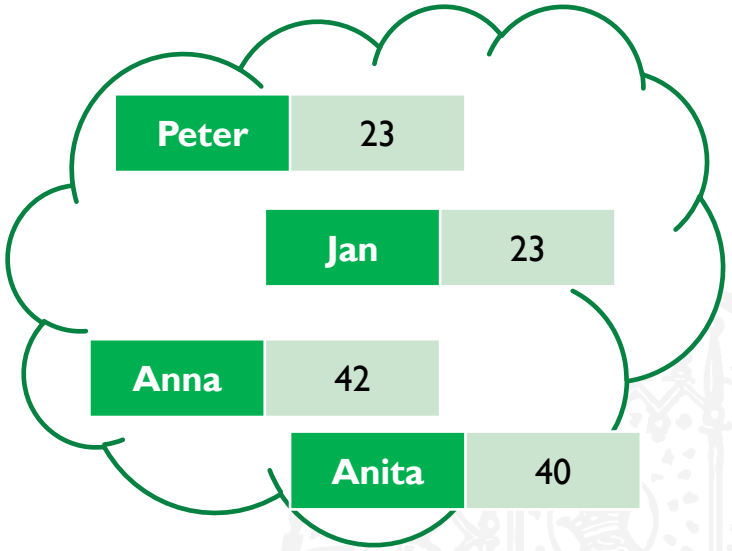
- Mit einer for schleife kann man iterieren
- ```
  for key, value in dict.items():
      print(key, value)
  ```
- ```
  for key in dict.keys():
      print(key, dict[key])
  ```
- ```
  for value in dict.values():
      print(value)
  ```


  ```

DICTIONARIES

65Y

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



REGEXES

65Y

- 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

65Y

- 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

65Y

- 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)

GREEDY / NON-GREEDY

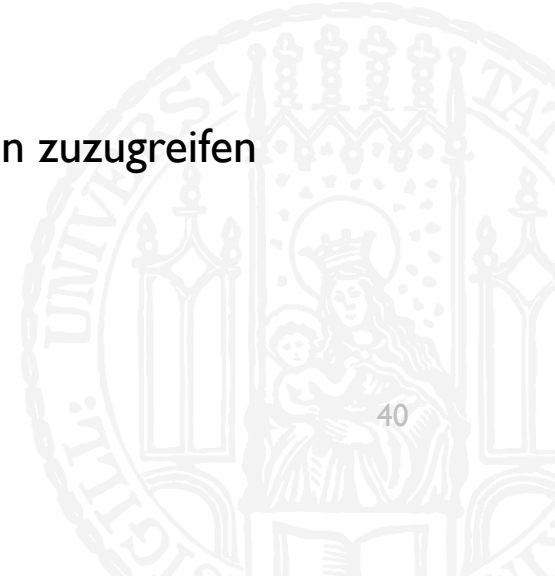
65Y

- **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

65Y

- 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

65Y

- 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

65Y

- 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

65Y

- 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
  ```

IMMUTABLES VS MUTABLES

65Y

- 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

65Y

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

CALL BY REFERENCE/ CALL BY VALUE 65Y

- 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 65Y

- 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

MUSTERLÖSUNGEN

MUSTERLÖSUNG 13-1

65Y

- Schreiben Sie eine Funktion, die eine Frequenzliste aller Buchstabenfolgen der Länge 2 entwickelt.

```
def länge2(text):
    twodictionary = {}

    for i in range(0,len(text) - 2):
        two = text[i:i+2]
        if (two in twodictionary):
            twodictionary[two] = twodictionary[two] + 1
        else:
            twodictionary[two] = 1

    return twodictionary
```

MUSTERLÖSUNG 13-2

65Y

- Schreiben Sie eine Funktion, die eine Frequenzliste aller Buchstabenfolgen der Länge 3 entwickelt.

```
def länge2(text):
    threedictionary = {}

    for i in range(0,len(text) - 3):
        three = text[i:i+3]
        if (two in twodictionary):
            threedictionary[three] = threedictionary[three] + 1
        else:
            threedictionary[three] = 1

    return threedictionary
```

MUSTERLÖSUNG 13-3

65Y

- Schreiben Sie eine Funktion, die ein dictionary übergeben bekommt und die die 10 häufigsten Werte des dictionary ausdruckt.

```
def zehnhäufigste(dictionary):
    i = 0
    for wort, frequenz in sorted(dictionary.items(),
                                 key=lambda x: x[1],
                                 reverse=True):
        if (i < 10):
            print (wort)
            i = i+1
```

MUSTERLÖSUNG 13-4

65Y

- Schreiben Sie ein Hauptprogramm, das die beiden Dateien einliest und jeweils für beide Sprachen die häufigsten 10 Buchstabenfolgen der Länge 2 und der Länge 3 ausgibt. Betrachten Sie das Ergebnis.

```
de = open('de.txt', 'r')
en = open('en.txt', 'r')

de_text = de.read()
en_text = en.read()

print ('Deutsch Länge 2:')
zehnhäufigste(länge2(de_text))
print ('Deutsch Länge 3:')
zehnhäufigste(länge3(de_text))

print ('Englisch Länge 2:')
zehnhäufigste(länge2(en_text))
print ('Englisch Länge 3:')
zehnhäufigste(länge3(en_text))
```