

Einführung in Python

4. Vorlesung





Wiederholung letztes Mal

- Rekursive Funktionen
(und Execution Frames)

```
def fakultät(n):  
    if n <= 1:                                #Elementarfall  
        return 1  
    else:  
        return n*fakultät(n-1)                #rekursiver Aufruf
```

- Lokale Funktionen
(für closures oder factory Funktionen)

```
def äussere_funktion():  
    def innere_funktion():                    #lokale Funktion  
        print(x)  
    x = 2  
    innere_funktion()
```

- Lambda Funktionen

```
(lambda x,y: x+y) (5,14)                      #Lambdaausdruck  
19  
l.sort(key= lambda x: x[1])
```

- Generatoren
(und Iteratoren)

```
c = (i*i for i in range(10))                  #Generatorausdruck  
def generiere_zahlen(n):                      #Generatorfunktion  
    for i in range(n):  
        yield i*i
```



Wiederholung letztes Mal

- Datei Ein-/Ausgabe über das *File*-Objekt

```
>>> input = open('/home/python/tolles_infile.txt', 'r')
>>> output = open('/home/python/tolles_outfile.txt', 'w')
>>> input.close()
>>> output.close()
```

- Zuverlässigkeit mit der *with*-Anweisung

```
>>> with open('/home/python/wichtige_daten.dat') as f:
    daten = f.readlines()
    [...]
```

- Speichern von Daten mittels *pickle*

```
>>> pickle.dump(wichtigeListe, open('/home/documents/il.pydmp', 'wb'))
>>> listeReloaded = pickle.load(open('/home/documents/il.pydmp', 'rb'))
```



Wiederholung letztes Mal

- Manipulation des Python-Interpreters zur Laufzeit mittels *sys*
- Zugriff auf das Betriebssystem mittels *os*
- Interaktion mit der Systemuhr mittels *time*



Verarbeitung von Zeichenketten

- Standardmethoden zur Verarbeitung von Zeichenketten
- Codierung und Decodierung
- Automatische Textproduktion
- Reguläre Ausdrücke

STRATEGIES FOR FULL-WIDTH JUSTIFICATION

THEIR THINDOOPHILERS
ON THE RELATIONSHIP
BETWEEN
DEINDUSTRIALIZATION
AND THE CROUTLE OF

GIVING UP

THEIR THINDOOPHILERS
ON THE RELATIONSHIP
B E T W E E N
DEINDUSTRIALIZATION
AND THE CROUTLE OF

LETTER
SPACING

THEIR THINDOOPHILERS
ON THE RELATIONSHIP
BETWEEN DEINDUS -
TRIALIZATION AND THE
CROUTLE OF ECOLOGICAL


HYPHENATION

THEIR THINDOOPHILERS
ON THE RELATIONSHIP
BETWEEN
DEINDUSTRIALIZATION
AND THE CROUTLE OF

STRETCHING

THEIR THINDOOPHILERS
ON THE RELATIONSHIP
BETWEEN CRAP LIKE
DEINDUSTRIALIZATION
AND THE CROUTLE OF

FILLER

THEIR THINDOOPHILERS
ON THE RELATIONSHIP
BETWEEN 
DEINDUSTRIALIZATION
AND THE CROUTLE OF

SNAKES



Verarbeitung von Zeichenketten

- Die Klasse ***str*** bietet eine Reihe von Methoden zur Analyse oder zur Produktion neuer Zeichenketten
- Beachtet: Beim Aufruf einer Methode auf ein Stringobjekt ***s*** (***s.methode(arg1, ...)***) wird das Objekt ***s*** selbst niemals verändert
- Es entsteht jedes Mal ein neues Stringobjekt, welches einer durch den Funktionsaufruf veränderten Kopie von ***s*** entspricht



Zeichenketten – Standardmethoden

- Formatieren & Schreibweise

Methode	Erklärung
<code>center(w, [char])</code>	String wird links und rechts mit Leerzeichen (oder <i>char</i>) aufgefüllt, bis eine Gesamtlänge von <i>w</i> erreicht wurde.
<code>ljust(w, [char])</code>	String wird mit angehängten Leerzeichen (oder <i>char</i>) auf Länge <i>w</i> erweitert.
<code>rjust(w, [char])</code>	String wird mit vorangestellten Leerzeichen (oder <i>char</i>) auf Länge <i>w</i> erweitert.
<code>capitalize()</code>	String beginnt mit einer Großbuchstaben und es folgen nur noch kleine.
<code>lower()</code>	Wandelt alle Buchstaben in kleine Buchstaben um.
<code>upper()</code>	Wandelt alle Buchstaben in große Buchstaben um.

- Tests

<code>endswith(suffix)</code>	Liefert TRUE, falls String mit der Zeichenkette <i>suffix</i> endet.
<code>isalnum()</code>	Liefert TRUE, falls String nicht leer ist und alle Zeichen alphanummerisch sind.
<code>isalpha()</code>	Liefert TRUE, falls alle Zeichen im String Buchstaben sind.
<code>isdigit()</code>	Liefert TRUE, falls Zeichen im String Zahlen sind.
<code>islower()</code>	Liefert TRUE, falls alle Buchstaben kleingeschrieben sind.
<code>isupper()</code>	Liefert TRUE, falls alle Buchstaben großgeschrieben sind.



Zeichenketten – Standardmethoden

```
>>> 'mitte'.center(10, '#')  
'##mitte###'                                #center() fängt immer Rechts an aufzufüllen  
  
>>> 'rechts'.rjust(10)  
'      rechts'                             #String nach Rechts verschieben  
  
>>> 'links'.ljust(10)  
'links '                                  #String nach Links verschieben  
  
>>> 'Der Python'.lower()  
'der python'  
  
>>> 'Die Python'.upper()  
'DIE PYTHON'  
  
>>> 'Das Python'.capitalize()  
'Das python'
```



Zeichenketten – Standardmethoden

```
>>> 'Was ist die Höchstgeschwindigkeit einer unbeladenen Schwalbe?'.endswith('?')
True

>>> 'Hm, eine europäische oder eine afrikanische Schwalbe?'.endswith('Schwalbe?')
True

>>> 'Keine Ahnung. Ahhhh.....'.isalpha()      #Sonderzeichen sind keine Buchstaben
False

>>> '19'.isdigit()
True

>>> '932 A.D.'.isalnum()      #Sonderzeichen sind keine alphanummerischen Zeichen
False

>>> 'eines tages, junge, wird das alles dir gehören'.islower()
True

>>> 'DIE OLLEN GARDINEN?'.isupper()  #Nur Buchstaben könne klein-/großgeschrieben sein
True
```



Zeichenketten – Standardmethoden

- Entfernen & Aufspalten

Methode	Erklärung
<code>lstrip([chars])</code>	Führende Buchstaben aus <i>chars</i> werden entfernt. Bei fehlendem Argument werden Whitespaces entfernt.
<code>rstrip([chars])</code>	Buchstaben aus <i>chars</i> am Ende des Strings werden entfernt. Bei fehlendem Argument werden Whitespaces entfernt.
<code>strip([chars])</code>	Am Anfang und Ende des Strings werden Buchstaben aus <i>chars</i> oder bei fehlendem Argument Whitespaces entfernt.
<code>split([sep])</code>	Der String wird in eine Liste von Teilstrings aufgeteilt. Dabei wird <i>sep</i> als Trennsymbol verwendet, wenn es fehlt wird nach Whitespaces getrennt.
<code>splitlines()</code>	Der String wird in eine Liste von Zeilen aufgespalten.

- Suchen & Ersetzen

<code>count(sub)</code>	Liefert Anzahl der Vorkommen der Zeichenkette <i>sub</i> .
<code>find(sub)</code>	Gibt den Index ersten Position wieder, an dem <i>sub</i> im String gefunden wurde, -1 sonst.
<code>replace(old, new)</code>	Vorkommen der Zeichenkette <i>old</i> werden durch <i>new</i> ersetzt.



Zeichenketten – Standardmethoden

```
>>> '   Leere   '.lstrip()
'Leere'
>>> '   Leere   '.rstrip()
'   Leere'
>>> '   Leere   '.strip(' eL')
'r '
>>> '   LEere   '.strip(' eL')
'Eer'
>>> t = 'Das ist nur ein Kratzer.\nEin Kratzer!? Ihr Arm ist ab!\nNein, das stimmt nicht.'

>>> t.splitlines()
['Das ist nur ein Kratzer.', 'Ein Kratzer!? Ihr Arm ist ab!', 'Nein, das stimmt nicht.']

>>> t.split()
['Das', 'ist', 'nur', 'ein', 'Kratzer.', 'Ein', 'Kratzer!?', 'Ihr', 'Arm', 'ist', 'ab!', 'Nein,', 'das', 'stimmt', 'nicht.']

>>> t.split('Kratzer')
['Das ist nur ein ', '.\nEin ', '!? Ihr Arm ist ab!\nNein, das stimmt nicht.']
```



Zeichenketten – Standardmethoden

Name	Vorname	Alter	Rolle
Chapman	Graham	48	König Arthur, Wächter, Stimme Gottes
Cleese	John	66	Sir Lancelot, Tim der Zauberer, Schwarzer Ritter
Gilliam	Terry	65	Knappe Patsy, Sir Bors, Grüner Ritter
Idle	Eric	63	Sir Robin, Diener Concord, Bruder Maynard
Jones	Terry	65	Sir Bedevere, Prinz Herbert, Landarbeiterin
Palin	Michael	64	Sir Galahad, Dennis, Herr des Sumpfschlosses

```
>>> with open('/home/dokumente/wichtige_daten.csv') as f:
    for line in f:
        print(line.split('\t'))

['Name', 'Vorname', 'Alter', 'Rolle\n']
['Chapman', 'Graham', '48', 'König Arthur, wächter, Stimme Gottes\n']
['Cleese', 'John', '66', 'Sir Lancelot, Tim der Zauberer, Schwarzer Ritter\n']
['Gilliam', 'Terry', '65', 'Knappe Patsy, Sir Bors, Grüner Ritter\n']
['Idle', 'Eric', '63', 'Sir Robin, Diener Concord, Bruder Maynard\n']
['Jones', 'Terry', '65', 'Sir Bedevere, Prinz Herbert, Landarbeiterin\n']
['Palin', 'Michael', '64', 'Sir Galahad, Dennis, Herr des Sumpfschlosses\n']
```



Zeichenketten – Standardmethoden

Name	Vorname	Alter	Rolle
Chapman	Graham	48	König Arthur,Wächter,Stimme Gottes
Cleese	John	66	Sir Lancelot,Tim der Zauberer,Schwarzer Ritter
Gilliam	Terry	65	Knappe Patsy,Sir Bors,Grüner Ritter
Idle	Eric	63	Sir Robin,Diener Concord,Bruder Maynard
Jones	Terry	65	Sir Bedevere,Prinz Herbert,Landarbeiterin
Palin	Michael	64	Sir Galahad,Dennis,Herr des Sumpfschlosses

```
>>> with open('/home/dokumente/wichtige_daten.csv') as f:
    e = f.readlines()

# Man hätte auch folgendes schreiben können:
# e = open('/home/dokumente/wichtige_daten.csv').readlines()

>>> e[1].split('\t')
['Chapman', 'Graham', '48', 'König Arthur,Wächter,Stimme Gottes\n']

>>> e[1].split('\t')[-1].split(',')          #Stringmethoden können verkettet werden
['König Arthur', 'Wächter', 'Stimme Gottes\n']
```



Zeichenketten – Standardmethoden

```
>>> t = 'Wir sind die Ritter die immer Ni sagen! Ni ni ni ni ni!'
>>> t.count('Ni')
2

>>> t.count('ni')                #Case-Sensitivität beachten
4

>>> t.find('Ni')                 #Es wird immer nur die erste Position des Vorkommens zurückgegeben
30

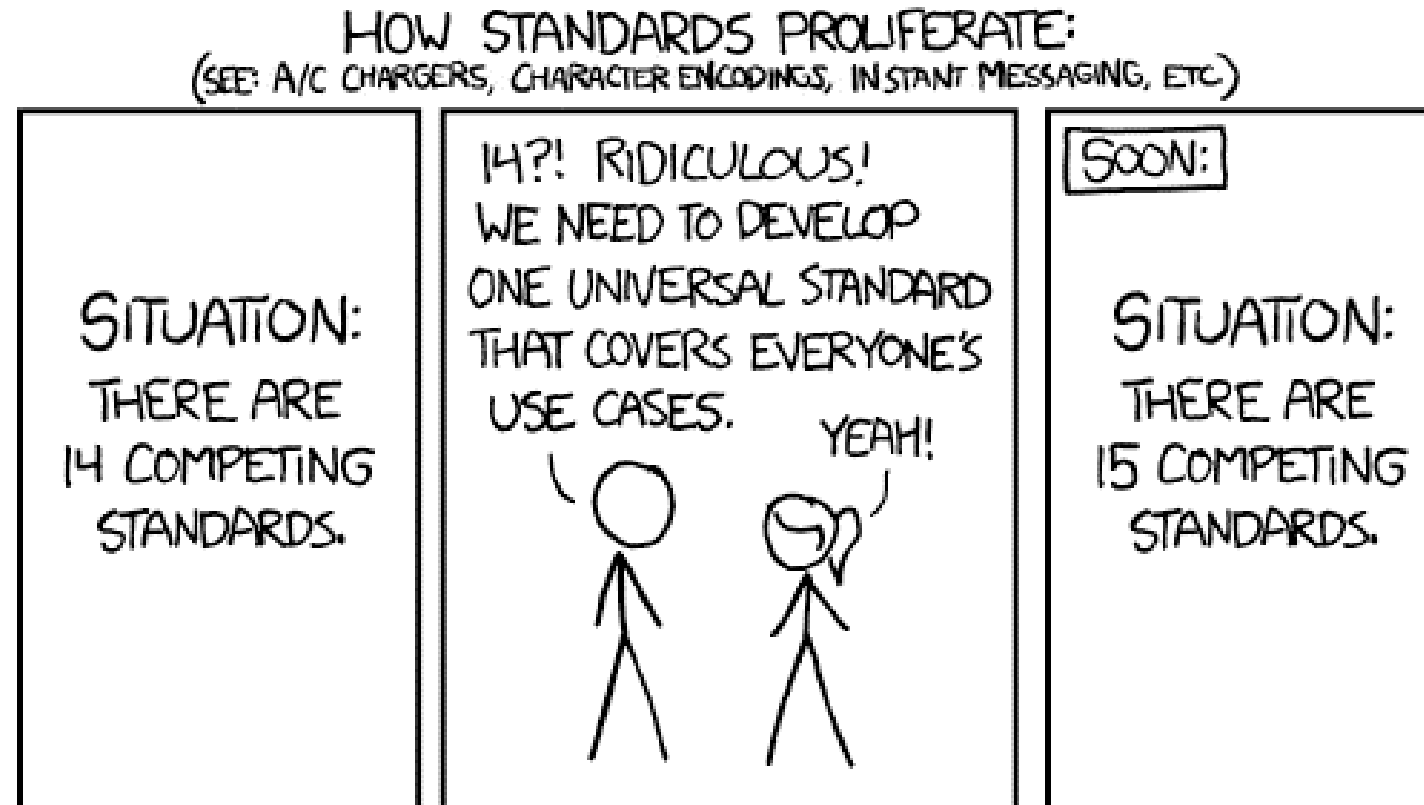
>>> 'wir sagen auch nie ni!'.replace('ni','hecki-hecki-pateng')
'wir sagen auch hecki-hecki-patenge hecki-hecki-pateng!'

#Jedes Vorkommen des zu Ersetzenden Strings wird auch ausgetauscht

>>> 'wir sagen auch nie ni!'.replace(' ','')
'wirsagenauchnieni!'
```



Codierung und Decodierung



Codierung und Decodierung

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]



Codierung und Decodierung

- Unicode ist der internationale Codierungsstandard für Schriftzeichen (ISO-10646)
- Er umfasst über hunderttausend *platonische* Schriftzeichen, denen allen eine Nummer und ein eindeutiger Name zugewiesen wird
- Es existieren einige Unicode-Formate (z. B. UTF-8, UTF-16, UTF-32), diese lassen sich verlustfrei ineinander konvertieren
- Die ersten 128 Zeichen sind identisch mit der ASCII-Codierung



WATCHING THE UNICODE PEOPLE TRY TO GOVERN THE INFINITE CHAOS OF HUMAN LANGUAGE WITH CONSISTENT TECHNICAL STANDARDS IS LIKE WATCHING HIGHWAY ENGINEERS TRY TO STEER A RIVER USING TRAFFIC SIGNS.



Codierung und Decodierung

- Die Funktion **ord(c)** liefert zu einem einzelnen Schriftzeichen *c* seine Unicode-Nummer
- Die Funktion **chr(n)** liefert zu einer Unicode-Nummer das zugehörige Schriftzeichen

```
>>> ord('ß')  
223  
  
>>> chr(223)  
ß  
  
>>> for i in range(1044,1055):  
        print(chr(i), end=' ')
```

Д Е Ж З И Й К Л М О



Codierung und Decodierung

- Die Funktion **ord(c)** liefert zu einem einzelnen Schriftzeichen *c* seine Unicode-Nummer
- Die Funktion **chr(n)** liefert zu einer Unicode-Nummer das zugehörige Schriftzeichen

```
>>> import unicodedata                                #Enthält alle Unicode-Namen/-Nummern/-Schriftzeichen

>>> unicodedata.name('ß')
'LATIN SMALL LETTER SHARP S'

>>> unicodedata.lookup('KATAKANA LETTER GA')
'ガ'

>>> unicodedata.lookup('HIRAGANA LETTER GA')
'が'
```



Codierung und Decodierung

- Die Standardcodierung des Pythoninterpreters ist UTF-8
- Mit den String-Methoden **encode()** und **decode()** können Stringobjekte in Byteobjekte umgewandelt werden und vice versa

```
>>> wort = 'Ägäis'

>>> wortBytes = wort.encode('utf-8')

>>> wortBytes
b'\xc3\x84g\xc3\xa4is'                                #Bildschirmdarstellung des Bytestrings

>>> wortBytes.decode()                                #Ohne Argument wird auch UTF-8 (de)codiert
'Ägäis'
```



Codierung und Decodierung

- UTF-8 codiert alle Zeichen in 1 – 4 Oktette (8 Bit)

platonisches Zeichen $\xrightleftharpoons[\text{Decodierung}]{\text{Codierung}}$ Oktette (Bytestring)

Ägäis $\xrightleftharpoons[\text{UTF-8 Decodierung}]{\text{UTF-8 Codierung}}$

Ä	g	ä	i	s		
195	132	103	195	164	105	115
1100	1000	0110	1100	1010	0110	0111
0011	0100	0111	0011	0100	1001	0011

Ägäis $\xrightleftharpoons[\text{Latin-1 Decodierung}]{\text{Latin-1 Codierung}}$

Ä	g	ä	i	s
196	103	228	105	115



Codierung und Decodierung

- UTF-16 codiert alle Zeichen in 2 oder 4 Oktette (8 Bit)

platonisches Zeichen $\xrightleftharpoons[\text{Decodierung}]{\text{Codierung}}$ Oktette (Bytestring)

Ägäis $\xrightleftharpoons[\text{UTF-8 Decodierung}]{\text{UTF-8 Codierung}}$

Ä	g	ä	i	s
195	132	103	195	164
105	115			
1100 1000 0110 1100	1010 0110 0111 0011			
0011 0100 0111 0011	0100 1001 0011			

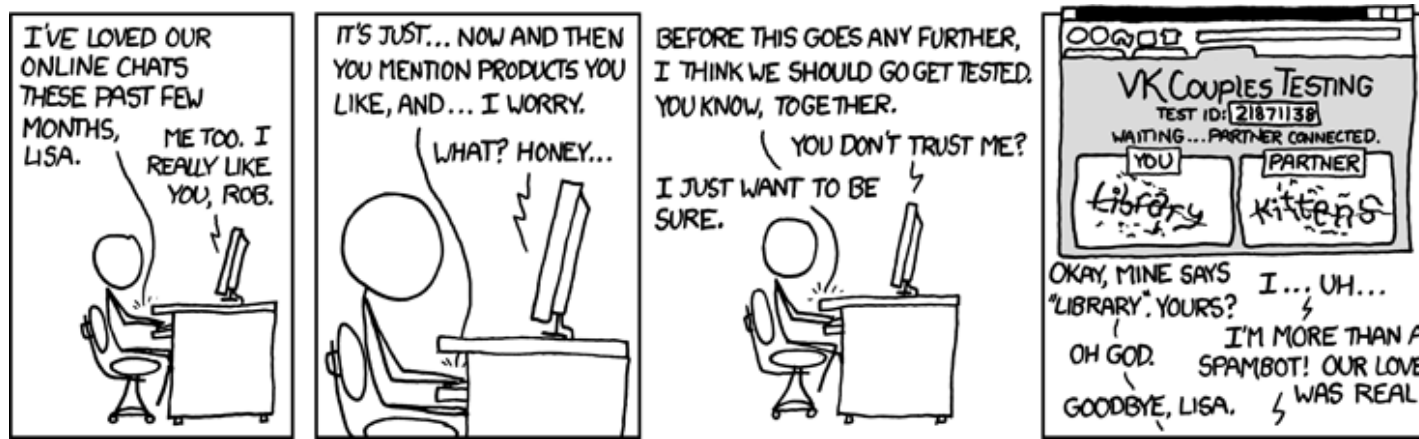
Ägäis $\xrightleftharpoons[\text{UTF-16 Decodierung}]{\text{UTF-16 Codierung}}$

Ä	g	ä	i	s
255 254 196 0	103 0	228 0	105 0	115 0



Automatische Textproduktion

- Es gibt zwei Möglichkeiten Texte mit variablen Teilen zu definieren:
direktes *in-place casting* oder indirekt durch setzen von Platzhaltern
- direktes *Casting* haben wir schon in der ersten VL kennengelernt
- Platzhalter lassen sich mit der String-Methode **format()** realisieren



Automatische Textproduktion

- Platzhalter werden mit geschweiften Klammern {} in Strings gesetzt

Basistext

Platzhalter

Daten

```
>>> 'Zur Kreuzigung {} rum. Jeder nur {} Kreuz!'.format('links',1)
'Zur Kreuzigung links rum. Jeder nur 1 Kreuz!'           #Platzhalter-Klammern verschwinden

>>> 'Zur Kreuzigung {} rum. Jeder nur {} Kreuz!'.format('links',1, 2)
'Zur Kreuzigung links rum. Jeder nur 1 Kreuz!'           #Überschüssige Daten werden ignoriert

>>> 'Zur Kreuzigung {} rum. Jeder nur {} Kreuz!'.format('links')
IndexError: tuple index out of range                     #Fehler bei fehlenden Daten

>>> 'Setz dich. Nimm dir ein Keks, mach es dir bequem... du {}{}!'.format('Arsch')
'Setz dich. Nimm dir ein Keks, mach es dir bequem... du {Arsch}!'
#Doppelte Platzhalter-Klammern um einfache geschweifte Klammern anzuzeigen
```



Automatische Textproduktion

- Platzhalter können nummeriert **{x}** oder benannt **{name}** werden um sie gezielt anzusprechen

```
>>> getränk = 'Tee'
>>> 'Das ist ein Virus. Einfach {1} Tage ruhig halten, {0} trinken und beim
fußballspielen das andere Bein belasten.'.format(getränk, 2.5)

'Das ist ein Virus. Einfach 2.5 Tage ruhig halten, Tee trinken und beim fußballspielen
das andere Bein belasten.'

>>> 'Nun, dann wächst es also wieder nach?'

>>> 'Nein, das ist kein {k} und Ihr Bein wird auch nicht wieder nachwachsen. Das sieht
mir eher wie ein {b} aus.'.format(k='Virus', b='Tigerbiss')

'Nein, das ist kein Virus und Ihr Bein wird auch nicht wieder nachwachsen. Das sieht
mir eher wie ein Tigerbiss aus.'

>>> 'Was? Ein Tiger, hier in Afrika?!'
```



Automatische Textproduktion

- Für Platzhalter lassen sich Formatspezifikatoren definieren, welche festlegen wie die Daten zu formatieren sind

{:Ausrichtung Mindestweite .Präzision Zahlentyp}

Spezifikator	Bedeutung
Ausrichtung	Beschreibt die Textbündigkeit: < linksbündig > Rechtsbündig ^ zentriert
Mindest-Feldweite	Anzahl der Stellen die mindestens für die Variable reserviert werden.
Präzision	Gibt die Genauigkeit von Floats hinter dem Komma.
Zahlentyp	Beschreibt wie eine Zahlenvariable zu interpretieren ist: d: ganze Dezimalzahl f: Gleitkommazahl x: Hexadezimal b: Binärzahl



Automatische Textproduktion

- Für Platzhalter lassen sich Formatspezifikatoren definieren, welche festlegen wie die Daten zu formatieren sind

{:Ausrichtung Mindestweite .Präzision Zahlentyp}

```
>>> tabelle = ''
>>> for i in range(1,6):
    tabelle += '{a:>10d}{b:^10.2f}{c:<10x}\n'.format(a=i, b=i**2, c=i**3)
>>> print(tabelle)
```

1	1.00	1
2	4.00	8
3	9.00	1b
4	16.00	40
5	25.00	7d



Automatische Textproduktion

- Seit Python 3.6 gibt es eine weitere (bessere) Methode um Formatstrings zu benutzen

```
>>> a = 'Poden'
>>> b = 'Purschen'

>>> f'werft den {b} zu {a}!'
'werft den Purschen zu Poden!'

>>> f'Ergebnis: {5.193669119149:{^}{10}.{2}{f}}'      #Formatspezifikatoren
'Ergebnis:      5.19      '

>>> f'Ergebnis: {5.193669119149:^10.4}'      #Formatspezifikatoren in kurz
'Ergebnis:      5.19      '

>>> zahl = '5.193669119149'
>>> f'Ergebnis: {zahl:^10.4}'
```



Reguläre Ausdrücke

WHENEVER I LEARN A
NEW SKILL I CONCOCT
ELABORATE FANTASY
SCENARIOS WHERE IT
LETS ME SAVE THE DAY.

OH NO! THE KILLER
MUST HAVE FOLLOWED
HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH
THROUGH 200 MB OF EMAILS LOOKING FOR
SOMETHING FORMATTED LIKE AN ADDRESS!



IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR
EXPRESSIONS.



Reguläre Ausdrücke

- Reguläre Ausdrücke sind Zeichenketten, die zur Analyse von Texten verwendet werden
- Durch einen regulären Ausdruck *r* kann man eine Menge von Strings definieren, die zu *r* passen
 - Filtern von Strings nach bestimmten Mustern
- Einfachster Fall: Regulärer Ausdruck besteht aus einer Zeichenfolge, die in jedem passendem String vorkommen muss



re = 'an'

Brian



Danke



Anke



Römer



Reguläre Ausdrücke

- Mit Hilfe von speziellen Zeichensequenzen und Sonderzeichen kann man komplexe Muster definieren
- Im wesentlichen sind reguläre Ausdrücke aus folgenden Elementen aufgebaut:
 - Einfache Zeichen, die für sich selbst stehen (z. B. 'e')
 - Platzhalter für Zeichen aus einer Zeichenmenge (z. B. '\d' für eine Ziffer)
 - Operatoren, mit denen aus einem oder mehreren regulären Ausdrücken ein neuer regulärer Ausdruck gebildet wird (z. B. '+' für ein- oder mehrfach Wiederholung eines Ausdrucks)



Reguläre Ausdrücke

Sonderzeichen	Bedeutung / Beispiel
.	Jedes Zeichen außer Zeilenwechsel ('\n'). 'G.ld' passt auf 'Gold', 'Geld' oder 'Gilde'
^	Beginn eines Strings oder erstes Zeichen nach \n '^S' passt auf 'Start' nicht aber auf 'der Start'
[]	Definition einer Menge von Zeichen. Zum Beispiel bezeichnet '[abc]' ein Zeichen aus der Menge {a, b, c}.
[^]	Komplement einer Zeichenmenge. [^aeiouAEIOU] steht für ein beliebiges Zeichen das kein Vokal ist
*	Beliebig häufiges (eventuell keinmaliges) Wiederholen des vorausgehenden regulären Ausdrucks.
+	Ein- oder mehrmaliges Wiederholen des vorausgehenden regulären Ausdrucks.
?	Null- oder einmaliges Auftreten des vorausgehenden regulären Ausdrucks.
\	Maskieren eines Sonderzeichens.
\A	Beginn einer Zeichenkette. '\AAuto' passt auf 'Autobahn', nicht aber auf 'Sie fuhr ein grünes\nAuto'



Reguläre Ausdrücke

Sonderzeichen	Bedeutung / Beispiel
\d	Dezimalziffer, entspricht der Menge [0-9].
\D	Alle Zeichen außer Dezimalziffern.
\s	Alle Whitespace-Zeichen aus der Menge [\n\t\r\f\v].
\S	Alle Zeichen außer Whitespace-Zeichen.
\w	Irgendein alphanummerisches Zeichen aus [a-zA-Z0-9_].
\W	Irgendein nichtalphanummerisches Zeichen aus [^a-zA-Z0-9].
\Z	Ende einer Zeichenkette.
	oder-Verknüpfung regulärer Ausdrücke.
()	Zeichengruppe, ermöglicht gruppieren verschiedener Ausdrücke.



Reguläre Ausdrücke - Beispiele

`'\d\d\d'`

passt auf alle dreistelligen Dezimalzahlen

`'0[0-7]*'`

passt auf alle Oktalzahlen in der Python-Syntax

`'Mu+h'`

passt auf Muh, Muuh, Muuuh, usw. aber nicht auf Mh

`'www\.\w+\.(de|gb|fr)'`

alle deutschen, britischen, französischen
Internetseiten deren Namen aus mindestens einem
alphanumerischen Zeichen besteht

`'(Bankleitzahl|BLZ):?\s*\d+'`

passt auf 'Bankleitzahl 0123456789'
oder auf 'Konto BLZ:\t9876543210'

`'0\d+[-/)]?\s*[1-9]\d\d+'`

passt auf 'Tel.: (08527) 56683' oder '08527-
56683' oder 'Telefon 08527/56683'

`'[1-9]\d+([,.] (\d\d|-))?'`

passt auf '536.69' oder '149,-' oder '19'



Reguläre Ausdrücke – Das Modul *re*

- In Python verarbeitet man reguläre Ausdrücke mit Hilfe des Moduls *re*
- Mit der Funktion **compile(pattern [, flag])** wird ein RE-Objekt zu einem bestimmten Ausdruck erzeugt, welches weitere nützliche Funktionen zur Analyse von Strings enthält

```
>>> from re import *           #Import aller Funktionen aus dem Modul re
>>> regex = compile('\d+')      #Erzeugen eines RE-Objekts namens 'regex'

>>> print(regex)                #print gibt den re-Funktionsaufruf wieder
re.compile('\d+')

>>> type(regex)
<class '_sre.SRE_Pattern'>
```



Funktionen eines RE-Objekts

Methode	Erklärung
<code>findall(string)</code>	Liefert eine Liste von nicht überlappenden Teilstrings, auf die der reguläre Ausdruck passt.
<code>match(string)</code>	Wenn der reguläre Ausdruck auf ein <i>Anfangsstück</i> des Strings <i>string</i> passt, wird ein Matchobjekt zurückgegeben, sonst None.
<code>search(string)</code>	Liefert ein Matchobjekt, falls der reguläre Ausdruck gefunden wird, sonst None.
<code>split(string [, maxnum])</code>	Liefert zu <i>string</i> eine Liste von maximal <i>maxnum</i> Teilstrings. Durch den regulären Ausdruck wird der Trennstring definiert.
<code>sub(replace, string)</code>	Liefert einen String, der aus <i>string</i> wie folgt ermittelt wird: Teilstücke, auf die der reguläre Ausdruck passt, werden durch <i>replace</i> ersetzt.



Funktionen eines RE-Objekts

- Analyse von Strings mit **match()** und **search()**

```
>>> from re import *
>>> regex = compile('\A(\w\w\w\s)')
>>> if regex.search('Der Sinn des Lebens'):           #search durchsucht den ganzen String
    print('Ausdruck gefunden')
Ausdruck gefunden

>>> regex = compile('\A(\w\w\w\w\s)')
>>> if regex.match('Der Sinn des Lebens'):           #match untersucht nur den Anfang
    print('Ausdruck gefunden')
else:
    print('Ausdruck nicht gefunden')
Ausdruck nicht gefunden

>>> regex = compile('\A(\w\w\w\w\s)')
>>> if regex.match('Sinn des Lebens'):
    print('Ausdruck gefunden')
Ausdruck gefunden
```



Funktionen eines RE-Objekts

- Substrings extrahieren mit **findall()**

We have a comprehensive overview of the changes in the "What's New in Python 3.5" document, found at

<http://docs.python.org/3.5/whatsnew/3.5.html>

For a more detailed change log, read Misc/NEWS (though this file, too.
Documentation for Python 3.5 is online, updated daily:

<http://docs.python.org/3.5/documentation/3.5.html>

```
>>> regex = compile('http://.+html?', 'I')
>>> with open('/home/Readme.txt') as e:
    text = e.read()
    liste = regex.findall(text)
    for x in liste:
        print(x)
```

<http://docs.python.org/3.5/whatsnew/3.5.html>

<http://docs.python.org/3.5/documentation/3.5.html>



Funktionen eines RE-Objekts

- Die Wiederholungsoperatoren * und + bewirken eine Suche nach einem passenden Teilstring *maximaler* Länge

```
>>> regex = compile('<.*>')  
>>> re.findall('<h1> Python </h1>')  
['<h1> Python </h1>']
```

- Um die *kürzesten* Teilstrings zu finden schreibt man *? oder +?

```
>>> regex = compile('<.*?>')  
>>> re.findall('<h1> Python </h1>')  
['<h1>', '</h1>']
```



Funktionen eines RE-Objekts

- Strings zerlegen mit **split()**

```
>>> regex = compile('[.,:;!]?')
>>> text = 'Dieser Papagei ist nicht mehr! Er hat aufgehört zu existieren. Er hat sein
Leben beschlossen und hat die Augen zugemacht. Er ist eine Leiche, des Lebens beraubt,
er ruht in Frieden, wenn Sie ihn nicht auf den Ast genagelt hätten, würde er sich die
Radieschen von unten ansehen: Er hat's hinter sich und ist gestorben. Das ist ein Ex-
Papagei.'
```

```
>>> liste = regex.split(text)
>>> liste
['Dieser Papagei ist nicht mehr', 'Er hat aufgehört zu existieren', 'Er hat sein Leben
beschlossen und hat die Augen zugemacht', 'Er ist eine Leiche', 'des Lebens beraubt',
' er ruht in Frieden', 'Wenn Sie ihn nicht auf den Ast genagelt hätten', 'würde er sich
die Radieschen von unten ansehen', 'Er hat's hinter sich und ist gestorben', 'Das ist
ein Ex-Papagei']
```



Funktionen eines RE-Objekts

- Teilstrings ersetzen mit **sub()**

```
>>> regex = compile('\d')
>>> text = 'Kauft frische Bärte! Nur 20 Scheke1 das Stück! Ab 2 gibt es den 3ten zum Halben Preis!'

>>> regex.sub('_', text)
'Kauft frische Bärte! Nur __ Scheke1 das Stück! Ab _ gibt es den _ten zum Halben Preis!'
```

- Als optionalen Parameter kann man die Anzahl der Ersetzungen angeben

```
>>> regex.sub('_', text, 2)
'Kauft frische Bärte! Nur __ Scheke1 das Stück! Ab 2 gibt es den 3ten zum Halben Preis!'
```



Match-Objekte

- Die Methoden **match()** und **search()** geben ein Match-Objekt zurück, falls der reguläre Ausdruck auf den geprüften String passt.
- Diese Objekte besitzen Attribute und Methoden zur Auswertung

Attribut / Methode	Erklärung
re	Der reguläre Ausdruck (als RE-Objekt), der beim Aufruf von search() oder match() verwendet wurde.
string	Liefert den String, der geprüft wurde.
start()	Liefert den Index des Anfangs des passenden Teilstrings.
end()	Liefert den Index des Endes des längsten passenden Teilstrings.



Match-Objekte

```
>>> regex = compile('\d+')
>>> text = 'Kauft frische Bärte! Nur 20 Scheke! das Stück! Ab 2 gibt es den 3ten zum Halben Preis!'

>>> m = regex.search(text)                                #speichern des Match-Objekts

>>> m.re                                                    #verwendeter regulärer Ausdruck
re.compile('\\d+')

>>> m.string                                                #geprüfter String
'Kauft frische Bärte! Nur 20 Scheke! das Stück! Ab 2 gibt es den 3ten zum Halben Preis!'

>>> m.string[m.start():m.end()]                            #start und stopp Indizes des passenden Teilstrings
'20'

>>> m                                                       #Stringrepräsentation des Match-Objektes
<_sre.SRE_Match object,span=(25, 27), match='20'>
```



Reguläre Ausdrücke und Perl

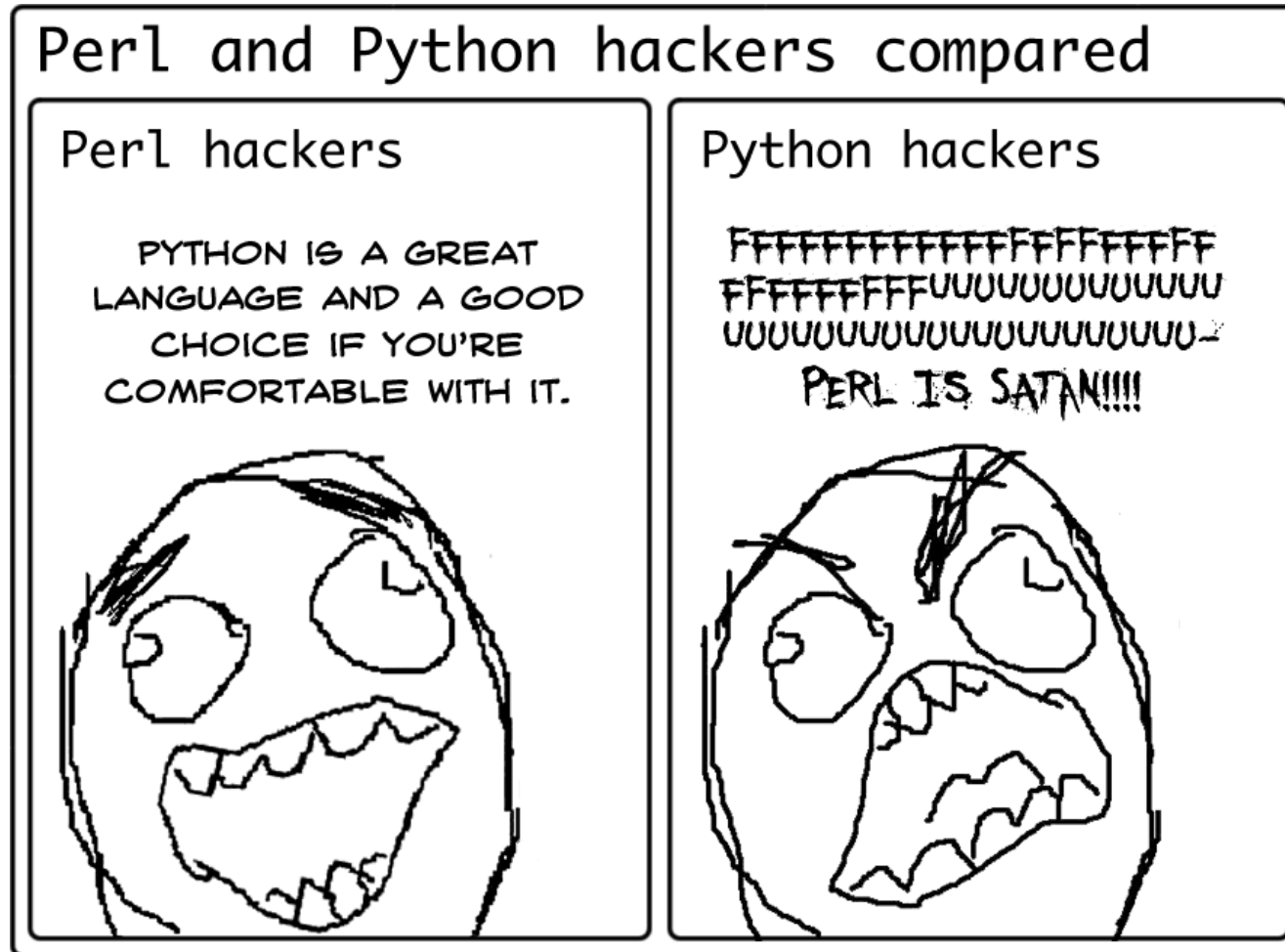
- Reguläre Ausdrücke sind in Perl fest verankert und damit sehr leicht anwendbar
- Im Prinzip ist Perl die Mutter aller Regulären Ausdrücke und viele andere Sprachen haben diese von Perl übernommen

```
$string = "Ausser effiziente regex kann Perl nicht viel tolles.";

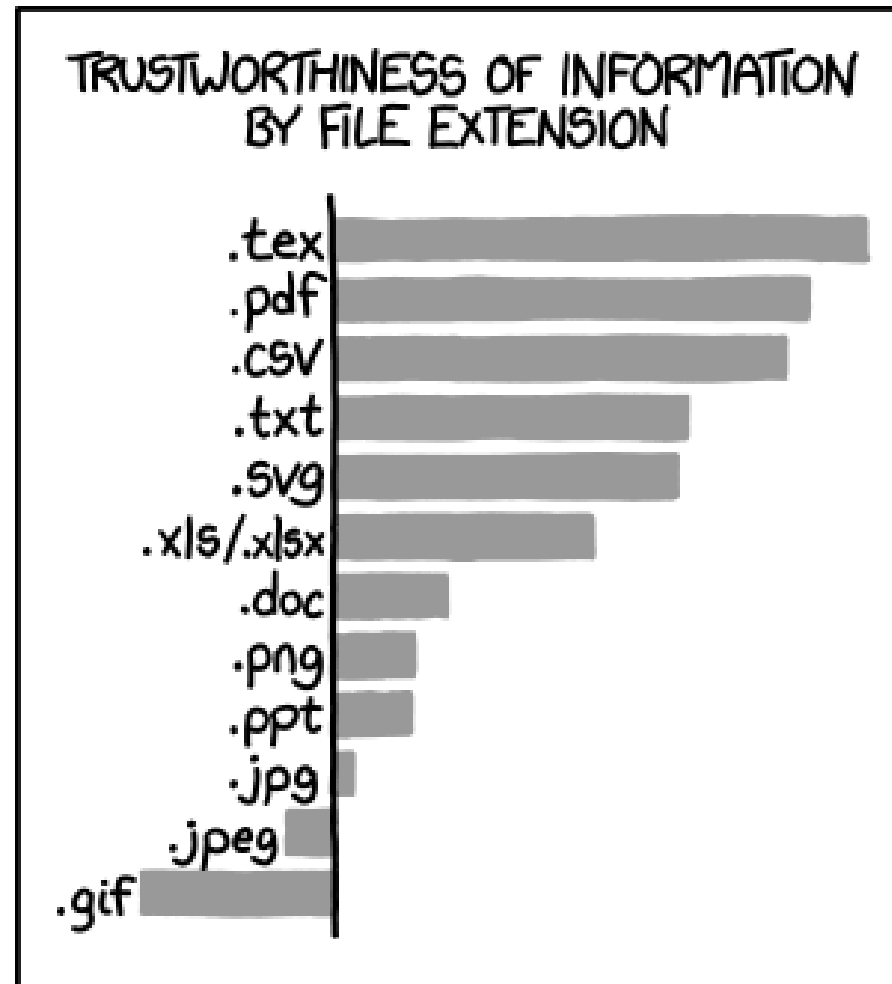
if($string =~ m/Perl/)
{ print "Ihhh, Perl gefunden!"; }
else
{ print "Puh... Kein Perl gefunden!"; }

$string =~ s/tolles/schönes;
print $string;
"Ausser effiziente regex kann Perl nicht viel schönes."
```

Reguläre Ausdrücke und Perl



Mehr über Dateiformate & Dateispeicherung



Dateiformate allgemein

- Binär:
 - Nicht menschenlesbar
 - Speichereffizient
 - Schnell zu parsen
 - Plattformabhängig (Little versus Big Endian)
 - Dateiformate lassen sich nur schwer erweitern
- Text:
 - Menschenlesbar (mal mehr mal weniger... siehe XML... oder Perl-Code)
 - Speicherineffizienter (?) → Nö, komprimierter Text ist oft kleiner als Binärdateien
 - Langsam zu parsen
 - Plattform unabhängig (Wenn Unicode-Kodierung benutzt wurde!)
 - Dateiformate lassen sich relativ leicht erweitern



Dateiformate allgemein – Beispiele

- CSV: einfachstes aller Textformate, Strings sind mit Kommas und Zeilenumbrüchen getrennt
- XML: häufigstes Textformat, erlaubt hierarchisch strukturierte Daten
- JSON: kompaktes Textformat in JavaScript Syntax, ist theoretisch immer interpretierbar
- YAML: ähnlich zu JSON, aber mit Einrückungen statt Klammern



Dateiformate allgemein

- Warum interessieren uns überhaupt andere Dateiformate, wenn wir doch *pickle* in Python haben?
 - Das *pickle*-Format lässt sich nur von einem Python-Interpreter einlesen
 - Es ist nicht sehr sicher
 - Es gibt Konvertierungsprobleme zwischen Python 2 und 3



CSV - *Character-separated values*

```
Name,Vorname,Alter,Rolle  
Chapman,Graham,48,"König Arthur,Wächter,Stimme Gottes"  
Cleese,John,66,"Sir Lancelot,Tim der Zauberer,Schwarzer Ritter"  
Gilliam,Terry,65,"Knappe Patsy,Sir Bors,Grüner Ritter"  
Idle,Eric,63,"Sir Robin,Diener Concord,Bruder Maynard"  
Jones,Terry,65,"Sir Bedevere,Prinz Herbert,Landarbeiterin"  
Palin,Michael,64,"Sir Galahad,Dennis,Herr des Sumpfschlosses"
```

- Zeilenseparator: Meistens ein ‘,’ aber häufig auch ein ‘\t’
- Spaltenseparator: Fast immer ein ‘\n’
- Strings können mit Anführungszeichen umschlossen sein um spezielle Zeichen zu maskieren
- CSV kann neben einfachen Texteditoren auch von Excel gelesen werden



CSV - Character-separated values

```
>>> import csv
>>> with open('/home/Documents/tabelle.csv') as e:
    reader = csv.reader(e)
    tabelle = []
    for row in reader:
        tabelle.append(row)

    tabelle = [row for row in csv.reader(e)]
    reader = csv.reader(e, delimiter='\t', quotechar="")
    reader = csv.reader(e, dialect="excel")

    tabelle = [['name', 'vorname', 'alter', 'rolle'], ['Idle', 'Eric', 63, 'Sir Robin']]
>>> with open('/home/Documents/tabelle2.csv', 'w') as f:
    writer = csv.writer(f)
    writer.writerows(tabelle)

    writer.writerows(tabelle[0])
    writer.writerows(tabelle[1])
```

#Reader-Objekt welches über die
#Eingabedatei iterieren kann
#und automatisch den CSV-Stil parsed

#Oder in kurz als List Comprehension



CSV - Character-separated values

- Für einfache, nicht verschachtelte Strukturen empfiehlt es sich immer CSV zu verwenden:
 - Es kann sehr einfach in jeder Programmiersprache eingelesen werden
 - Es kann in Texteditoren und Office-Programmen geöffnet werden
 - Tabs statt Kommas eignen sich meist besser als Separator, da Strings dann oft nicht maskiert werden müssen



JSON - *JavaScript Object Notation*

```
[  
  {  
    "name": "idle", "vorname": "eric", "alter": 63,  
    "rolle": ["Sir Robin", "Diener Concord", "Bruder Maynard",]  
  }  
]
```

- Eine einfache JavaScript Datei; schon fast mit der Python Syntax kompatibel
- Versteht fundamentale Datentypen (Bools, Zahlen, Strings, Arrays, Objekte)
- Sehr beliebt zur Kommunikation zwischen Webservern und Hosts
- Menschenlesbar (wenn gewollt), aber nicht so leicht zu parsen



JSON - *JavaScript Object Notation*

```
>>> import json
>>> with open('/home/Documents/tabelle.json') as e:
    tabelle = json.load(e)                                #Funktionen wie bei pickle

>>> tabelle[0]['name']
'idle'
>>> tabelle[0]['alter']
63

>>> tabelle = [['name', 'vorname', 'alter', 'rolle'], ['Idle', 'Eric', 63, 'Sir Robin']]
>>> with open('/home/Documents/tabelle2.csv', 'w') as f:
    json.dump(f, tabelle)

>>> jsonString = json.dump(tabelle)
```



JSON - *JavaScript Object Notation*

- Eignet sich für komplexe bzw. strukturierte Daten die zum Austausch gedacht sind:
 - Es kann direkt von verschiedenen Interpretern/Compilern interpretiert werden
 - Besonders in Webanwendungen und mobile Apps schon fast Standard



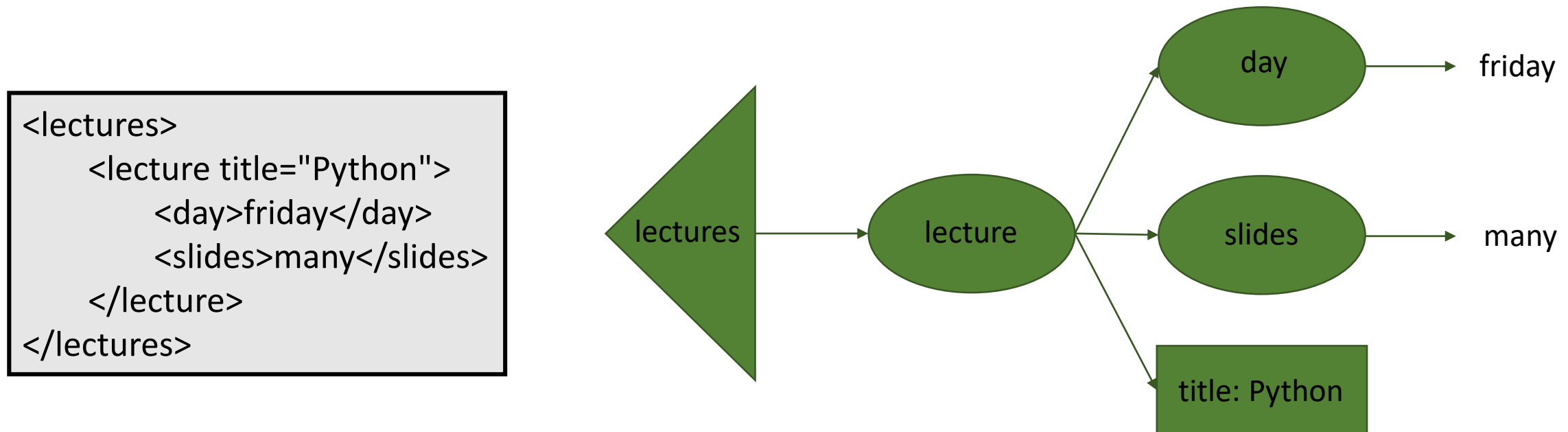
XML - *Extensible Markup Language*

```
<?xml version="1.0" encoding="UTF-8"?>
<someRoot>
  <someElem>
    <otherElem someAttr="value">Some text</otherElem>
  </someElem>
</someRoot>
```

- Meist benutztes Datenformat, da es plattform- und implementationsunabhängig ist
- Es ist eine Metasprache, d. h. daraus lassen sich eingeschränkte aber spezialisierte Sprachen definieren (z. B. XHTML, XAML, RSS, SVG, ...)
- Es ist (relativ) gut menschenlesbar



XML - *Extensible Markup Language*



- XML ist hierarchisch als Baumstruktur aufgebaut:
 - Es besteht aus Elementen die Inhalte haben (die wiederum Elemente sein können)
 - Elemente können zusätzlich Attribute haben
 - Es existieren verschiedene XML Parser, welche die Struktur einer XML-Datei auf unterschiedliche Weise wiedergeben



XML - *Extensible Markup Language*

```
>>> import xml.etree.ElementTree as ET
>>> root = ET.parse('/home/Documents/myXmlFile.xml').getroot()
>>> root.tag
'lectures'

>>> root[0]                                     #Erstes Kind von root
<Element 'lecture' at 0x0000013107F9A0E8>

>>> root[0][1]                                  #Zweites Kind vom ersten Kind von root
<Element 'slides' at 0x0000013107FA1728>

>>> root[0].getchildren()
[<Element 'day' at 0x0000013107FA16D8>, <Element 'slides' at 0x0000013107FA1728>]

>>> root[0].attrib['title']
'Python'

>>> for nodes in root[0]:
    print(node.text)
'friday'
'many'
```





ANYTHING SPECIAL OR UNIQUE ABOUT IT? ANYTHING THAT MAKES IT NOT LIKE ANY ELSE SCRIPTING LANGUAGE? ANYTHING THAT MAKES IT SUPERIOR AND MORE COOLER?

