

Objektorientierung und Funktionale Programmierung

Strings

Kristof Van Laerhoven
kvl@eti.uni-siegen.de

Michael Moeller
Michael.moeller@uni-siegen.de

uni-siegen.de



In Python sind strings beliebige Aneinanderreihungen von Zeichen, wobei Zeichen alles sind was man mit der Tastatur erzeugen kann (inklusive aller Sonderzeichen und dem ,unsichtbaren' Leerzeichen). Man kann strings mit einfachen, doppelten oder dreifach-doppelten Anführungszeichen erzeugen.

```
1 a = "Dies ist ein string\n"
2 b = 'Dies ist auch ein string\n'
3 c = "Dies ist ein string 'mit Anführungszeichen'\n"
4 d = 'Dies ist ein string "mit anderen Anführungszeichen"\n'
5 e = """Und
6     "so"
7     kann man
8     'viele'
9     Anführungszeichen verwenden
10    und sogar Absätze berücksichtigen"""
11 print('\n', a,b,c,d,e)
```

Der Ausdruck `\n` erzeugt eine neue Zeile.

```
In [13]: runfile('some_random_code_for_the_lecture.py')
```

```
Dies ist ein string
Dies ist auch ein string
Dies ist ein string 'mit Anführungszeichen'
Dies ist ein string "mit anderen Anführungszeichen"
Und
    "so"
    kann man
    'viele'
    Anführungszeichen verwenden
    und sogar Absätze berücksichtigen
```

Konkatenieren

```
15 string_1 = "Hallo"  
16 string_2 = ' zusammen!'  
17 string_3 = string_1 + string_2  
18 print(string_3)
```

Anzahl der Zeichen bestimmen

```
20 number_of_chars = len(string_3)  
21 print(number_of_chars)
```

Slicing: Das Zeichen an bestimmten Stellen auslesen

23	<code>print(string_3[2])</code>	1
24		
25	<code>print(string_3[0:3])</code>	Hal
26		
27	<code>print(string_3[0:6:2])</code>	Hlo
28		
29	<code>print(string_3[:3])</code>	Hal
30		
31	<code>print(string_3[-1])</code>	!

Strings wiederholen durch Multiplikation

```
35 print('hi'*5)    hihihihhi
```

- Python beginnt die Indizierung mit 0!
- Ein Ausdruck mit : wie 0:3 schließt die 3 nicht mit ein!
- Die Zahl nach dem zweiten Doppelpunkt ist die Schrittweite
- Negative Indizes verhalten sich zyklisch.

Aufteilen nach Trennzeichen

```
string_1 = "Hallo zusammen, wie geht's?"  
print(string_1.split(' '))  
print(string_1.split(','))  
  
['Hallo', 'zusammen,', 'wie', "geht's?"]  
['Hallo zusammen', " wie geht's?"]
```

Hier sehen wir, dass große Teile von Python eigentlich objektorientiert sind: strings sind eine **Klasse** für welche die **Methode** **split** implementiert ist. Dies soll uns aber an dieser Stelle nicht weiter beschäftigen.

Durch **split** entsteht ein neuer Typ von Variable, eine **Liste**. Wir gehen später genauer auf Listen ein. Hier können sie es als Sammlung von mehreren strings verstehen.

```
39 string_1 = "Hallo zusammen, wie geht's?"  
40 test = string_1.split(' ')  
41 print(type(test))  
42 print(test[1])  
43 print(test[1][0])
```

```
<class 'list'>  
zusammen,  
z
```

Zeichen zählen

```
string_1 = "Hallo zusammen, wie geht's?"  
test = string_1.count(' ')  
print('Es gibt', test, 'Leerzeichen in string_1!')
```

```
Es gibt 3 Leerzeichen in string_1!
```

Zeichenketten ersetzen

```
string_1 = "hallo, hallo zusammen!"  
test = string_1.replace('hallo', 'tschüss')  
print(test)
```

```
tschüss, tschüss zusammen!
```

Substrings finden

```
string_1 = "Hallo zusammen!"  
test1 = string_1.find('zusammen')  
test2 = string_1.find('tschüss')  
print(test1)  
print(test2)
```

```
6  
-1
```

Zahlen in strings umwandeln

```
test = 'Die Antwort ist ' + str(42)  
print(test)
```

```
Die Antwort ist 42
```

Formatieren von Strings

```
string_1 = 'Hier gibt es einen {} Platzhalter'
print(string_1.format('wunderbaren'))

string_1 = 'Man kann auch {} {} Platzhalter verwenden'
print(string_1.format('zwei', 'wunderbare'))

string_1 = 'Oder den {adjektiv} Platzhaltern Namen {verb}'
print(string_1.format(verb='zuordnen', adjektiv='wunderbaren'))
```

```
pi = 3.141592
string_1 = 'Man kann auch Zahlen wie {:.2f} formatieren'
print(string_1.format(pi))
```

Man kann auch Zahlen wie 3.14 formatieren

Siehe auch <https://docs.python.org/3/library/string.html#format-string-syntax>

Strings vergleichen

```
test = 'hallo'
print(test == 'hallo')
print(test == 'Hallo')
print(test != 'Hallo')
```

True
False
True

Das war mir zu viel...

Ich kann weder alle Manipulationsmöglichkeiten zeigen, noch können sie sich diese alle merken.

Wenn man eine (Basis-)Funktionalität sucht, am besten im Internet suchen!

Erste Anlaufstelle: Python Dokumentation <https://docs.python.org/3/library/string.html>

Dies gilt oft auch für Fehler im Code: Wenn man aus einer Fehlermeldung nicht sehen kann, was falsch am eigenen Code ist, kann es helfen zu suchen!

```
string_1 = 'Hi'  
string_1[1] = 'a'
```

```
File "C:\Users\Michael\sciebo2\lectures\lectures SoSe22\OFP\Code\temp_democode  
\some_random_code_for_the_lecture.py", line 80, in <module>  
    string_1[1] = 'a'  
  
TypeError: 'str' object does not support item assignment
```

stackoverflow.com/questions/10631473/str-object-does-not-support-item-assignment

▲ In Python, strings are immutable, so you can't change their characters in-place.

137 You can, however, do the following:



Diese Veranstaltung ist dazu da ihnen Konzepte wie **mutable** und **immutable** Datenstrukturen zu erklären. Dazu kommen wir noch.

Was gibt es noch außer Strings?

Es gibt drei Arten von vorimplementierten Zahlen in der Python

Integer

```
a = 1  
print(type(a))
```

```
<class 'int'>
```

= ganze Zahlen

Floats

```
b = 1.0  
print(type(b))
```

```
<class 'float'>
```

= Gleitkommazahlen
(nach IEEE 754 double precision binary
floating-point number format)

Komplexe Zahlen

```
c = 1.0 + 0j  
print(type(c))
```

```
<class 'complex'>
```

= Komplexe Gleitkommazahlen

Durch die Befehle `int`, `float` und `complex` lassen sich Zahlentypen ineinander konvertieren.

```
a = 1.0
b = int(a)
c = complex(a)
d = float(b)
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

```
<class 'float'>
<class 'int'>
<class 'complex'>
<class 'float'>
```

Auch Strings lassen sich in Zahlen umwandeln (wenn der String eine Zahl ist)

```
a = input('Gib deine Lieblingszahl ein: ')
b = float(a)+1
print(type(a), type(b))
print(b, ' ist größer als deine Lieblingszahl!')
```

```
Gib deine Lieblingszahl ein: 4.2
<class 'str'> <class 'float'>
5.2 ist größer als deine Lieblingszahl!
```

Folgendes sind übliche Operationen auf Zahlen

Operator	Beschreibung	Beispiel	Resultat
+	Addition zweier Zahlen	3+2	5
-	Subtraktion zweier Zahlen	3-2	1
*	Multiplikation zweier Zahlen	3*2	6
/	Division zweier Zahlen	3/2	1.5
//	Ganzzahlige Division	3//2	1
%	Modulus (Rest der Division)	5%3	2
**	Exponent (linker Wert hoch rechter Wert)	3**2	9

- Python liefert als Division von zwei Integers automatisch einen Float.
- Die Ganzzahlige Division rundet immer ab, z.B. ist $-4//3 = -2$, weil $-2 \leq -\frac{4}{3} < -1$
- Warnung: Floats werden nur mit endlicher Präzision gespeichert – die führt zu numerischen (Rechen-)fehlern!
- `print(0.1*3.0)` `0.30000000000000004`
- Alle obigen Operatoren können mit nachfolgendem = verwendet werden, um `x = x (Operator) Zahl` abzukürzen.

Zwei weitere wichtige Datentypen sind

- **Booleans** (welche nur die Werte **True** und **False** haben können), und
- **NoneType** was nur den Wert **None** haben kann (und oft zur Initialisierung verwendet wird).

```
1  gewinner = None
2
3  print(gewinner == None)
4  gewinner = "Michael"
5  print(gewinner != None)
```

```
True
True
```

Python Dokumentation

- Zahlen <https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex>
- Boolsche Werte / Testen <https://docs.python.org/3/library/stdtypes.html#truth-value-testing>
- Rundungsfehler bei Floating Point Operationen mittels Modul ‚decimal‘ minimieren:
<https://docs.python.org/3/library/decimal.html>

Was können sie jetzt schon?

Schreiben sie ein Programm, welches als Benutzereingabe ein Längenmaß in Metern entgegennimmt und dies in ein Längenmaß in cm umwandelt.

Strings mit `' '`, `" "` und `""" """`

Absatz `\n`

Strings zusammenführen `+`

Strings a-mal wiederholen `*a`

Slicing `[a:b:c]`

Länge `len`

Aufteilen nach Zeichen `z.split(z)`

Neuer Datentyp Liste `list`

Datentyp zurückgeben `type`

Anzahl der Substrings `z` zählen `.count(z)`

Substrings `z` durch `s` ersetzen `.replace(z,s)`

Index von Substrings `z` finden `.find(z)`



Andere Datentypen in string umwandeln `str`

Mit `{}` und `.format` strings formatieren

Sind zwei Dinge gleich? `==`

Sind zwei Dinge nicht gleich? `!=`

Datentypen `int`, `float`, `complex` und Umwandlung

Rechenoperationen `+` `-` `*` `/` `//` `%` `**`

Datentypen Boolean und Nonetype: `True`, `False`, `None`