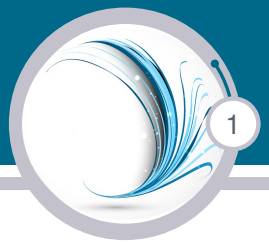
Institut für Informatik
Forschungsgruppe Datenbanken und Informationssysteme (DBIS)
Universität Innsbruck

Algorithmik und Programmieren

Martin Pichl
Datenbanken und Informationssysteme (DBIS)
dbis.uibk.ac.at
martin.pichl@uibk.ac.at

27. September 2017

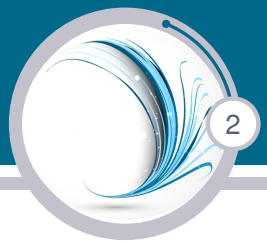
Inhalt



Theoretischer Input IV: Datenstrukturen

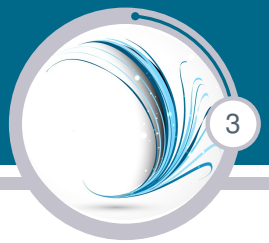
Theoretischer Input IV: Datenstrukturen

Theoretischer Input IV
Datenstrukturen



- Bisher: Einzelne Variablen und Konstanten
- Datenstrukturen erlauben es “große” Datenmengen zusammenzufassen und zu verarbeiten

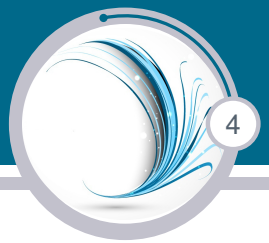
Theoretischer Input IV
Listen



- Die einfachste Datenstruktur in Python ist die Liste
- Eine Liste ist eine geordnete endliche Sammlung von Elementen beliebiger Datentypen
- Achtung: geordnet heißt nicht zwangsläufig sortiert!

```
1 >>>  list = [2,3,1,2]
2 >>>  list
3 [2, 3, 1, 2]
```

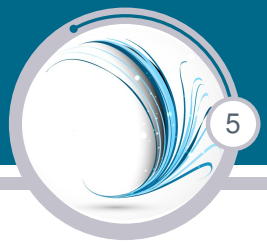
Theoretischer Input IV
Kontrollfluss



- Mit einem `for`-Statement kann ein Programmblock für jedes Element einer Sequenz `<sequence>` ausgeführt werden
- Diese `<sequence>` kann eine Liste, ein Tupel, eine Map ... sein

```
1 for  <variable> in <sequence>:
2   <executed once for each element in sequence>
3 else:
4   <executed once inthe end>
```

Theoretischer Input IV
Kontrollfluss Beispiel



- Mit dieser Schleife wird jedes Element der Liste [1,2,3] einmal ausgegeben
- Sie können statt der Variable i auch jeden anderen gültigen Bezeichner verwenden

```
>>> for  i in [1,2,3]:
...   print(i)
... else:
...   print('end')
...
1
2
3
end
```

Theoretischer Input IV
Zugriff auf Elemente in Listen

- Eine Liste kann Elemente verschiedener Datentypen beinhalten

```
>>> for  i in ['a',2.3,5]:
...    print(i)
...
a
2.3
5
```

- Einzelne Elemente bzw. Sublisten können auch referenziert werden

```
>>> list = [1,2,3]
>>> list[2]
3
>>> list[1]
2
>>> list[0:2]
[1, 2]
```

Theoretischer Input IV
Listenfunktionen I

- `append` hängt ein neues Element an die Liste
- `+` verkettet Listen

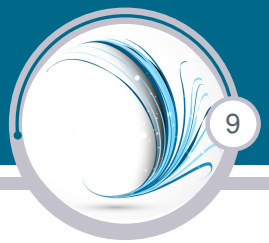
```
>>> list_a = ['a','b','c']
>>> list_b = ['x','y','z']
>>> list_a.append('d')
>>> list_a
['a','b','c','d']
>>> print(list_a + list_b)
['a', 'b', 'c', 'x', 'y', 'z']
```

Theoretischer Input IV
Listenfunktionen II

- `*n` wiederholt eine Liste n mal
- `in` prüft ob ein Element in der Liste enthalten ist
- `len(list)` gibt die Länge der Liste zurück

```
>>> list_b * 3
['x', 'y', 'z', 'x', 'y', 'z', 'x', 'y', 'z']
>>> 2 in [1,2,3]
True
>>> len(list_a)
3
```

Theoretischer Input IV
Tupel



- Ähnelt einer Liste
- Bereits aus der Mathematik bekannt: z.B. kartesische Koordinaten (x,y)
- Die meisten Funktionen für Listen können auch auf ein Tupel angewandt werden

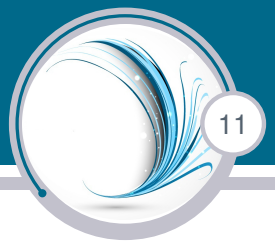
```
>>> tuple = (3,7)
>>> x,y = tuple
>>> x
3
>>> tuple[1]
7
>>> tuple * 3
(3, 7, 3, 7, 3, 7)
>>> tuple + (17,6,90)
(3,7,17,6,90)
>>> 3 in tuple
True
```

Theoretischer Input IV
Map

- Eine Map (auch: Dictionary) ist eine Sammlung aus Schlüssel/Wert Paaren
- Einem eindeutigen Schlüssel wird dabei ein Wert zugewiesen (der nicht eindeutig sein muss)
- Maps haben keine Ordnung

```
>>> price = {'car':30000, 'computer':2000, 'shoes':70, '
    cheap_mobile':70}
>>> price['shoes']
370
>>>price.values()
[30000,2000,70,70]
>>>price.keys()
['car','computer','shoes','cheap_mobile']
>>>len(price)
94
>>>'car' not in price
False
```

Theoretischer Input IV
Zeichenketten



- Strings werden in Python mit einfachen oder doppelten Hochkomma gekennzeichnet
- Zwischen beiden Varianten gibt es keinen Unterschied
- Bei einfachen Hochkommas darf das doppelte Hochkomma im String enthalten sein (und umgekehrt)
- Viele Funktionen die bei Listen besprochen wurden, können auch auf Strings angewendet werden: `for`,`+`,`*`,`n` ...

```
>>> text = 'a' + "b" + "c"
>>> for ca in text:
...     print c
...
a
b
c
>>> "abc" * 3
'abcabcabc'
>>> "test" in "this is a texttext!"
True
```

Theoretischer Input IV
Zeichenketten

- Umfangreiche Funktionen für Strings sind im Modul `string` zusammengefasst
- Mehr dazu in der Python Dokumentation

Theoretischer Input IV
Referenzkopie und echte Kopie

- Möchten Sie eine Datenstruktur kopieren, ist eine einfache Zuweisung nicht genug!
- Im unten angeführten Beispiel kopiert `list_b` nur eine Referenz, d.h. zwei Listen greifen auf den selben Speicherbereich zu
- Änderungen in einer der beiden Listen sind in der anderen unmittelbar sichtbar.

```
1 >>> list_a = [1 ,2 ,3]
2 >>> list_b = list_a
3 >>> list_c = list(list_a)
4 >>> list_a[1] = 5
5 >>> list_b
6 [1 , 5 , 3]
7 >>> list_c
8 [1 , 2 , 3]
```

Bis zum nächsten Mal :)