Lukas Müller

Programmieren für Studierende der Naturwissenschaften

V3 – Aggregierte Datentypen

Inhalte

- V1: Grundlagen der Programmierung
  P1: Hilfe beim Einrichten von Python an eigenen Rechnern, erste Programme ausführen
- V2: Elementare Datentypen und Kontrollstrukturen
  P2: Übungen
- V3: Aggregierte Datentypen
  P3: Übungen
- V4: Aggregierte Datentypen und Funktionen
  P4: Übungen
- V5: Testen, Fehlermeldungen und Selbsthilfe
  P5: Übungen

Inhalte

- V6: Externe Packages, Einführung NumPy und SciPy
  P6: Übungen
- V7: Externe Packages 2
  P7: Übungen
- V8: Umgang mit externen Daten und Visualisierung
  P8: Übungen
- V9: Entwurf von Algorithmen ODER Aufarbeitung besprochener Themen
  P9: Übungen, selbstständige Arbeit in Kleingruppen
- V10: Betriebssysteme (Windows, Linux, macOS) ohne Übung

Schleifen Rückblick – Allgemein

- Die kopfgesteuerte oder vorprüfende Schleife:
  - zuerst wird die Abbruchbedingung geprüft wird, bevor der Schleifenrumpf durchlaufen wird (meist durch das Schlüsselwort WHILE (=solange-bis) angezeigt.

- Die fußgesteuerte oder nachprüfende Schleife (mit einem Trick in Python implementierbar):
  - erst nach dem Durchlauf des Schleifenrumpfes wird die Abbruchbedingung überprüft z.B. durch ein Konstrukt REPEAT-UNTIL (=wiederholen-bis).

Zählschleife

Python Syntax:

```
for <Variablenname/Laufindex> in <Sequenz>:
    <Anweisungen>
<Anweisungen hinter der Schleife>
```

While-Schleife

Python-Syntax:

```
while <logischer Ausdruck>:
   <Anweisungen>
else:
   <Anweisungen>
<Ausdruck hinter Schleife>
```

- Das “while-Statement” benutzt man zur wiederholten Ausführung des Code-Blockes (suite), solange die “expression” zu “True” ausgewertet wird
- Wenn der logische Ausdruck zu “False” ausgewertet wird, wird die Anweisung des else-Zweigs ausgeführt, falls vorhanden (SEHR SELTEN!)
- In der Anweisung muss der Wahrheitswert des logischen Ausdrucks irgendwann geändert werden, sonst Gefahr einer Endlosschleife!

Beispiele für Schleifen

```
count = 0
while count < 9:
    print('The count is:', count)
    count = count + 1

print("Good bye!")
```

```
var = 1
while var == 1 :
    num = raw_input("Enter a number  :")
    print("You entered: ", num)
         # This constructs an infinite loop
print("Good bye!")
```

- Eigentlich ein schlechtes Beispiel. Hier könnte man einfacher eine for-Schleife verwenden.
- Beispiel für Konstruktion einer Endlosschleife.

Beispiele für Schleifen

temp.py - /Users/alexanderwolodkin/Documents/temp.py (3.9.0)

```
password = ""
trys = 0

while password != "111111":
    password = input("Geben Sie das korrekte Passwort ein: ")
    trys += 1
else:
    print("Geschafft nach nur ", trys, "Versuch(-en)")
```

Python 3.9.0 Shell

```
Geben Sie das korrekte Passwort ein: 123
Geben Sie das korrekte Passwort ein: 321
Geben Sie das korrekte Passwort ein: admin
Geben Sie das korrekte Passwort ein: 111111
Geschafft nach nur  4 Versuch(-en)
>>>
```

temp.py - /Users/alexanderwolodkin/Documents/temp.py (3.9.0)

```
password = ""
trys = 0

while password != "111111":
    password = input("Geben Sie das korrekte Passwort ein: ")
    trys += 1

print("Geschafft nach nur ", trys, "Versuch(-en)")
```

Python 3.9.0 Shell

```
Geben Sie das korrekte Passwort ein: qwertz
Geben Sie das korrekte Passwort ein: asdf
Geben Sie das korrekte Passwort ein: 123456
Geben Sie das korrekte Passwort ein: 111111
Geschafft nach nur  4 Versuch(-en)
>>>
```

Strings

- „String“ ist ein Datentyp. Ein String ist eine Zeichenkette und wird in Anführungszeichen geschrieben.
- Vergleichsoperationen auf Strings sind möglich, aber nicht unbedingt intuitiv.

Indexierung (Strings und Listen)

- Konzeptionelles Modell des Strings: Zeichen stehen in „Zellen“
- Diese Zellen sind nummeriert
- Jedes Zeichen ist unter seinem eigenen Index `i` in eckigen Klammern zugreifbar: `s[i]`

Slicing (String und Listen)

- Extrahiert zusammenhängende (Zellen) -bereiche über Indizes. Ergebnis ist wieder ein String.
- Regel: `s[a:b]` extrahiert Teilbereich von Index a bis b-1
  - `s[a]` ist das erste Element im Teilbereich
  - `s[b-1]` ist das letzte Element im Teilbereich
  - Beispiel: `s[1:5]` geht von 1 bis **ausschließlich** 5
- Regel: `s[a:]` geht von Index a bis zum letzten Element
  - Beispiel: `s[2:]` geht von Index 2 bis zum letzten Element
- Regel: `s[:]` erstellt eine echte (!) Kopie von `s`

Indexbereiche

Indexwerte außerhalb des Wertebereichs [0,11] oder [-12,-1] führen zu einem "IndexError".

```
>>> S[-12]
'E'
>>> S[-13]
Traceback (most recent call last):
File "<pyshell#204>", line 1, in …
IndexError: string index out of range
```

Anders **beim Slicing**, dort werden die mit roten Pfeilen markierten Indexwerte auf ± len(S) „gecasted"

```
>>> S[9:14]
'oll'
>>> S[-9:-12]
''
>>> S[-14:3]
'EPR'
```

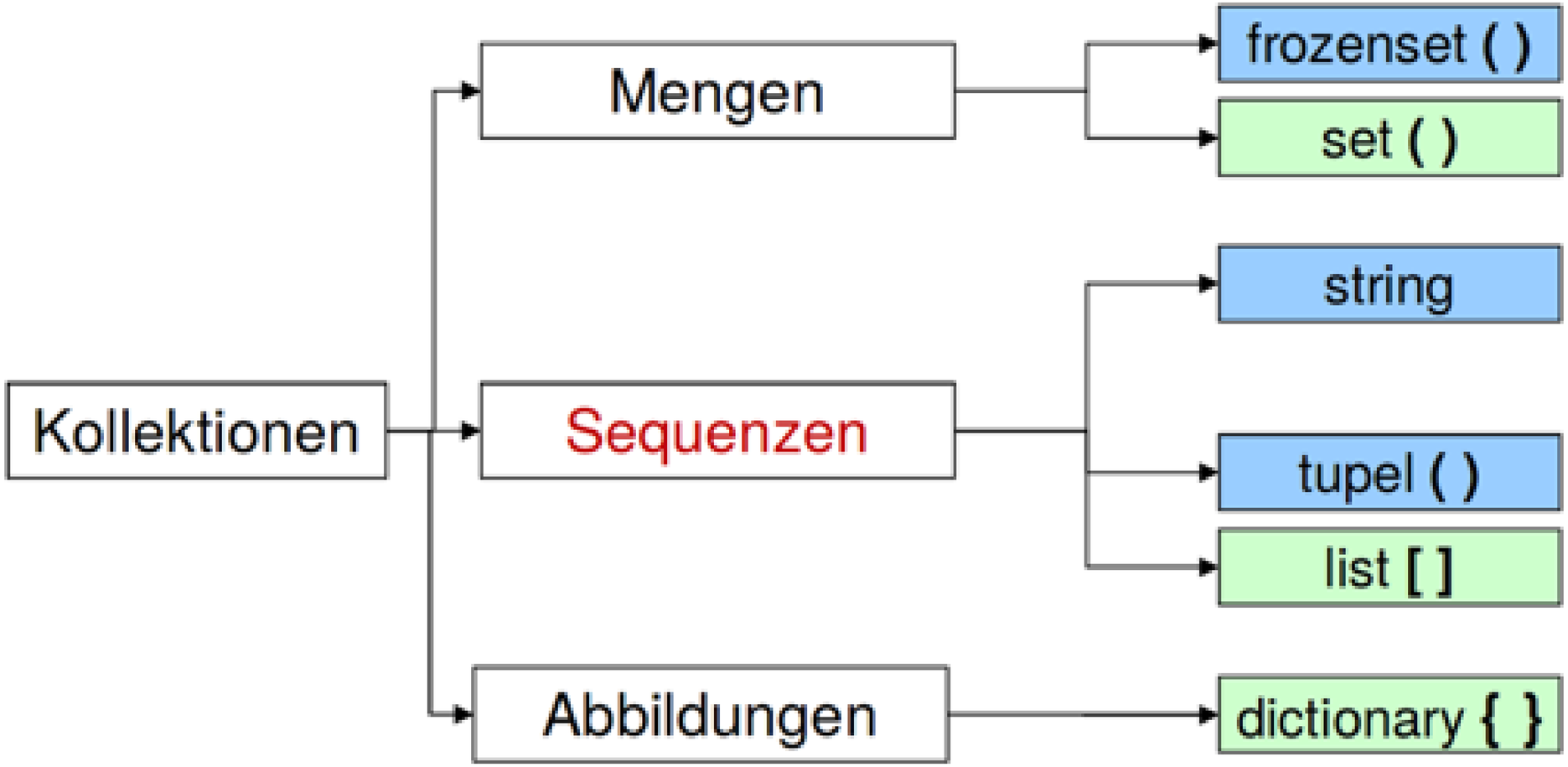
Funktionen auf String

- len() : Gibt die Anzahl an Zeichen in einem String aus.
- Groß- und Kleinschreibung: .lower() und .upper()
- Teilstrings ersetzen: s.replace(<zuErsetzen>, <Ersatzstring>)
- Teilstrings zählen: s.count(<zuZählen>)

Aggregierte Datentypen

- Ein Datentyp (in der Informatik) ist die Zusammenfassung von Objektmengen mit den darauf definierten Operationen
- Zusammengesetzte (aggregierte) Datentypen sind Datenkonstrukte, welche aus einfacheren (elementaren) Datentypen oder zusammengesetzten Datentypen zusammengesetzt sind. Der Fokus liegt hierbei auf den Operationen
- Aggregierte Datentypen stehen in den meisten Programmiersprachen zur Verfügung
- Programmiersprachen können aggregierte Datentypen auf zwei Ebenen unterstützen:
  - Sie stellt Konstruktoren zur Verfügung, mit deren Hilfe der Aufbau eines aggregierten Datentyps beschrieben werden kann
  - Sie verfügen über vordefinierte aggregierte Datentypen (builtins in Python)

Aggregierte Datentypen in Python



Mutable und Immutable - veränderlich und unveränderlich

- Einträge in einem frozenset sind im Gegensatz zu einem set immutable.

```
a = frozenset([4,3,2,1])
print(a)

a[1] = 0
print(a)
```

```
Python 3.9.0 Shell
temp.py
frozenset({1, 2, 3, 4})
Traceback (most recent
call last):
  File "/Users/alexande
rwolodkin/Documents/tem
p.py", line 12, in <mod
ule>
    a[1] = 0
TypeError: 'frozenset'
object does not support
item assignment
```

```
temp.py
my_text = "Text123456789"
my_text[2]="E"
print(my_text)
```

```
Python 3.9.0 Shell
Traceback (most recent
call last):
  File "/Users/alexande
rwolodkin/Documents/tem
p.py", line 2, in <modu
le>
    my_text[2]="E"
TypeError: 'str' object
does not support item a
ssignment
>>>
Ln: 837   Col: 4
```

Set / Menge

- Anlegen einer Menge:

```
my_set = set([1, 2, 3])
my_set_1 = set("Testmenge")
```

- Mehrfach vorkommende Elemente werden gelöscht
- {1,2,3,3} = {1,2,3} = {3,1,2}

Listen

- Eine Liste ist eine **zusammengesetzte** (keine elementare) Datenstruktur, die in Python eine der Sequenztypen (wie String) ist, d.h, jedes Element hat genau einen (oder keinen) Nachfolger und ist damit iterierbar

- Der Inhalt lässt sich über **Indizes** ansprechen (wie bei Strings). Auch **Slicing** funktioniert!

- Listen werden durch **eckige Klammern** angezeigt, die Listenelemente durch Kommata getrennt. Bsp.: [1, 15, 11, 23] oder [3.141, 'ufbasse', 1, 2]

- Man kann Listenelemente „in place“ ändern (das ist bei vielen anderen Datentypen, u.a. Strings, nicht möglich). Daher heißen Listen: **mutable**

- **ACHTUNG, Datentypen!** In Python dürfen verschiedene Datentypen in einer Liste liegen. In anderen Programmiersprachen ist das nicht immer so

Listen

- Listenelemente können einen beliebigen, nicht einheitlichen Datentyp haben, insbesondere auch Listen (Listen von Listen)

```
>>> a = [[1, 4, 9], 15, [3, [11,5]], 23]
>>> a[0]
[1, 4, 9]
>>> a[0][2]
9
>>> a[2][1][1]
5
```

- **Achtung:** Man kann NICHT wie in anderen Programmiersprachen oder der Mathematik üblich mit durch Kommata getrennte Indizes auf Elemente zugreifen, also: a[0,2] führt zu einem Fehler

Listen - Operationen

`L.append(x)`	Das Element x hinten an L anfügen
`L.clear()`	Leert die Liste L
`L.count(x)`	Zählt das Auftreten von x in der Liste L
`L.index(x)`	Gibt den Index des ersten Auftretens von x in L zurück
`L.extend(x)`	Erweitert L um die Liste x
`L.insert(i, x)`	Fügt x am Index i in Liste L ein (und verschiebt den Rest von L nach hinten)
`L.remove(x)`	Entfernt alle x aus Liste L
`L.reverse()`	Dreht die Liste L um
`L.sort()`	Sortiert die Elemente der Liste

Tupel

- Tupel werden als durch Komma getrennte Werte in runden Klammern notiert: (x, y, z).
- Sie sind nicht „in place“ veränderbar (sie sind immutable).
- Das Lesen von Einträgen funktioniert wie bei Listen.

Tupel – Beispiele

```
>>> tup = (1,2,3)
>>> tup(1)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    tup(1)
TypeError: 'tuple' object is not callable
>>> tup[1]
2
>>> tup[1] = 5
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    tup[1] = 5
TypeError: 'tuple' object does not support item assignment
>>> for i in tup:
        print(i)


1
2
3
```