Lukas Müller

Programmieren für Studierende der Naturwissenschaften

V4 – Weitere aggregierte Datentypen und Funktionen

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

Lernen

- Etwas klappt nicht? Bitte nicht verzweifeln!
  - Neue Dinge lernen braucht **Zeit**
  - Ihr würdet auch nicht erwarten, nach 3 Tagen japanisch zu sprechen oder ein Pferd schnitzen zu können

- **Beschreibung des Problems:**
  - Zerlegen in Teilprobleme
  - Teilprobleme sprachlich ausformulieren (nicht einfach in der Aufgabenstellung markieren)
  - Zeichnen, visualisieren, wenn die Zusammenhänge komplexer erscheinen
  - Wesentliche Objekte definieren und gegebenenfalls ihre Eigenschaften beschreiben
  - Dabei nicht zu feingranular denken

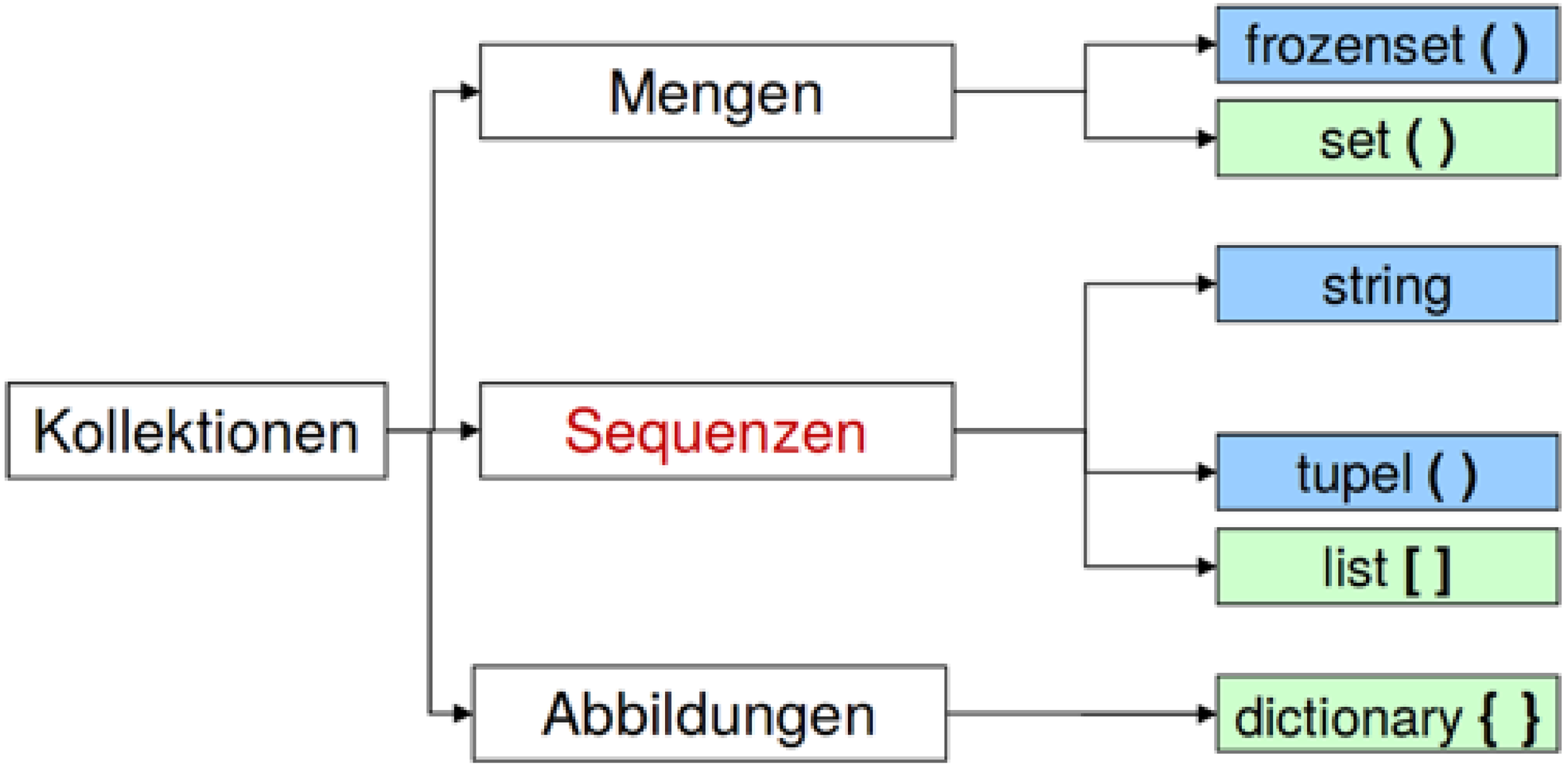
Schleifentypen

Fragen von eurer Seite?

„Durchzählen“ vs. logische Bedingung

ABER: Oft gibt es eine Lösungen mit beiden Schleifentypen

Rückblick


Kollektionen
Mengen
Sequenzen
Abbildungen
frozenset ( )
set ( )
string
tupel ( )
list [ ]
dictionary { }

Set und Frozenset

- Wie in der Mathematik:
  - Die Reihenfolge ist nicht von Bedeutung
  - Es geht um „enthalten sein“, daher keine Dopplungen

- Hauptnutzen ist:
  - Testen auf Mitgliedschaft, Duplikate entfernen
  - Klassische Mengenoperationen wie Durchschnitt oder Differenz zwischen Mengen berechnen

- Anlegen von Mengen:
  - Durch Konvertierung aus einem iterierbaren Datentyp (z.B. string oder list)
  - Elemente müssen immutable und vergleichbar sein
  - **Achtung!** Die leere Menge lässt sich nicht durch {} erzeugen! Das wäre ein dictionary

Beispiele

```
>>> a = {1,2,3}
>>> a = {1,2,3,4,1,1,2}
>>> a
{1, 2, 3, 4}
>>> b = set('Test')
>>> b
{'T', 's', 't', 'e'}
>>> b = set([1,2,3,2])
>>> b
{1, 2, 3}
>>> b = {}
>>> type(b)
<class 'dict'>
>>> c = {[1,2], [2,3]}
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    c = {[1,2], [2,3]}
TypeError: unhashable type: 'list'
```

Operatoren =

`len(N)`	Kardinalität von N	Integer
`x in M`	(True/False)	Ist x Element von M?
`x not in M`		ist x nicht Element von M?
`N <= M  N < M`	M.issubset(N)	N istTeilmenge von M (True/False)
`N >= M  M > N`	M.isssuperset(N)	N ist Obermenge von M (True/False)
`M | N`	M.union(N)	Vereinigung von M und N
`M |= N`	M.update(N)	
`M & N`	M.intersection(N)	Schnittmenge von M und N
`M &= N`	M.intersection_update(N)	
`M - N`	M.difference(N)	Differenz von M und N
`M -= N`	M.difference_update(N)	
`M ^ N`		Symmetrische Differenz

Mengen

- `m.add(x)`
  Füge ein Element x zur Menge M hinzu. (Hat keine Wirkung, wenn x schon Element von M ist)
- `m.clear()`
  Leert die Menge M
- `m.pop()`
  Entfernt ein Element aus der Menge M
- `m.remove(x)`
  Entferne Element x aus der Menge M (x muss ein Element von M sein, sonst Keyerror)

… Python Doku anschauen und ausprobieren!

Dictionary

- Ein bisschen anders, als die anderen aggregierten Datentypen
- Dictionaries implementieren partielle Funktionen. Hierzu benutzt man 2-Tupel (Paare) der Form (Schlüssel, Wert), geschrieben zum Beispiel als {Schlüssel:Wert}
- Da der Wert wieder ein n-Tupel sein kann, aber auch eine Liste, etc. sind beliebigen partielle Funktionen implementierbar.
- Im Gegensatz zu Sequenzen, die mit einem Zahlen Intervall indiziert werden, werden Dictionaries über Schlüssel indiziert
- Die Schlüssel müssen irgendeinen immutable Typ haben. Strings und Zahlen können also immer Schlüssel sein. Tupel können als Schlüssel verwendet werden, wenn sie nur Strings, Zahlen, Tupel, Frozensets enthalten

Dictionary

- Ein Paar geschweiften Klammern {} erzeugt ein leeres Dictionary
- Eine durch Kommata getrennte Folge von (Schlüssel:Wert)-Paaren innerhalb der Klammern fügt die initialen Paare in das Dictionary ein.

- Hauptoperationen auf einem Dictionary sind:
  - Das Speichern eines Wertes unter einem Schlüssel und das Abrufen dieses Wertes bei Angabe des Schlüssels
  - Es ist auch möglich, ein (Schlüssel:Wert)-Paar mit __delitem__ () zu löschen
  - Wird beim Speichern ein Schlüssel verwendet, der bereits existiert, so wird der alte damit assoziierte Schlüssel vergessen.
  - Es erzeugt einen Fehler, einen Wert mit einem nicht existierenden Schlüssel abzurufen.

Datentyp None

- Der Datentyp None hat in Python nur einen einzigen Wert:
  - Die Konstante None
  - None ist ein Schlüsselwort. Es dient als Platzhalter für Variablen, die eigentlich keinen Wert haben (oder dieser noch nicht bekannt ist)

- Funktionen, die keinen Wert zurückgeben, haben implizit None als Rückgabewert
  - Wenn der interaktive Interpreter einen Ausdruck auswertet, gibt er ihn nur aus, wenn der Rückgabewert nicht None ist
  - Testen in der Konsole

```
>>> a = print("test")
test
>>> print (a)
None
```

Funktionen

Funktionen

Warum reichen die Kontrollstrukturen, die wir bisher kennen nicht aus??

- Funktionen ermöglichen:
  - eine bessere Strukturierung von Programmen
  - Modularisierung (viele Bauteile mit definierten Schnittstellen)
  - Wiederverwendung von Codeteilen extern und intern
  - Effizienzsteigerung der Kompilate

- ***Achtung: Im Rahmen dieser Veranstaltung werden die Unterschiede zwischen Funktionen und Methoden nicht näher betrachtet!***
  - Studierende der Informatik lernen diese Unterschiede im Kontext der „Objektorientierten Programmierung“ kennen

Unterprogramme – Funktionen

- Eine Folge von Anweisungen wird unter einem Namen zusammengefasst
- Es können Argumente (sog. Parameter) an diese Folge übergeben, und ggf. auch ein Wert oder Werte zurückgeliefert werden
- Die Parameter werden in der Regel durch Reihenfolge, Typ und Anzahl und/oder durch Namen festgelegt
- **Die gute Nachricht:** Wie man so etwas aufruft, wisst ihr schon!

Funktionen

- Funktionen werden mit der `def`-Anweisung definiert, Parameter in runden Klammern direkt dahinter.
- Der Funktionsrumpf muss **eingerückt** werden.
- Das Ende der Funktionsdefinition wird durch Rücknehmen der Einrückung angegeben.
- Return ist das Schlüsselwort, das veranlasst, dass der Wert dem Funktionswert zugewiesen und die Funktion beendet wird.

```
def add(x, y):
    print(x, y)
    return x + y
```

- Eine Rückgabe (**return**) ist optional. Aber Achtung! Wird kein Wert angegeben oder wird die Return-Anweisung weggelassen, so wird das Objekt **None** zurückgegeben.

Funktionen

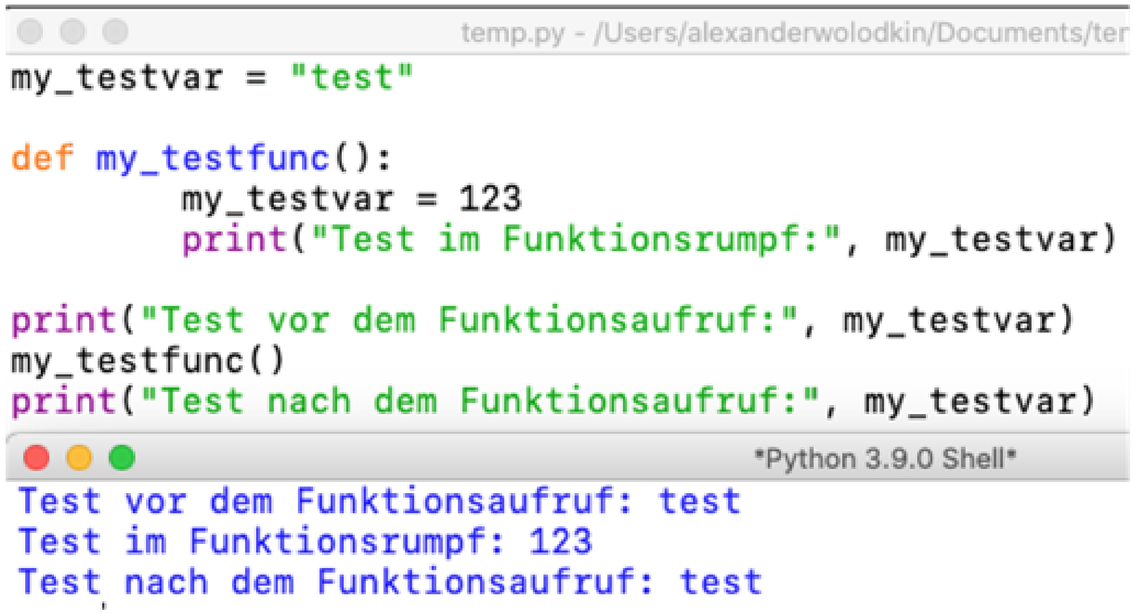
Aufrufe

- Die Funktionsdefinition muss im Programmtext (lexikalisch) vor dem Aufruf erfolgen (erst dann ist der Name der Funktion bekannt)
- Alle Argumente in der Funktionsdefinition müssen beim Aufruf konkret und in der richtigen Reihenfolge angegeben werden

- Default – Parameter

Namensräume

- Alle Elemente, die wir verwenden oder reservieren, liegen in einem Namensraum.
- Funktionen bilden eigene Namensräume
- Auch Module (siehe kommende Themen) können eigene Namensräumebilden



- Zugriffe auf Elemente in unterschiedlichen Namensräumen sind nicht sofort intuitiv verständlich. Beachten Sie das angegebene Beispiel