Zusammenfassung Einführung in die Programmierung WS1819

Clemens Mittermaier

Wintersemester 2018/19

Inhaltsverzeichnis

1 Grundlagen

1.1 Definition verschiedener Begriffe:

Informatik ist die Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von Digitalrechnern (Computern).

Computer sind universell einsetzbare Geräte zur automatischen Verarbeitung von Daten. Bei einem Computer wird zwischen vier Konzepten unterschieden:

1. Input/Output
2. Algorithmus
3. Programm
4. (Berechnungs)prozess

Algorithmen sind Vorschriften zum Durchführen einer Berechnung (Folge von Einzelschritten) mit folgenden Eigenschaften:

- Präzision: Die Bedeutung jedes Einzelschrittes ist eindeutig festgelegt.
- Effektivität: Jeder Einzelschritt ist ausführbar.
- Finitheit (statisch): Die Vorschrift ist ein endlicher Text.
- Finitheit (dynamisch): Bei der Ausführung wird nur endlich viel Speicher benötigt.
- Terminierung: Die Berechnung endet nach endlich vielen Einzelschritten-für alle legalen Eingaben.

Weitere gewünschte Eigenschaften sind:

- Determinismus: Die Folgeschritte sind immer eindeutig festgelegt.
- Determiniertheit: Bei gleicher Eingabe erzeugt die Vorschrift die gleiche Ausgabe- berechnet also eine Funktion.
- Generalität: Die Vorschift kann von eine ganze Klasse von Problemen lösen.

Programm ist der Algorithmus notiert in einer geeigneten Sprache. Es gibt verschiedene Arten von Programmiersprachen:

- Systemprogrammiersprachen
- Höhere Programmiersprachen
- Deklarative Programmiersprachen

2 Erste Schritte in Python

Python 3 ist eine objektorientierte, dynamisch getypte, interpretierte und interaktive höhere Programmiersprache.

int	Integer	Ganze Zahlen	0,1,2,3
float		Rationale Zahlen	4.5, 5.01
complex		Komplexe Zahlen	4+i
str	String	Zeichenkette	'abcd'
bool	Boolean	Wahrheiten	'True', 'False'

2.1 Verschiedene Typen

Funktonsaufrufe Funktionen sind Abbildungen von einem Definitionsbereich auf einen Bildbereich. Eine Funktion erwartet Argumente aus dem Definitionsbereich und gibt einen Funktionswert (oder Rückgabewert) aus dem Bildbereich zurück. Funktionen sollen vorranging lokale Variablen uns Parameter nutzen. Funktionen können globale Variablen lesen, falls es keine lokale Variable gleichen Namens gibt.

Bedingte Anweisungen Erlauben die Auswahl zwischen alternativen Anweisungen

Programme richtig schreiben Programme können auf verschiedene Weisen gestartet werden:

- Expliziter Aufruf von Python3
- Starten als Skirpt
- Starten durch Klicken
- Starten druch Import
- Starten mit einer IDE

2.2 Sequenzen

Drei Sequenz-Typen:

- Strings
- Tupel
- Listen

Strings Strings in Python enthalten Unicode-Zeichen

Tupel und Listen Tupel und Listen sind eine Sequenz von Objekten. Tupel werden mit () gekennzeichnet, Listen mit []. Tupel sind unveränderlich, Listen lassen sich verändern.

Sequenzen Strings, Tupel und Listen sind Sequenzen. Sie enthalten andere Objekte in einer bestimmten Reihenfolge. Sequenztypen unterstützen die folgenden Operationen:

- Verkettung
- Wiederholung
- Indizierung

- Mitgliedschaftstest
- Slicing
- Iteration

Iteration Das Durchlaufen einer von Sequenzen mit for-Schleifen. **Wichtige Anweisungen:**

- Break: Beendet die Schleife vorzeitig
- Continue: Beendet die Schleifeniteration vorzeitig
- Else: Wird nach dem Beenden der Schleife ausgeführt, falls es keinen Break gab.

2.3 Entwurf von Schleifen

Funktionen über Sequenzen verwenden for-in-Schleifen. Ergebnisse werden meist mit einer Akkumulatorvariable berechnet. Funktionen über mehrere Sequenzen verwenden for-range-Schleifen. Der verwendete Range hängt von der Problemstellung ab. While-Schleifen werden verwendet, wenn die Anzahl der Schleifendurchläufe nicht von vorne herein bestimmt werden kann oder soll. Typischerweise beim Verarbeiten von Eingaben oder bei der Berechnung von Approximationen. Jede While-Schleife muss eine dokumentierte Terminationsbedingung haben.

3 Objekte und Klassen

3.1 Objekte und Attribute

Alle Werte werden in Python als Objekte gespeichert. Das bedeutet, dass sie auch Attribute und Methoden haben, die man mit Punktnotation aufrufen kann.
Jedes Objekt besitzt eine Identität, die es von alles anderen Objekten unterscheidet. Der *X is Y* ist *True* wenn X und Y das selbe Obejkt sind. Es gibt einen Unterschied zwischen Gleichheit (Zwei Objekte haben den selben Inhalt) und Identität (Zwei Objekte sind identisch).
Verwende in der Regel den Gleichheitstest.

NoneType Der *NoneType* hat nur einen einzigen Wert: *None*.

3.2 Klassen beschreiben Objekte

Klassen beschreiben Objekte mit gleichen Eigenschaften (**Attribute**) Ein **Record** ist eine Objekt, das mehrere untergeordnete Objekte (**Attribute**) enthält.
Neue Records und Klassen werden mit der *class*-Anweisung eingeführt.

Erzwugung von Instanzen Mit jedem Aufruf der Klasse als Funktion wird eine neue Instanz erzeugt. (Das ist eine Objekt in Java, das alle Attribute und Methoden der Klasse besitzt.) Alle erzeugten Instanzen sind untereinander nicht-identisch und ungleich!
Instanzen sind Records, welche dynamisch neue Attribute erhalten können. (Über die Punkt-Notation. z.B. phone.name().)

3.3 Aufbau einer Klasse

Eine Klasse benötigt einen individuellen Namen und vor allem eine init-Funktion.Die Klasse hat immer ein Argument weniger als init-Funktion. Der erste Parameter heißt immer *self*
class Article:
def –init–(self, name, price):
self.name = name
self.price = price

Damit man ein Objekt problemlos von einer Metode in die andere Übergeben kann gibt es die **repr**-Methode. Das ist ein String, der von Python wieder eingelesen werden kann.

4 Bäume

Bäume sind nichts neues... Ein Baum besteht aus einer Wurzel, hat nen Haufen Knoten, und alle Knoten, die nicht zu weiteren Knoten führen heißen Blätter.

4.1 Binärbäume

Ein Binärbaum ist definiert, dass jeder Knoten höchstens zwei Kinderknoten haben kann. Ein **gewichteter Binärbaum** bekommt wird bereits beim Aufbau sortiert. Das führt zu Suchbäumen.

Traversierung von Binärbäumen Eine rekursive Methode um alle Knoten eines Binärbaumes auszugeben. Es gibt eine ganze Reihe von verschiedenen Methoden:

1. Pre-Order:
   Zuerst der Knoten selbst, dann der linke, dann der rechte Teilbaum.
2. Post-Order:
   Erst der linke, der rechte Teilbaum und zum Schluss der Knoten.
3. In-Order:
   Links, Knoten, Rechts. Diese Methode führt bei einem gewichteten Suchbaum zur Ausgabe der richtigen Reihenfolge.
4. Reverse In-Order:
   Rechter Teilbaum, Knoten, linker Teilbaum.
5. Level-Order
   Besuchen der Ebenen von Links nach Rechts.

4.2 Suchbäume

Ein Suchbaum ist ein Binärbaum mit der Eigenschaft, dass alle Markierungen im linken Teilbaum kleiner sind als im Knoten, alle im rechten Teilbaum größer.
Die Suche funktioniert genauso. Ein Knoten wird verglichen.

- Wenn der Knoten den gesuchten Wert hat, so gibt er sich selbst zurück.
- Sollte der gesuchte Wert geringer sein, wird die Suche an den linken Teilbaum übergeben.
- Sollte der Wert größer als der des Knoten sein, dann wird die Suche an den rechten Teilbaum übergeben.

5 Testen & Debuggen

- Syntaxfehler
  Der Compiler kann das Programm nicht übersetzten, weil es einen formalen Fehler hat. Dabei kann es sich um einfache Rechtschreibfehler, falsch gesetzte Klammern oder andere Dinge handeln.

- Laufzeitfehler
  Während der Ausführung passiert nichts oder es gibt eine Fehlermeldung. Das Programm wartet auf die Antwort aus anderen Quellen oder befindet sich in einer Endlosschleife

- Logische Fehler
  Das Programm funktioniert, allerdings sind die Ausgaben und Aktionen andere als erwartet.

5.1 Debugging

Debugging bedeutet einen Softwarefehler zu finden und zu beheben. Das bedarf zwei Schritten, dem finden und dann das ausbessern. Das Finden kann durch den Computer unterstützt werden und macht die Suche so deutlich einfacher. Es gibt verschiedene Möglichkeiten um sich auf die Suche nach einem Bug zu machen.

- von Hand
  zum Beispiel mit Pythontutor

- Print-Funktion
  Falls kein Debugger zur Verfügung steht, ist es sinnvoll an den richtigen Stellen eine Print Methode zum Ausdruck von Variablen zu geben.

- Ein Debugger-Programm
  Diese gibt es in den meisten Programmieroberflächen.

- Beobachten von internen Werten

- Aus den Rückgaben des Programmes Schlüsse ziehen

- logging
  dabei werden prints generell im Code integriert und mit Schaltern an und wieder abgestellt.

5.2 Debugging generell

Post-Mortem-Tools werden eingesetzt, wenn das Programm abgekratzt hat. Dabei wird der Programmzustand nach dem Fehler untersucht. Wichtige Wekrzeuge sind Stack Backtrace und die Variablenbelegung.

Interaktive Debugger werden mit der IDE mitgelieftert (meistens). Setzten Breakpoints, mit denen man an denen die Variablen überprüfen kann. Es gibt auch einzelschrittausführung mit der man sich Schritt für Schritt durch das Programm arbeiten kann.

5.3 Automatische Tests

Testen eines Programmes heißt, fehlerhaftes Verhalten zu provozieren. Die Idee ist, einem Programm Grenzfälle vorzulegen um sicher zu gehen, dass es mit der gewünschten Aufgabe klarkommt. Kleines Handbuch:

1. Schon vor dem Programmieren: Grenzfälle überlegen
2. beim Programmieren: Tests, die zu einem Fehler geführt haben müssen gespeichert und wieder verwendet werden
3. Nach dem Programmieren beim Debuggen: Wiederholung der Tests um Sicherzustellen, dass Bugfixes keine alten und neuen Fehler hervorruft.

Testgetriebene Entwicklung bedeutet, dass der Fortschritt eines Programmes anhand der bestandenen vorherdeffinierten Tests gemessen wird.

5.4 Fehlerfreies Programmieren

Fehlerfreies Programmieren ist so gut wie nicht möglich, da sich die Fehlerquellen überall verstecken können.

6 Rekursion

Rekursion ist das sichselbt wieder aufrufen einer Funktion.

6.1 Endrekursion

Endrekursion bedeutet, dass der Rekursionsaufruf in der letzten Zeile zusammen mit dem Return aufgerufen wird.
Diese Form der Rekursion ist nicht ideal. Um sie zu umgehen, können endrekursive Funktionen durch while-Schleifen implementiert werden. Die endrekursiven Aufrufe werden zu Zuweisungen an die Parameter. Die Abbruchbedingung der Rekursion wird negiert zur Bedingung der while-Schleife.
Vorher (Mit Endrekursion):

```
def search (tree , item ):
      if tree is None :
m return False
      elif tree . mark == item :
           return True
      elif tree . mark < item :
           return search ( tree.left , item )
      else :
           return search ( tree.right , item )
```

Danach(Iterativ:)

```
def search (tree , item ):
     while tree is not None :
          if tree . mark == item :
               return True
```

```
        elif tree . mark > item :
             tree = tree . left
        else :
             tree = tree . right
    else :
        return False
```

Das Umwandeln von allgemeiner Rekursion in ein iteratives Programm ist deutlich schwieriger als bei Endrekursion.

7 Objektorientierte Programmierung

Objektorienterte Programmierung ist eine Programmierungsidee, die versucht ein Programm durch die beteiligten Objekte zu modellieren. Dabei werden alle Zustände in Objekten gespeichert, und können nur durch Methoden aufgerufen und verändert werden.

Klassen sind in der Objektorientierten Programmierung der Bauplan der einzelnen Objekte und enthält die Definition der Attribute und der Methoden, die dem Objekt zur Verfügung stehen sollen.
Methoden eines Objektes können über die Punkt-Notation aufgerufen werden.

7.1 Vererbung

Klassen können auf andere Klassen als Vorlage oder besser gesagt Superklasse zugreifen. Ein Objekt der Unterklasse hat zu den eigenen Attributen und Methoden die selben Attribute und Methoden wie ein Objekt der Superklasse, allerdings kann eine Unterklasse Attribute und Methoden der Superklasse überschreiben.

Da sich in der Entwicklung die enzelnen Parameter einer Unterklasse ändern kann man in der init-Methode auch einfach einen *keyword-Parameter* übergeben. Dieser wird als $**kwargs$ bezeichnet

7.2 Weitere Begriffe

Aggregierung Oft sind Objekte aus anderen Objekten zusammengesetz (Attribute sind auch Objekte). Methodenaufrufe an ein Objekt führen dann zu Methodenaufrufen auf eingebettete Objekte.

Properties Sollen den Zugriff auf Attribute von Objekten in der Objektorientierten Programmierung in Python beschränken. In vielen Programmiersprachen (Java) können Attribute als privat deklariert werden, mit dem Sinn, das ein Objekt gekapselt ist. In Python sind Attribute im wesentlichen öffentlich und können/sollten/müssen durch Getter und Setter als **Properties** geschützt werden.

Dateninvariante ist eine logische Aussage über die Atrribute eines Objekts, die während des gesamten Lebensdauer des Objektes erfüllt sein muss. (keine Variation, steht ja auch so im Namen) Damit eine Invariante nicht verändert werden kann, sollte vor der eigentlichen Änderung

die Eingabe durch eine **Exeption** überprüft werden. das wird über eine **assert**-Funktion aufgerufen.

assert radius > 0 , „Radius sollte größer als Null sein “.

Datenkapselung verhindert eben den Zugriff auf Attribute von außen. Interaktion mit einem Objekt geschieht nur durch Methoden.

Regeln bei Invariante

1. Jede Invariante **muss** im docstring der Klasse dokumentiert sein.
2. Der Konstruktor muss die Einhaltung der Invariante prüfen
3. Das Attribut muss als Property ohne Setter definiert werden.

Der Attributwert für den Radius wird im Feld __radius gespeichert und sind damit nicht direkt aufrufbar. „radius„ist nun eine getter Methode (c.radius Siehe beispiel auf den Folien)

7.3 Operatorüberladung

Ein Operator ist überladen, wenn er je nach Typ und Argument unterschiedlich definiert ist. (Ein Art mehrfachbelegung.) Dabei ist natürlich vorsicht geboten! Falls ein Operator überladen ist, ist nicht immer ersichtlich, welcher Code ausgeführt wird.

7.4 Magische Methoden

Methoden, die mit zwei Unterstrichen beginnen heißen „magisch“. __init__ zum Beispiel. Es gibt drei Arten von magische Methoden:

- Allgemeine Methoden:
  - Konstruktion/Destruktion
  - Vergleich und Hashing
  - String-Kinversion
  - Verwendung einer Instanz als Funktion
  - Attributzugriff
  - Magischie Attribute
  - Vergleich
- Numerische Methoden
- Container Methoden

8 Dictionaries

Dicionaries Sind eine Abbildung von Schlüsseln auf Werte.
Grundoperationen:

- Einfügen von Schlüsseln
- Entfernen eines Schlüssels

- Nachschlagen eines Wertes zu einem Schlüssel
- Anwesenheit eines Schlüssels

Das bringt natürlich verschiedene Vorraussetztungen für die Schlüssel mit sich:

1. Die Schlüssel müssen auf Gleichheit gestestet werden
2. Die Schlüssel müssen unveränderlich (immutable) sein

Dictonaries sind so implementiert, dass der Wert zu einem gegebenen Schlüssel sehr effizient und unabhöngig von der Anzahl der bestehnden Einträge bestimmt werden kann. Im Gegensatz zu Sequenzen sind Dictionaries ungeordnet.
Dictionaries können auf verschiedene Weisen erzeugt werden:

- `{key1: value1, key2: value2}`
  Hier sind „keys“unveränderliche Python-Objekte. Für die Werte in den Schlüsseln kann alles genommen werden, was gespeicher werden.
- `dict ( key1=value1, key2=value2)`
  Hier sind die Schlüssel Variablennamen, die vom `dict`-Konstruktor in Strings konvertiert werden.

Dateien können in Hashtabellen sicher gespeichert werden. Eine Hashtabelle weißt jedem Schlüssel einen (durch schiere Masse an Möglichkeiten) Hashwert zu. Dieser wird bei Hashtabellen als Speicherzelle genutzt. In anderen Anwendungen kann er auch als Authentification dienen. Schlüssel **müssen** hashbar sein. Das bedeutet, dass die Hashfunktion damit ein eindeutiges Feld ausrechnen können muss. Eine Hashtabelle hat darüber hinaus auch keine spezielle Ordnung der Elemente, was dazu führt, dass die Ausgabe der Tabelle nicht geordnet ist.

Schlüssel dürfen nicht verändert werden! Erlaubte Schlüssel sind nur Tupel, Strings und Zahlen. So lange das Tupel keine veränderbaren Objekte enthält. Verboten sind Listen, Dictionaries und Ojekte, die Listen und/oder Dictionaries enthalten.

9 Mengen

Mengen sind eine endliche Zusammenfassung von Elementen.Mengen sind einzigartig, weil sie wie Dictionaries sind. Die Elemente müssen auf Gleichheit getestet werden und sind unveränderlich. Eine Menge kann also nicht dasselbe Element mehrmals beinhalten.
Mengen haben auch einige Funktionen. Das wichtigste ist, dass `setS` veränderliche Strukturen schafft, während `frozensetS` ist das nicht möglich.

10 Ausnahmen

Exeptions sind Ausnahmen. Entweder melden sie ein Prblem mit dem Programm oder man verwendet sie als „Rückgabewert “. Das Auslösen einer Exeption löst einen Programmabbruch aus. Damit werden alle Schleifen, Vergleiche und Methoden ignoriert und es wird nach einem Code gesucht, der sich um die Fehlermeldung kümmert ohne das Programm zu runinieren. Mit einem `try-exept` werden in einem `try`-Teil ein Code ausgeführt. Falls ein Fehler auftritt, wird der code beendet und landet im `exept`-Teil. Da wird dann entschiede, was mit dem Fehler gemacht

werden soll. Sollte im `exept`-Teil auch keine Behandlung gegeben sein, dann kann man alle sonstigen Fehler mit der `raise`-Methode in den nächst höheren `exept`-Teil weiterreichen. (bedeutet, sollte eine Methode von einer anderen Methode aufgerufen worden sein, oder ähnliches. Man kann/sollte einen `exept` druchaus noch auf einzelne Fehler zuspitzen und angepasste Reaktionen dazu schreiben. Ein unspezifisches `exept` ist nicht gern gesehen, da es alle Fehler abfängt.

11 Iteratoren

11.1 Generatoren

Generatoren sind quasi `for`-Schleifen, die bei jedem Druchlauf ein weiters Element zurück geben. Das funktioniert über die sog. `yield`-Funktion. Der Rumpf eines Generators wird mit `return` beendet. `yield from gen` entspricht:

```
for x in gen:
    yield x
```

Backtracing bedeutet, dass die Lösung Schritt für Schritt zusammengesetzt wird. Erweist sich eine Lösung als nicht korrekt, werden Schritte zurück genommen.

11.2 Iteratoren

Iteratoren sind eine verallgemeinerung von Generatoren. Die `for`-Schleife kann für viele Container-Objekte die Elemente durchlaufen. Man kann ein Objekt iterrierbarm achen, indem wir das *Iterator-Protokoll* implementieren. Dafür muss die magische Methode `__iter__` definiert werden, die einen Iterator zurück liefert. Jeder Iterator implementiert die magische Methode `__next__`, die das nächste Element liefert. Gibt es kein weiteres Element, so muss die Ausnahme `StopIteration` ausgelöst werden. Ein iterierbares Objekt erzeugt bei jedem aufruf von `iter()` einen neuen Iterator, besitzt aber keine `__next__`-Methode. Ein Iterator dagegen liefert sich selbst beim Aufruf von `iter()`, aber jeder Auffruf con `next()` ergibt ein neues Objekt aus dem Container. Da Iteratoren auch die `__iter__`-Methode besitzen, können Iteratoren an allen Stellen stehen, an denen ein iterierbares Objekt stehen kann.

12 Dateien

Dateien können von Programmen geöffnet werden. Dabei haben sie drei verschiedene Beretigungen:

1. „r “: Lesen der Datei
2. „w “: Schreiben in der Datei
3. „r+ “: Beides

13 Funktionale Programmierung

Es gibt verschiedene Konzepte der Programmierung. Imperative Programmierung beschreibt wie ein Computer etwas errechnen soll. Deklarative Prgrammierung beschreibt, was berechnet werden soll. Bisher haben wir uns nur mit Imperativer Programmierung beschäftigt. Doch Funktionale Programmierung ist eine Unterart der Deklarativen Programmierung.

13.1 Programmierparadigmen

- Keine explizite Berarbeitung eines Berechnungszustands.
- Logisch Programmierung beschreibt das Ziel durch logische Formeln.
- Funktionale Programmierung beschreibt das Ziel durch **mathematische Formeln**

Funktionen sind Bürger erster Klasse. Das bedeutet, dass Funktionen auch als Daten relevant sind.

Funktionen höherer Ordung haben Argumente oder Ergebnisse, die selbst wieder Funktionen sind.

Es gibt keine Schleifen. Nur **REKURSION**.
Es gibt keine Anweiseungen. Nur **AUSDRÜCKE**.

In rein Funktionalen Sprachen gibt es keine Zuweisungen und keine Seiteneffekte. Eine Variable bekommt zu beginn einen Wert, und diesen behält sie bis ans Ende des Programms.

referentielle Transparenz Eine Funktion gibt immer das gleiche Ergebnis bei den gleichen Argumenten.

13.2 Funktionale Programmierung in Python

Dazu gibt es nicht viel zu sagen. An sich funktioniert alles. Vorsicht bei Variablen. Am besten nur lokale Variablen benutzen, keine Globalen. Auch hat Python gewissen begrenzungen, was die Rekurusionstiefe angeht, das haben funktionale Programmiersprachen nicht.

13.3 Funktionsdefinition

Funktionen sind der gleiche Rotz wie vorher auch.

13.4 Lambda-Notation

`lambda`-**Operator** definiert eine kurze, namenlose Funktion, deren Rumpf durch einen Ausdruck gegeben ist. Die Idee ist, dass diese `lambda`-Funktion nur einer Stelle benutzt wird, und deshalb nicht wichtig genung ist um ihr einen Namen zu geben.

13.5 `map`,`reduce`,`filter`

map ...

- ... hat zwei Argumente: Eine Funktion und ein iterierbares Objekt.
- ... wendet die Funktion auf jedes Element der Eingabe an und liefert die Funtionswerte als Iterator.
- ... kann auch mit einer k-stelligen Funktion und k weiteren Eingaben aufgerufen werden.
- Für jeden Funtionaufruf wird ein Argument von jeder der k Eingaben angefordert. Sie stopt, falls eine Eingabe keinen Wert mehr liefert.

filter erwartet als Argument eine Funktion mit einem Parameter und ein iterierbares Objekt. Es liefert einen Iterator zurück, der die Objekte aufzählt, bei denen die Funktion nicht „False “zurück gibt.

reduce wendet eine Funktion mit zwei Argumenten auf ein iterierbares Objekt und einen Startwert an. Der Startwert fungiert wie ein Akkumulator: bei jedem Iterationsschritt wird der STartwert durch *alter STartwert + nächster Iterationswert* ersetzt. Am Ende ist der Startwert das Ergebnis. Falls kein Startwert vorhanden ist, verwende das erste Element der Iteration.

13.6 Comprehension

Comprehension (zu deutsch Abstraktion) kann Listen deklarativ und kompakt beschreiben. Es ist entlehnt aus der funktionalen Programmiersprache Haskell. Ähnlich der mathematischen Mengenschreibweise: $\{x \in U | \phi(X)\}$ (alle x aus U, die die Bedingung ϕ erfüllen). Comprehensions lassen sich auch für Dictionaries Mengen und alles andere auch verwedet werden.

13.7 Geschachtelte Funktionsdefintionen

Ein Namensraum ist eine Abbildung von Namen auf Werte (in Python oft durch ein `dict` realisiert.) Innerhalb von Python gibt es:

- den `__builtins__`-Namenraum mit allen vordefinieurten Funktionen und Variablen.
- den Namensraum von Modulen, die importiert werden.
- den globalen Namensraum
- den lokalen Namensraum innerhalb einer Funktion
- Namensräume haben verschiedene Lebensdauern; der lokale Namensraum einer Funktion exisitert nur während ihrer Ausführung.
- auf gleiche Variablennamen in verschiedenen Namensräumen kann oft mit der Punkt-Notatio zugegriffen werden.

Scope ist der Gültigkeitsbereich einer Variablen. Damit wird der textuelle Bereich in einem Programm, in dem die Variable ohne die Punkt-Notation referenziert werden kann. Die Hierarchie ist, dass der innere den äußeren Bereich überbietet.

1. lokaler Bereich
2. nicht-lokaler Bereich
3. globaler Bereich
4. builtin-Namensraum

Gibt es eine Zuweisung im akutellen Scope, so wied von einem lokalen Namen ausgegangen. `global` bzw. `nonlocal`. Wenn ein Name nicht gefunden wird, dann gibt es eine Fehlermeldung.

14 Closures

Sind Definitionen in einer Definition um nochmal Sachen zu machen.