

Programmierung mit Python für Programmierer

Kapitel 1 - Einführung und Datentypen

Autor: Dr. Christian Heckler

Vorbemerkungen

- Verwendete Literatur: siehe Referenzen
- Verwendete Symbole:
 - **!**: Beispielprogramm
 - **i**: Weitere Erläuterungen im Kurs
 - **?**: Übung

Vorraussetzungen und Ziele

- Voraussetzungen:
 - Erfahrungen in einer höheren Programmiersprache
 - Die folgenden Konzepte werden als bekannt vorausgesetzt
 - Bezeichner / Variablen / Zuweisungen
 - Datentypen: Zahlen, Zeichenketten (Strings), Kollektionen
 - Kontrollstrukturen: Verzweigungen, Schleifen
 - Prozeduren, Funktionen, Module
 - Objektorientierung (?)
- Ziel: Kompakte Einführung in die Grundlagen, so dass die Erarbeitung der Details und Module im Selbststudium möglich ist, z.B. über die genannten Bücher und die Dokumentation (<https://docs.python.org>)

Kursinhalt

- Variablen und „Laufzeitmodell“
- Eingebaute Datentypen
- Kontrollstrukturen
- Funktionen, Module und Pakete
- Objektorientierung

Eigenschaften von Python

- Interpretierte Skriptsprache
- Dynamisch getypt
- Verschiedene Programmierparadigmen (prozedural, OO, funktional, ...)
- Für alle gängigen Plattformen verfügbar (Windows, Unix, Mac)
- Insbesondere für Anfänger geeignet
- Umfangreiche Standardbibliothek
- Entwickelt von Guido van Rossum am CWI Amsterdam (Centrum Wiskunde & Informatica) ab 1989
- Weite Verbreitung (s. diverse Programmiersprachenranglisten)

Versionen von Python

- Es gibt verschiedene Versionen von Python
- Aktuell:
 - Python 2.x
 - Python 3.x
- Python 2 und Python 3 sind nicht (hunderprozentig) kompatibel
- Im Kurs wird Python 3 verwendet
- Der Interpreter für python 3 kann „python3“ (Linux, Apple) heißen.
- Die Version erhält man über den Aufruf

```
python -V bzw. python3 -V
```

Python-Interpreter

- Der Python-Interpreter kann ein Python-Programm ausführen, das in einer Textdatei (mit der Endung .py) abgespeichert ist.
- Es gibt aber auch einen interaktiven Modus, in dem Python-Anweisungen ausgeführt werden können, ohne ein Programm zu schreiben.

Python-Programme

- Erstellung mit beliebigem Texteditor
- Zeichencodierung UTF-8: wird von Python standardmäßig angenommen – alternativ erste Zeile im Quelltext mit der tatsächlichen Codierung

```
# -*- coding: utf-8 -*-
```

- Einheitliche Einrückung (Tabs oder Leerzeichen)
- Z.B.: Geany (www.geany.org), thonny (thonny.org), Visual Studio Code
- Es gibt auch IDEs, z.B. PyCharme, Eclipse-Plugin, IDLE, ...
- Programmausführung
 - aus dem Editor / Entwicklungsumgebung (wenn unterstützt)
 - Doppelklick, falls Dateiendung „.py“ mit Python-Interpr. verk.
 - Kommandozeile: `python prg.py` bzw. `python3 prg.py`
 - Unix: Shebang: `#!/usr/bin/env python3`

Interaktiver Modus


- Der Interpreter kann auch im interaktiven Modus aufgerufen werden:
 - Kommandozeile: `python` (bzw. `python3` unter Unix)
 - Über das Windows-Startmenu
 - Evtl. aus der Entwicklungsumgebung
- Der Interpreter meldet sich mit dem „Prompt“: `>>>`
- Jetzt können Anweisungen eingegeben werden, die mit der Return-Taste abgeschlossen werden, z.B.:

```
>>> print("Hallo")  
Hallo
```

❓ Übung - Erstes Programm und interaktive Shell

- Erstellen Sie ein Programm, das `Hallo Welt` ausgibt.
- Geben Sie `Hallo Welt` in der interaktiven Shell aus.

Einführendes Beispiel

-  zahlenraten.py

Kommentare

- `#` Kommentar bis zum Zeilenende
- Blockkommentar:

```
'''Drei einfache Anführungszeichen  
für Kommentare über mehrere Zeilen.  
Streng genommen ist das kein Kommentar,  
sondern ein String über mehrere Zeilen,  
der aber als Kommentar verwendet werden kann.  
'''
```

Anweisungen

- normal einzeilig
- ohne Abschluss mit Strichpunkt o.ä.
- Mehrere Anweisungen in einer Zeile durch Strichpunkt getrennt möglich (aber unüblich):

```
a=1; b=2
```

- Mehrzeilig:
 - erlaubt, wenn Anfang und Ende eindeutig (z.B. bei Parameteraufzählung bei Prozeduren)
 - Explizit durch \ :

```
a=\n4711
```

Blöcke und Hilfe

- Anweisungsblöcke werden durch einheitliches Einrücken gebildet.



Keine Vermischung von Leer- und Tabulatorzeichen. Empfehlung:
Nur Leerzeichen verwenden!

- Hilfe: In der Shell:

- `help()`
- `help(str)`

Variablen

- Variablen müssen nicht deklariert werden.
- Eine Variable wird angelegt, wenn ihr zum ersten mal ein Wert zugewiesen wird.
- Sind ungetypt, d.h. eine Variable kann Objekte verschiedenen Typs zugewiesen bekommen.
- Die Objekte selbst haben einen festen Typ.
- Zuweisungsoperator: =
- Beispiel:

```
a = 3  
a = "Eine Zeichenkette (String)"
```

Variablen 2

- Gültige Variablennamen („Bezeichner“) bestehen aus:
 - Buchstaben (groß und klein)
 - Unterstrich `_`
 - Ziffern (`0 - 9`) (aber nicht am Anfang)
 - Unicode-Zeichen (Standardkodierung ab Python 3 ist UTF-8)
- beliebige Länge
- Groß- und Kleinschreibung wird unterschieden
- Python-Schlüsselwörter dürfen nicht verwendet werden.
- Konvention:
 - Kleinschreibung
 - `meine_variable`, `alter_in_jahren`

Laufzeitmodell

- Es gibt nur Referenzdatentypen („alles ist ein Objekt“).
- Bei der Zuordnung eines Wertes zu einer Variablen wird für die Variable eine Referenz auf das entsprechende Objekt angelegt.
- Jedes Objekt hat ⓘ
 - eine Identität (abzufragen über `id()`),
 - einen Typ (`int` , `float` , `str` , ...)
 - einen Wert.

Gleichheit vs. Identität

- Bei Test auf Gleichheit (`a == b`) wird
 - eine entsprechende Methode der Klasse aufgerufen (vgl. `equals` bei Java), falls eine solche Methode implementiert wurde;
 - andernfalls werden die Identitäten verglichen: `id(a) == id(b)` (analog zu `==` bei Java).
- Bei den Standardtypen ist `==` implementiert.
- Mit `is` wird auf Identität abgefragt, d.h. `a is b` entspricht `id(a) == id(b)`

Beispiel

```
a=1234  
b=a
```

Danach zeigen `a` und `b` auf dasselbe Objekt `1234` vom Typ `int`. Es gilt also:

- `a is b`
- und somit natürlich auch `a==b`

```
b=1234
```

Jetzt gilt:

- `a is not b` (`a` und `b` zeigen auf unterschiedliche Objekte)
- `a == b`: Der Gleichheitsoperator für die Klasse `int` ist entsprechend implementiert.

type und isinstance

- Der Typ eines Objektes läßt sich mit `type()` bestimmen.
- Eine Abfrage auf den Typ ist mit `instanceof` möglich:
 - `isinstance(a, int)`
 - `isinstance(a, (int, float))`
- `isinstance()` berücksichtigt dabei auch die Ableitungshierarchie, d.h. wenn `c2` von `c1` abgeleitet ist, und `a` ein Objekt vom Typ `c2` referenziert, dann gilt: `isinstance(a, c1) == True`

Eingebaute Datentypen

- Sind direkt (ohne Import) nutzbar.
- Werden syntaktisch unterstützt, z.B.: Dictionaries:

```
d = {}  
d["4711"] = "Wasser"
```

- Eingebaute Datentypen:
 - NoneType
 - Zahlen (ganze Zahlen, Gleitkommazahlen, komplexe Zahlen)
 - Zeichenketten („Strings“)
 - Wahrheitswerte
 - Kollektionen
 - Nicht: Zeit und Datum (dafür gibt es aber Module)

Veränderliche und unveränderliche Datentypen

- „Mutable“, „immutable“
- Unveränderlich:
 - Ein Objekt kann nach der Erzeugung nicht mehr verändert werden.
 - Braucht man einen neuen Wert, wird ein neues Objekt erzeugt.
 - Bsp: Zeichenketten (Strings analog zu Java)
 - Vorteil:
 - Keine Seiteneffekte, wenn mehrere Referenzen auf ein Objekt zeigen.
 - Damit mehrfache Verwendung eines Objektes möglich.

Veränderlich vs. unveränderlich

- Veränderliche Datentypen:
 - Ein Objekt kann verändert werden.
 - Beispiel: Liste
 - Vorteil: Operationen sind schneller und verbrauchen im Zweifelsfall weniger Speicher
- Bsp: `ver_vs_unver.py` !

Zahlen

- unveränderlich
- Ganze Zahlen (`int` für Integer):
 - können beliebig groß werden (Beschränkung durch Speicher)
 - Division:
 - eine Division mit `/` ergibt immer eine Gleitkommazahl
 - Ganzzahlige Division `//`
 - Rest bei der ganzzahligen Division: `%`
 - Literale:
 - binär: `x = 0b1000`
 - oktal: `x = 0o7000`
 - hexadezimal: `x = 0xf000`
 - Umwandlung in String: `bin(x)`, `oct(x)`, `hex(x)`

Zahlen II

- Gleitkommazahlen (`float` für floating point)
- begrenzte Genauigkeit und begrenzte Größe (siehe `sys.float_info`)
- Literale:
 - `v = 3.14`
 - `v = 3.14e-12`
 - Spezielle Werte: `inf`, `-inf`, `nan`, siehe `gleitkomma.py` !
- Komplexe Zahlen
- Operatoren auf Zahlen:
 - `+`, `-`, `,`, `/`, `//`, `%`, `abs()`, `*`
 - s. auch https://www.python-kurs.eu/python3_operatoren.php

NoneType

- Es gibt nur ein Objekt von diesem Typ: `None`
- Repräsentiert das Nichts.
- Bsp: Unterscheidung zwischen einer noch nicht verwendeten Variablen und einer Var., die (noch) keinen (gültigen) Wert hat.
- Bsp: Liste von Verbrauchsdaten

```
verbrauch_HJ1 = [20.1, 34.5, None, 22.4, 23.5, 31.3]
```

- Unterschied zu `null` bei anderen Programmiersprachen, z.B. Java, C++:
 - Bei `a = null` hat die Referenz keinen Wert (`a` verweist auf kein Objekt). Die Var. `a` kann von beliebigem (Referenz-) Typ sein.
 - Bei `a = None` verweist `a` auf das `None`-Objekt (vom Typ `NoneType`).

Wahrheitswerte (bool)

- Zwei Werte: `True`, `False`
- Operatoren:
 - `and`
 - `or`
 - `not`
- Bsp:

```
b1 = True
b2 = False
b3 = b1 or b2
print(b3)
```

Wahrheitswerte II

- „Kurzschlussauswertung“ boolescher Ausdrücke:
 - Auswertung von links nach rechts
 - Auswertung wird beendet, wenn das Ergebnis feststeht
 - Anwendungsbeispiele:
 - Auswertung des weiteren Ausdrucks nicht möglich

```
x != None and x > 100
```

- Auswertung des weiteren Ausdrucks aufwändig:

```
x == 5 and aufwaendige_funktion(x)
```

- Mehrfachvergleiche: `5 <= x <= 10`

Einschub: Binäre Operatoren

- Neben den logischen Operatoren `and`, `or`, `not` gibt es auch binäre Operatoren.
- Diese arbeiten auf der Binärdarstellung **ganzer Zahlen**.
- Binäre Operatoren:
 - Und: `&`
 - Oder: `|`
 - Nicht: `~`
 - Schieben nach links: `<<`
 - Schieben nach rechts: `>>`
- Bsp: `9 & 3` ergibt `1`

Zeichenketten

- Bsp: `s = "Hallo Python"` oder `s = 'Hallo Python'`
- Sequenz (Folge) von einzelnen Zeichen („Character“)
- Zeichen: Beliebiges Unicode-Zeichen
- Standardmäßige Kodierung UTF-8
- Verwendung von Unicode-Escape-Sequenzen, z.B. `s = "Saarbr\u00fccken"`
- Die üblichen Esc-Sequenzen gibt es auch, z.B. `\n`
- Keine Ersetzung von Sonderzeichen: `s = r"Hallo\n"`
- Strings über mehrere Zeilen:
 - Trennung über `\` (keine neue Zeile)
 - Dreifache Anführungszeichen (neue Zeile)

Zeichenketten: Länge und Indizierung

- Die Länge des Strings ist die Anzahl der Zeichen.
- Zugriff auf ein Zeichen über dessen Index.



Die Indizierung beginnt (wie in der Informatik üblich) mit 0

- Beispiel: `s = "Python"`
 - `len(s)`
 - `s[1]`
 - `s[-1]`
- Der Datentyp `string` ist **unveränderlich**, d.h. Änderungen der Art `s[0]='J'` sind nicht möglich.

Zeichenketten: Operatoren und Funktionen

- Vergleich: `==`, `<`, `<=`, `>`, `>=`
- `+` : Konkatination
- `*` : Wiederholung
- `len` : Länge
- Indizierung (`[]` -Operator)
- Ausschneiden („Slicing“):
 - `"Python"[2:4]`
 - `"Python"[:4]`
 - `"Python"[2:]`
 - `"Python"[:]`



Der letzte Index ist exklusiv (im Bsp. ist das 4. Zeichen - bei Zählung von 0 an - nicht inkludiert).

Datentyp bytes

- Analog Strings, aber Sequenz von **Bytes**.
- Kann für Binärdaten verwendet werden.
- Beispiel: `b"Hallo"`

Listen

- Sequenz von Objekten beliebigen Typs.
- Erzeugung einer „leeren“ Liste: `l = []`
- Da beliebige Objekte erlaubt sind, sind also auch „verschachtelte“ Listen möglich.
- Beispiele:

```
l1 = [1, 2.2, "Text"]  
l2 = [7, [3,4], 7]
```

- Länge und Indizierung wie bei Strings.
- Mehrfachindizierung: `l2[1][0]`

Listen 2

- Bei dem Datentyp `list` handelt es sich um einen **veränderlichen** Datentypen.

```
l1 = [1,2,3]
l2 = l1
l1[0] = 66
```



Da es sich um einen veränderlichen Datentyp handelt, sind Seiteneffekte möglich. ⓘ

- Slicing funktioniert wie bei Zeichenketten (`l1[1:2]`).
- Beim Slicing wird eine Kopie des Listenausschnitts zurückgegeben
 - somit sind hier keine Seiteneffekte zu befürchten
 - `l[:]` liefert also eine Kopie der Liste

Operationen auf Listen

- Test, ob `element` in `liste` enthalten: `element in liste`
- Element zu Liste hinzufügen: `l.append(7)`
- An einer bestimmten Stelle: `l.insert(index, element)`
- Element löschen: `x.remove(7)`
- „Flache“ Kopie: `l.copy()`
- `pop(i)` : Gibt das `i`-te Element zurück und entfernt es
- `pop()` : Gibt das letzte Element zurück und entfernt es
- Siehe auch:
 - <https://docs.python.org/3.6/library/stdtypes.html#lists>
 - https://www.python-kurs.eu/python3_sequentielle_datentypen.php

Listen zusammenfügen

- `l1.extend(l2)` : Die Liste `l1` wird um alle Elemente der Liste `l2` erweitert.
- `l3 = l1 + l2` : Es wird eine neue Liste mit den Elementen von `l1` und `l2` erzeugt. Diese wird der Variablen `l3` zugewiesen.
- `l1 += l2` : Funktioniert also, ist aber gegenüber `extend` ineffizient, da erst eine neue Liste mit allen Elementen erzeugt wird, die dann der Variablen `l1` zugewiesen wird.

Tupel

- analog Listen; ein Tupel ist aber **unveränderlich**.
- Erzeugung über `()` statt `[]`, z.B.:

```
adr = ("Max", "Mustermann", "Irgendwogeg", "Lummerland")
```

- Leeres Tupel: `t = ()`
- Tupel mit einem Element: `t = (1,)`
- Packing: `t = 1, 2`
- Unpacking: `x, y = t`
- Haben keine Kopier-Methode (warum auch, sind ja unveränderlich)
- Anwendungsbeispiel: Ähnlich Rekords bei anderen Programmiersprachen.

Dictionaries

- Schlüssel-Wert-Paare (Maps)
- Bsp:

```
leer = {}  
en_de = {"red": "rot", "green": "grün"}  
verbr = {"Januar": 32.1, "Februar": 21.3, "März": 48.7}
```

- Zugriff, Änderung eines Wertes, Einfügen eines Wertes

```
verbr["Januar"]  
verbr["Januar"] = 34.7  
verbr["April"] = 17
```

- Ein Dictionary ist also **veränderlich**.
- Die Schlüssel-Wert-Paare haben keine Ordnung.
- Prüfung, ob Element vorhanden ist: "April" in verbr

Dictionaries: Werte

- Erlaubte Werte („Einträge“): Beliebige Datentypen
- Auch ein Dictionary kann also Wert eines Eintrages sein („verschachtelte Dictionaries“)

Dictionaries: Schlüssel

- Damit ein Objekt als Schlüssel eines Dictionaries verwendet werden kann, muss die zugehörige Klasse „hashable“ sein, d.h. eine Hash-Methode implementieren.
- Eingebaute Datentypen:
 - Alle unveränderlichen Typen (Zahlen, Tupel, Strings) können verwendet werden.
 - Die veränderlichen Typen (list, dict) können nicht verwendet werden.
- Eigene Klassen (s. Kapitel OO):
 - Haben standardmäßig eine Hash-Funktion.
 - Können also verwendet werden.
 - Die Hash- und Equal-Methode sollten sinnvoll überschrieben werden (vgl. Java).

Operationen auf Dictionaries

- `d.clear()`
- `d.copy()` : Gibt „flache“ Kopie zurück
- `d.items()`
- `d.keys()`
- Siehe auch:
 - <https://docs.python.org/3.6/library/stdtypes.html#mapping-types-dict>
 - https://www.python-kurs.eu/python3_dictionaries.php

Mengen

- Eine Menge ist eine ungeordnete Sammlung von einmaligen und unveränderlichen Elementen
- Für die erlaubten Elementtypen gilt dasselbe wie für Schlüssel bei Dictionaries („Hashable“).
- Erzeugung einer Menge:
 - `s = set()`
 - `s = set('Hamburg', 'München', 'Düsseldorf')`
 - `s = {'Hamburg', 'München', 'Düsseldorf'}`
 - `s = {}` hingegen erzeugt ein leeres Dictionary
- Elemente hinzufügen: `s.add("Saarbrücken")`

Mengen II

- Element löschen: `s.remove('Hamburg')`
- Mengen sind also **veränderlich**
- `frozenset(s)` liefert unveränderliche Menge, die dann auch ihrerseits Element einer Menge sein kann.
- Weitere Operationen:
 - `s.clear()`
 - `s.copy()` : gibt „flache“ Kopie zurück
 - `s1.difference(s2)`
- s. auch:
 - https://www.python-kurs.eu/python3_sets_mengen.php
 - <https://docs.python.org/3.6/library/stdtypes.html#set-types-set-frozenset>

Referenzen

- [Ste] Ralph Steyer: Programmierung Grundlagen, Herdt-Verlag
- [Kle] Bernd Klein: Einführung in Python 3, Hanser-Verlag
- [Kof] Kofler: Python – Der Grundkurs, Rheinwerk Computing
- [EK] Johannes Ernesti, Peter Kaiser: Python 3 – Das umfassende Handbuch, Rheinwerk Computing
- [Mat] Eric Matthes: Python Crashkurs – Eine praktische, projektbasierte Programmierereinführung, dpunkt.verlag
- [Swe] Sweigart: Eigene Spiele programmieren: Python lernen