

# 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 (hundertprozentig) 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](http://www.geany.org)), thonny ([thonny.org](http://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`

# 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