Informatik I: Einführung in die Programmierung

2. Erste Schritte in Python

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Prof. Dr. Peter Thiemann

15. October 2025

UNI FREIBURG

Allgemeines

Programmiersprachen

UNI FREIBURG

Ada, Basic, C, C++, C♯, Cobol, Curry, F♯, Fortran, Go, Gödel, HAL, Haskell, Java, JavaScript, Kotlin, Lisp, Lua, Mercury, Miranda, ML, OCaml, Pascal, Perl, PHP, Python, Prolog, R, Ruby, Scheme, Shakespeare, Smalltalk, Swift, TypeScript, Visual Basic, u.v.m.

Programmiersprachen

UNI FREIBURG

Ada, Basic, C, C++, C♯, Cobol, Curry, F♯, Fortran, Go, Gödel, HAL, Haskell, Java, JavaScript, Kotlin, Lisp, Lua, Mercury, Miranda, ML, OCaml, Pascal, Perl, PHP, Python, Prolog, R, Ruby, Scheme, Shakespeare, Smalltalk, Swift, TypeScript, Visual Basic, u.v.m.

Wir verwenden Python (genauer Python 3), eine

- objektorientierte,
- dynamisch getypte,
- interpretierte und interaktive
- höhere Programmiersprache.

Die Programmiersprache Python …

- Anfang der 90er Jahre als Skriptsprache für das verteilte Betriebssystem Amoeba entwickelt;
- gilt als einfach zu erlernen;
- wurde kontinuierlich von Guido van Rossum bei Google (seit 2013 Dropbox; seit 2020 Microsoft Distinguished Engineer: Python in Excel) weiterentwickelt.
- bezieht sich auf die Komikertruppe *Monty Python*.

Guido van Rossum (Foto: Wikipedia)

Literatur

Hier eine Auswahl von Lehrbüchern zu Python3.

- Allen Downey, *Think Python: How to Think Like a Computer Scientist*, O'Reilly, 2nd edition, 2015
- als PDF herunterladbar oder als HTML lesbar (Green Tea Press): `https://greenteapress.com/wp/think-python-2e/`
- als deutsche Version: Programmieren lernen mit Python, O'Reilly, 2013.
- Mark Lutz, *Learning Python*, O'Reilly, 2013 (deutsche Ausgabe ist veraltet!)
- Michael Weigend, *Python ge-packte Referenz*, mitp Verlag, 8. Auflage 2020 (als Nachschlagwerk, V3.8)
- Viele Videos und Online-Kurse
- Allgemeiner Hintergrund: Perdita Stevens, *How to Write Good Programs. A Guide for Students*, Cambridge University Press, 2020.

UNI FREIBURG

Warum Python?

Warum Python?

- Softwarequalität
  - Lesbarkeit
  - Wiederverwendbarkeit
- Programmierer-Produktivität
  - Python-Programme sind oft 50% kürzer als vergleichbare Java oder C++-Programme.
  - Kein Edit-Compile-Test-Zyklus, sondern direkte Tests
- Portabilität
- Support-Bibliotheken („Batterien sind enthalten“)

Wann besser nicht?

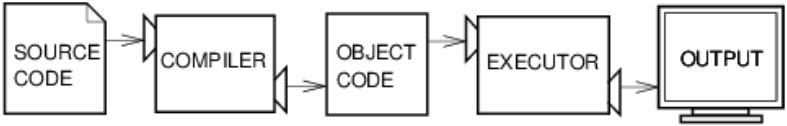
UNI FREIBURG

- Geschwindigkeit: Python ist „langsamer“ als Java, C++, Rust
- Wieviel langsamer?
  `https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html`
- Hardwarenähe: Nicht geeignet für Gerätetreiber, (kleine) Mikrocontroller
- Sicherheitskritische Anwendungen: medizinische Geräte, Luftfahrzeuge, Militär

Python-Interpreter

Interpreter- versus Compiler-Sprachen

Interpreter- versus Compiler-Sprachen



Abbildungen aus Downey 2013

Interaktiver und Skript-Modus

UNI FREIBURG

Der Python-Interpreter kann auf folgende Arten gestartet werden:

- im interaktiven Modus (ohne Angabe von Programm-Parametern)

Interaktiver und Skript-Modus

Der Python-Interpreter kann auf folgende Arten gestartet werden:

- im interaktiven Modus (ohne Angabe von Programm-Parametern)

→ Ausdrücke und Anweisungen können interaktiv eingegeben werden, der Interpreter wertet diese aus und druckt ggf. das Ergebnis.

Interaktiver und Skript-Modus

UNI FREIBURG

Der Python-Interpreter kann auf folgende Arten gestartet werden:

- im interaktiven Modus (ohne Angabe von Programm-Parametern)

→ Ausdrücke und Anweisungen können interaktiv eingegeben werden, der Interpreter wertet diese aus und druckt ggf. das Ergebnis.

- im Skript-Modus (unter Angabe einer Skript-/Programm-Datei)

Interaktiver und Skript-Modus

Der Python-Interpreter kann auf folgende Arten gestartet werden:

- im interaktiven Modus (ohne Angabe von Programm-Parametern)

→ Ausdrücke und Anweisungen können interaktiv eingegeben werden, der Interpreter wertet diese aus und druckt ggf. das Ergebnis.

- im Skript-Modus (unter Angabe einer Skript-/Programm-Datei)

→ Ein Programm (auch Skript genannt) wird eingelesen und dann ausgeführt.

UNI FREIBURG

Interaktives Nutzen der Shell

Erste Schritte: Ausdrücke

UNI FREIBURG

Nach Starten des Interpreters erscheint das Prompt-Zeichen. Der Interpreter wertet jeden eingegebenen Ausdruck aus und gibt das Ergebnis aus:

Python-Interpreter

```
>>>
```

Erste Schritte: Ausdrücke

Nach Starten des Interpreters erscheint das Prompt-Zeichen. Der Interpreter wertet jeden eingegebenen Ausdruck aus und gibt das Ergebnis aus:

Python-Interpreter

```
>>> 7 * 6
```

Erste Schritte: Ausdrücke

Nach Starten des Interpreters erscheint das Prompt-Zeichen. Der Interpreter wertet jeden eingegebenen Ausdruck aus und gibt das Ergebnis aus:

Python-Interpreter

```
>>> 7 * 6
42
>>>
```

Erste Schritte: Ausdrücke

UNI FREIBURG

Nach Starten des Interpreters erscheint das Prompt-Zeichen. Der Interpreter wertet jeden eingegebenen Ausdruck aus und gibt das Ergebnis aus:

Python-Interpreter

```
>>> 7 * 6
42
>>> "Hello world"
```

Erste Schritte: Ausdrücke

UNI FREIBURG

Nach Starten des Interpreters erscheint das Prompt-Zeichen. Der Interpreter wertet jeden eingegebenen Ausdruck aus und gibt das Ergebnis aus:

Python-Interpreter

```
>>> 7 * 6
42
>>> "Hello world"
'Hello world'
>>>
```

Erste Schritte: Ausdrücke

UNI FREIBURG

Nach Starten des Interpreters erscheint das Prompt-Zeichen. Der Interpreter wertet jeden eingegebenen Ausdruck aus und gibt das Ergebnis aus:

Python-Interpreter

```
>>> 7 * 6
42
>>> "Hello world"
'Hello world'
>>> "spam " * 4
```

Erste Schritte: Ausdrücke

UNI FREIBURG

Nach Starten des Interpreters erscheint das Prompt-Zeichen. Der Interpreter wertet jeden eingegebenen Ausdruck aus und gibt das Ergebnis aus:

Python-Interpreter

```
>>> 7 * 6
42
>>> "Hello world"
'Hello world'
>>> "spam " * 4
'spam spam spam spam '
```

Erste Schritte: „Drucken“ im Skriptmodus

UNI FREIBURG

Die `print`-Funktion gibt den Wert eines Ausdrucks aus: **?? PythonTeX ??**Liefert die Ausgabe: **?? PythonTeX ??**

Erste Schritte: „Drucken“ im Skriptmodus

UNI FREIBURG

Die `print`-Funktion gibt den Wert eines Ausdrucks aus: **?? PythonTeX ??**Liefert die Ausgabe: **?? PythonTeX ??** **?? PythonTeX ??**Liefert die Ausgabe: **?? PythonTeX ??**

Erste Schritte: „Drucken“ im Skriptmodus

Die `print`-Funktion gibt den Wert eines Ausdrucks aus: **?? PythonTeX ??**Liefert die Ausgabe: **?? PythonTeX ?? ?? PythonTeX ??**Liefert die Ausgabe: **?? PythonTeX ?? ?? PythonTeX ??**Liefert: **?? PythonTeX ??**

`print` ist der übliche Weg, Ausgaben zu erzeugen.

Etwas mehr zu `print`

Ein weiteres Detail zu `print`: **?? PythonTeX ??**Liefert: **?? PythonTeX ??**

- `print` kann mehrere Ausdrücke durch Kommas getrennt verarbeiten.
- Die Ergebnisse werden in derselben Zeile durch Leerzeichen getrennt ausgegeben.
- Ein Zeilenvorschub kann durch **??** in einen String eingefügt werden.

?? PythonTeX ??Liefert: **?? PythonTeX ??**

Rechnen

Zahlen

UNI FREIBURG

Python kennt mehrere Datentypen für Zahlen:

- `int` für ganze Zahlen;
- `float` für Gleitkommazahlen
  (eine verrückte Teilmenge der rationalen Zahlen);

int

UNI FREIBURG

Schreibweise für Konstanten vom Typ `int`:

Python-Interpreter

```
>>> 10
10
>>> -20
-20
```

int

UNI FREIBURG

Schreibweise für Konstanten vom Typ `int`:

Python-Interpreter

```
>>> 10
10
>>> -20
-20
```

Syntax

Die Schreibweise von Konstanten ist ein Aspekt der **Syntax** einer Programmiersprache. Sie beschreibt, welche Zeichen erlaubt sind, welche Worte vordefiniert sind und wie Sätze (Programme) in der Programmiersprache aussehen müssen.

Rechnen mit `int`

UNI FREIBURG

Python benutzt für Arithmetik die folgenden Symbole:

- Grundrechenarten: +, −, ∗ /
- Ganzzahlige Division: //
- Modulo: %
- Potenz: ∗∗

Rechnen mit `int`: Beispiele

UNI FREIBURG

Python-Interpreter

```
>>> 14 * 12 + 10
```

Rechnen mit `int`: Beispiele

UNI FREIBURG

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>>
```

Rechnen mit `int`: Beispiele

UNI FREIBURG

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 14 * (12 + 10)
```

Rechnen mit `int`: Beispiele

UNI FREIBURG

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 14 * (12 + 10)
308
>>>
```

Rechnen mit `int`: Beispiele

UNI FREIBURG

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 14 * (12 + 10)
308
>>> 13 % 8
```

Rechnen mit `int`: Beispiele

UNI FREIBURG

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 14 * (12 + 10)
308
>>> 13 % 8
5
>>>
```

Rechnen mit `int`: Beispiele

UNI FREIBURG

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 14 * (12 + 10)
308
>>> 13 % 8
5
>>> 11 ** 11
```

Rechnen mit `int`: Beispiele

UNI FREIBURG

Python-Interpreter

```
>>> 14 * 12 + 10
178
>>> 14 * (12 + 10)
308
>>> 13 % 8
5
>>> 11 ** 11
285311670611
```

Integer-Division: Ganzzahlig oder nicht?

UNI FREIBURG

Der Divisionsoperator `/` liefert das Ergebnis als `float`.
Der Operator `//` rundet auf die nächste ganze Zahl ab.

Python-Interpreter

```
>>> 20 / 3
```

Integer-Division: Ganzzahlig oder nicht?

UNI FREIBURG

Der Divisionsoperator `/` liefert das Ergebnis als `float`.
Der Operator `//` rundet auf die nächste ganze Zahl ab.

Python-Interpreter

```
>>> 20 / 3
6.666666666666667
>>>
```

Integer-Division: Ganzzahlig oder nicht?

UNI FREIBURG

Der Divisionsoperator `/` liefert das Ergebnis als `float`.
Der Operator `//` rundet auf die nächste ganze Zahl ab.

Python-Interpreter

```
>>> 20 / 3
6.666666666666667
>>> -20 / 3
```

Integer-Division: Ganzzahlig oder nicht?

UNI FREIBURG

Der Divisionsoperator `/` liefert das Ergebnis als `float`.
Der Operator `//` rundet auf die nächste ganze Zahl ab.

Python-Interpreter

```
>>> 20 / 3
6.666666666666667
>>> -20 / 3
-6.666666666666667
>>>
```

Integer-Division: Ganzzahlig oder nicht?

UNI FREIBURG

Der Divisionsoperator `/` liefert das Ergebnis als `float`.
Der Operator `//` rundet auf die nächste ganze Zahl ab.

Python-Interpreter

```
>>> 20 / 3
6.666666666666667
>>> -20 / 3
-6.666666666666667
>>> 20 // 3
```

Integer-Division: Ganzzahlig oder nicht?

UNI FREIBURG

Der Divisionsoperator `/` liefert das Ergebnis als `float`.
Der Operator `//` rundet auf die nächste ganze Zahl ab.

Python-Interpreter

```
>>> 20 / 3
6.666666666666667
>>> -20 / 3
-6.666666666666667
>>> 20 // 3
6
>>>
```

Integer-Division: Ganzzahlig oder nicht?

UNI FREIBURG

Der Divisionsoperator `/` liefert das Ergebnis als `float`.
Der Operator `//` rundet auf die nächste ganze Zahl ab.

Python-Interpreter

```
>>> 20 / 3
6.666666666666667
>>> -20 / 3
-6.666666666666667
>>> 20 // 3
6
>>> -20 // 3
```

Integer-Division: Ganzzahlig oder nicht?

UNI FREIBURG

Der Divisionsoperator `/` liefert das Ergebnis als `float`.
Der Operator `//` rundet auf die nächste ganze Zahl ab.

Python-Interpreter

```
>>> 20 / 3
6.666666666666667
>>> -20 / 3
-6.666666666666667
>>> 20 // 3
6
>>> -20 // 3
-7
```

Gleitkommazahlen

UNI FREIBURG

- Syntax von `float`-Konstanten: mit Dezimalpunkt und optionalem Exponent:
  `2.44`, `1.0`, `5.`, `1.5e+100` (bedeutet $1{,}5 \times 10^{100}$)

Die arithmetischen Operatoren für `float` sind die gleichen wie für die ganzzahligen Typen:

- Grundrechenarten: `+`, `-`, `*`, `/`, `//`
- Potenz: `**`
- Rest bei Division für ganzzahliges Ergebnis: `%`

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>>
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>>
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
1.5
>>>
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
1.5
>>> print(1.5 ** 100)
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
1.5
>>> print(1.5 ** 100)
4.06561177535e+17
>>>
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
1.5
>>> print(1.5 ** 100)
4.06561177535e+17
>>> print(10 ** 0.5)
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
1.5
>>> print(1.5 ** 100)
4.06561177535e+17
>>> print(10 ** 0.5)
3.16227766017
>>>
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
1.5
>>> print(1.5 ** 100)
4.06561177535e+17
>>> print(10 ** 0.5)
3.16227766017
>>> print(4.23 ** 3.11)
```

Rechnen mit `float`

UNI FREIBURG

Python-Interpreter

```
>>> print(1.23 * 4.56)
5.6088
>>> print(17 / 2.0)
8.5
>>> print(23.1 % 2.7)
1.5
>>> print(1.5 ** 100)
4.06561177535e+17
>>> print(10 ** 0.5)
3.16227766017
>>> print(4.23 ** 3.11)
88.6989630228
```

Automatische Typkonversionen

Haben die Operanden unterschiedliche Typen, wie in `(-1) ** 0.5`, werden die Operanden nach folgenden Regeln konvertiert:

- Ist einer der Operanden ein `float`, so wird der andere zu `float` konvertiert (falls er das nicht schon ist).

Wieviel ist $2 - 2.1$?

UNI FREIBURG

Python-Interpreter

```
>>> 2 - 2.1
```

Wieviel ist $2 - 2.1$?

UNI FREIBURG

Python-Interpreter

```
>>> 2 - 2.1
-0.10000000000000009
```

Wieviel ist $2 - 2.1$?

UNI FREIBURG

Python-Interpreter

```
>>> 2 - 2.1
-0.10000000000000009
```

- Die meisten Dezimalzahlen können **nicht** exakt als Gleitkommazahlen dargestellt werden (!)

Wieviel ist $2 - 2.1$?

UNI FREIBURG

Python-Interpreter

```
>>> 2 - 2.1
-0.10000000000000009
```

- Die meisten Dezimalzahlen können **nicht** exakt als Gleitkommazahlen dargestellt werden (!)
- Programmier-Neulinge finden Ausgaben wie die obige oft verwirrend — die Ursache liegt in der Natur der Gleitkommazahlen und ist unabhängig von der Programmiersprache.

Überläufe und Unterläufe

UNI FREIBURG

- Ganze Zahlen können beliebig groß (und klein) werden.
- Gleitkommazahlen haben einen eingeschränkten Wertebereich (meist gemäß dem IEEE-754 Standard, double precision).
- Durch Interpreter, aber nicht durch Python festgelegt.

Python-Interpreter

```
>>> 1e-999
```

Überläufe und Unterläufe

UNI FREIBURG

- Ganze Zahlen können beliebig groß (und klein) werden.
- Gleitkommazahlen haben einen eingeschränkten Wertebereich (meist gemäß dem IEEE-754 Standard, double precision).
- Durch Interpreter, aber nicht durch Python festgelegt.

Python-Interpreter

```
>>> 1e-999
0.0
```

Überläufe und Unterläufe

UNI FREIBURG

- Ganze Zahlen können beliebig groß (und klein) werden.
- Gleitkommazahlen haben einen eingeschränkten Wertebereich (meist gemäß dem IEEE-754 Standard, double precision).
- Durch Interpreter, aber nicht durch Python festgelegt.

Python-Interpreter

```
>>> 1e-999
0.0
>>> 1e+999
```

Überläufe und Unterläufe

UNI FREIBURG

- Ganze Zahlen können beliebig groß (und klein) werden.
- Gleitkommazahlen haben einen eingeschränkten Wertebereich (meist gemäß dem IEEE-754 Standard, double precision).
- Durch Interpreter, aber nicht durch Python festgelegt.

Python-Interpreter

```
>>> 1e-999
0.0
>>> 1e+999
inf
```

Überläufe und Unterläufe

UNI FREIBURG

- Ganze Zahlen können beliebig groß (und klein) werden.
- Gleitkommazahlen haben einen eingeschränkten Wertebereich (meist gemäß dem IEEE-754 Standard, double precision).
- Durch Interpreter, aber nicht durch Python festgelegt.

Python-Interpreter

```
>>> 1e-999
0.0
>>> 1e+999
inf
>>> 1e+999 - 1e+999
```

Überläufe und Unterläufe

UNI FREIBURG

- Ganze Zahlen können beliebig groß (und klein) werden.
- Gleitkommazahlen haben einen eingeschränkten Wertebereich (meist gemäß dem IEEE-754 Standard, double precision).
- Durch Interpreter, aber nicht durch Python festgelegt.

Python-Interpreter

```
>>> 1e-999
0.0
>>> 1e+999
inf
>>> 1e+999 - 1e+999
nan
```

`inf` = *infinity*; `nan` = *not a number*. Mit beiden kann weiter gerechnet werden!

Zusammenfassung

UNI FREIBURG

- Python ist eine objektorientierte, dynamisch getypte, interpretierte und interaktive höhere Programmiersprache.
- Python ist sehr populär und wird in den USA als die häufgste Anfängersprache genannt.
- Manchmal ist Python zu langsam und speicherhungrig.
- Erste numerische Typen in Python: `int` und `float`.
- ... mit den üblichen arithmetischen Operationen.