Einführung in die wissenschaftliche Programmierung mit Python

Markus Reinert[1]

Universität Rostock, Institut für Physik

7. Juni 2022

[1] ✉ markus.reinert@io-warnemuende.de

Warum Python?

TIOBE Index May 2022:
Indikator für die Beliebtheit von Programmiersprachen
`https://www.tiobe.com/tiobe-index/`

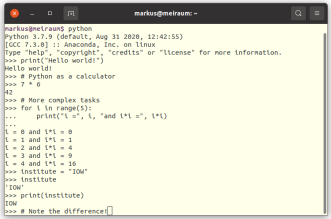
1	**Python**
2	C
3	Java
⋮	
13	R
20	Matlab
25	Julia
30	Fortran

"Python plays a key role in our production pipeline. Without it a project the size of Star Wars: Episode II would have been very difficult to pull off."
`https://www.python.org/about/quotes/`

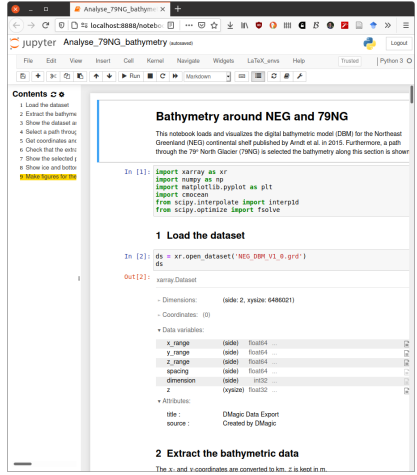
→ **Python ist beliebt, praktisch und bestens für die Wissenschaft geeignet!**

Wie verwendet man Python?

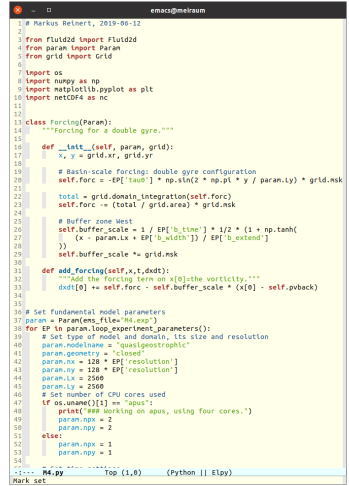
Interaktiv im Terminal:



Als Jupyter Notebook im Internet-Browser:



Als Skript im Texteditor:



Grundlegende Arithmetik

Operation	Operator	Beispiel
Addition	`+`	`1 + 2`
Subtraktion	`-`	`2 - 1`
Multiplikation	`*`	`3 * 4`
Division	`/`	`9 / 4` (gibt 2.25)
Ganzzahl-Division	`//`	`9 // 4` (gibt 2)
Modulo-Rechnung	`%`	`9 % 4` (gibt 1)
Potenzrechnung	`**`	`3 ** 2` (gibt 9)
Wurzelziehen	`**(1/2)`	`9 ** (1/2)` (gibt 3.0)

Beachte:
Integer (Ganzzahlen) werden automatisch in *Floats* (Fließkommazahlen) umgewandelt, wenn es notwendig ist oder wenn ein Operand ein Float ist.

. . . also eigentlich gibt es doch nichts zu beachten (außer man benutzt Python 2).

Außerdem: nur runde Klammern `()` dürfen in Rechnungen verwendet werden.

Strings (Zeichenketten) 1

Strings definiert man wahlweise so: `"Ich bin ein Text"` oder so `'Ich auch'`.

Und wenn der Text Anführungszeichen enthält?

1. `"Er sagte 'Hallo'."` (doppelte Anführungszeichen sind „unsichtbar“)
2. `'Er sagte "Hallo".'` (einfache Anführungszeichen sind „unsichtbar“)
3. `'Er sagte \'Hallo\'.'` (equivalent zu 1.)
4. `"Er sagte \"Hallo\"."` (equivalent zu 2.)

Und wenn der Text besonders lang ist?

- ```
  """Dieser Text geht über drei (!) Zeilen
  und kann " sowie ' enthalten.
  """
  ```
- hauptsächlich als Kommentar / Beschreibung (einer Funktion *etc.*) verwendet
  ```

Strings (Zeichenketten) 2

Mit Strings kann man rechnen:

- `"Hallo " + "Welt"` ergibt `"Hallo Welt"`
- `3 * "wiu"` ergibt `"wiuwiuwiu"`

Achtung: `"4" + "2"` ergibt `"42"` und `"1" * 5` ergibt `"11111"`

Um mathematisch zu rechnen:

- `int("4") + int("2")` ergibt `6`
- `float("0.5") + float("0.5")` ergibt `1.0`
- andersrum: `str(3/2)` ergibt `"1.5"`

Hilfreiche Funktionen:

- `"  Wörter mit Leerzeichen  ".strip()` ergibt `"Wörter mit Leerzeichen"`
- `"Das_ist_ein_Text".replace("_", " ")` ergibt `"Das ist ein Text"`

Strings (Zeichenketten) 3

Gute Möglichkeiten um Zahlen in Text einzufügen:

1. `"The density is {:.1f} kg/m³.".format(1030.49)`
   - die Zahl im Argument von `format` wird verwendet

2a. `f"The density is {rho:.1f} kg/m³."`
   - der Wert der Variable `rho` wird verwendet

2b. `f"The density anomaly is {rho-1000:.1f} kg/m³."`
   - einfache Rechnungen sind möglich
   - komplexe Rechnungen auch, aber dafür ist evtl. Möglichkeit 1 übersichtlicher

Erklärungen:

- die Zahl wird anstelle der geschweiften Klammern eingefügt (`f` für Float)
- die Zahl wird auf 1 Nachkommastelle gerundet (wegen `.1`)
- und es gibt noch viele weitere Formatierungsmöglichkeiten

Variablen und Konstanten

Variablen:

- werden in Python normalerweise mit = definiert
- können beliebige Daten/Objekte enthalten:
  - Zahlen: `x = 3`
  - Strings: `text = "Hallo"`
  - Listen, Funktionen, ...
- können Namen aus Buchstaben, Zahlen und Unterstrichen haben, aber Namen dürfen nicht mit einer Zahl beginnen
- sollten **sinnvolle** Namen haben

Konstanten:

- gibt es in Python nicht
- jede Variable kann überschrieben bzw. verändert werden
- Konvention: Variablennamen in GROSSBUCHSTABEN stehen für Konstanten

Grundlegende Datensammlungen: Listen und *Tuples*

Liste	**Tuple**
Sammlung beliebiger Daten	unveränderbare Liste
`temp = [20, 22, 20.5, "error", 21]`	`dimensionen = (128, 256, 32)`
meist für *unbekannte* Anzahl von Daten *desselben* Typs verwendet	meist für *bekannte* Anzahl von Daten *unterschiedlichen* Typs verwendet

Named Tuple – oft besser geeignet als ein einfaches Tuple:

```
# zu Beginn des Programms:
from collections import namedtuple
# bevor das Tuple verwendet wird:
Dimension = namedtuple("Dimension", ("x", "y", "z"))
# Definition des NamedTuples:
dim = Dimension(128, 256, 32)     # oder: Dimension(x=128, y=256, z=32)
# Verwendung:
dim.x     # ergibt 128
dim.z     # ergibt 32
```

Listen und *Tuples:* Zugriff auf die Daten

- Anzahl der Elemente in einer Liste: `len(daten)`
- `daten[i]` gibt Element `i` aus Liste `daten` zurück
  (geht auch mit NamedTuples, sollte dort aber vermieden werden, siehe vorherige Folie)
- `daten[i] = x` überschreibt Element `i` der Liste `daten` mit dem Wert von `x`
  (geht nicht mit Tuples oder NamedTuples)
- **Wichtig:** Zählung beginnt bei `i = 0`
  Bsp.: hat eine Liste 10 Elemente, so können Indizes 0 bis 9 verwendet werden

Strukturierte Datensammlungen: *Dictionaries*

- `personal_data = {"name": "Max", "age": 23, "city": "Rostock"}`
- Zugriff auf ein Element: `personal_data["name"]`
- Zugriff auf alle Elemente: `personal_data.items()`

For- und *While-*Schleifen

For-Schleife:

feste Anzahl an Wiederholungen, z. B.:

- *n*-mal
- für jedes Element einer Datensammlung
- für jede Datei in einem Ordner

While-Schleife:

Anzahl der Wiederholungen geknüpft an eine Bedingung, z. B.:

- Konvergenz eines numerischen Verfahrens
- sequentielles Lesen von Daten

Wichtig:

Jede Anweisung, die zu einer Schleife gehört,
muss mit derselben Anzahl an Leerzeichen eingerückt sein!

Üblich sind 4 Leerzeichen oder ein Tab.

Ausnahme: einzelne Anweisungen können in derselben Zeile stehen.

Bedingte Anweisungen mit *if*

Vergleichsoperatoren:

- Gleichheit: `==`
- Ungleichheit: `!=`
- Vergleich: `<, >, <=, >=`
- Verknüpfung: `and, or, not`

Mehrere Vergleiche:

```
if x > 0:
  print("x ist positiv")
elif x == 0:
  print("x ist null")
else:
  print("x ist negativ")
```

Wichtig: richtige Einrückung beachten (wie bei Schleifen).

In-line Schleifen und Bedingungen

- können manchmal praktisch sein
- sollten Lesbarkeit verbessern
- können Funktionen ersetzen (z. B. `map`)

Beispiele:

- `half_data = [x / 2 for x in data]`
- `inverse_data = [1 / x if x != 0 else 0.0 for x in data]`
- ```
  print(
      "The influence is",
      "significant" if p <= alpha else "not significant",
  )
  ```
  ```

Funktionen 1

Anweisungen, die an mehr als einer Stelle verwendet werden
und eine bestimmte *Funktion* erfüllen, können als Funktion definiert werden.

Funktionen haben Eingabewerte (Argumente) und Ausgabewerte.

Arten von Funktionsargumenten:

Argumente ohne Namen

Beispiel:

```
print(3, "ist kleiner als", 4)
```

Optionale Namen

Beispiel:

```
np.zeros((3, 4))
# ist equivalent zu
np.zeros(shape=(3, 4))
```

Argumente mit Namen

Beispiel:

```
print("Dieser Text", end=" ")
# Argument ‘‘end’’ kann nicht
# ohne Namen verwendet werden
print("geht hier weiter.")
```

- **Verwende die Schreibweise mit Namen, wenn es die Lesbarkeit verbessert!**
- Argumente ohne Namen müssen immer vor Argumenten mit Namen kommen.
- Die Reihenfolge von Argumenten ohne Namen ist von Bedeutung.
- Die Reihenfolge von Argumenten mit Namen ist egal.

Funktionen 2

Einfachstmögliche Funktion:

```
def square(x):
    return x ** 2
```

Kompliziertere Funktion:

```
def gauss(x, mu=0.0, sigma=1.0):
    """Calculate the value of the Gauss distribution with mean mu and std.dev. sigma at x.

    x: float
    mu: mean of the distribution (default 0)
    sigma: standard deviation (default 1)
    """
    if sigma == 0:
        print("Error!")
        return np.NaN   # beendet den Funktionsaufruf
    z = (x-mu) / (2*sigma)
    return np.exp(- z ** 2) / np.sqrt(2*np.pi*sigma**2)
```

Sammlungen von Funktionen und Daten

Sammlung von (veränderlichen) Daten:	**Sammlung von Funktionen (und ggf. Konstanten):**	**Sammlung von Daten mit *zugehörigen* Funktionen:**
Listen, (Named) Tuples, Dictionaries, andere Datenstrukturen	Module (= Bibliotheken): Python-Dateien, die man mit `import` in ein Programm einbinden kann	Klassen

Programmierstil

"Programs are meant to be read by humans
and only incidentally for computers to execute."
(Harold und Sussman)

- → aussagekräftige Namen wählen
- → Leerzeichen einbauen
- → lange Zeilen umbrechen, am Besten innerhalb von Klammern
- → Einrückungen verwenden
- → Kommentare hinzufügen *# in Python geht das so*

Weitere Tipps:
`https://www.python.org/dev/peps/pep-0020/`
`https://google.github.io/styleguide/pyguide.html`