Woche 04: Programmierung - Module und Funktionen am Beispiel von NumPy (Teil 1)

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele

- Erklären können, was Module und Funktionen in Python bedeuten
- Module in Python importieren können
- Erklären können, was *ndarrays* sind
- *ndarrays* erzeugen können

Inhalt

Was sind Module und Funktionen in Python?

Für fast alle handwerklichen Tätigkeiten gibt es eine Menge Werkzeuge, die unsere Arbeit erleichtern. Die Hilfsmittel und Instrumente werden oft themenbezogen zusammengestellt und in Werkzeugkisten oder Werkzeugkoffern gelagert. So findet man in praktisch jedem

Haushalt einen Werkzeugkoffer mit einem Hammer, einem Satz Schraubenziehern, einem Schraubenschlüssel und einer Kombizange. Einige Hilfsmittel unterstützen uns bei einfachen Aufgaben, andere dagegen übernehmen komplexe Funktionen und erfordern vor Inbetriebnahme einige Einstellungen.

Für die meisten Programmiersprachen gibt es auch solche Werkzeugkisten mit Hilfsmitteln, die schon jemand fertig programmiert und zur freien Nutzung zur Verfügung gestellt hat. Diese Werkzeugkisten werden oft als Packages oder Bibliotheken bezeichnet. Im Zusammenhang mit Python hat sich die Bezeichnung „Modul“ durchgesetzt. Die Module enthalten oft Hilfsprogramme, eigene Datenstrukturen und sogar neue Datentypen.

Für Python wurden bereits sehr viele Module geschrieben. Deswegen kann man in Python große Projekte schnell umsetzen, ohne dafür selbst viel programmieren zu müssen. Besonders beliebt ist diese Programmiersprache, wenn es um die KI-Forschung geht. Inzwischen bietet Python die größte Auswahl an Modulen mit fertig programmierten Machine und Deep Learning Methoden. Diese Methoden stehen üblicherweise in Form von Funktionen innerhalb der Module zur Verfügung.

Unter einer Funktion versteht man in Programmiersprachen einen ausgelagerten Programmabschnitt. Die Auslagerung der Programmabschnitte bietet sich dann an, wenn die gleiche Abfolge von Befehlen in einem Programm an mehreren Stellen oder sogar in anderen Programmen verwendet wird. In diesem Fall wird der Programmabschnitt als eine Funktion mit einem eigenen Namen definiert. Um den Programmcode bzw. die Abfolge von Befehlen einer Funktion in einem Programm auszuführen, wird dann einfach die Funktion durch ihren Namen aufgerufen, anstatt sämtliche Befehle, die die Funktion umfasst, an der notwendigen Stelle neu einzugeben.

Wir haben bereits mehrere sogenannte built-in Funktionen verwendet, die in Python standardmäßig mitgeliefert werden. Eine davon ist die *print()*-Funktion. Du kannst den Programmcode dieser Funktion im Internet finden.

Quelle [1]

Keine Angst, wenn du ihn nicht verstehst. Der Programmcode wurde in einer anderen Programmiersprache geschrieben, die Python ausführen kann. Aber jedes Mal, wenn wir den Namen der *print()*-Funktion als Befehl in unseren Programmen angeben, wird die *print()*-Funktion aufgerufen und ihr Programmcode ausgeführt.

Importieren von Modulen

In diesem Video schauen wir uns das Modul *NumPy* an. NumPy steht für *Numerical Python*. Es ist ein wichtiges Modul für wissenschaftliche Berechnungen.

Quelle [2]

Um die Komponenten eines Moduls im Programm nutzen zu können, muss das Modul zuerst mit dem Befehl *import* importiert werden. Der Name des Moduls, also *numpy*, muss direkt nach der *import*-Anweisung folgen. Um einen Alias zu vergeben, was bei NumPy oft *np* ist, fügt man noch *„as np“* hinzu.

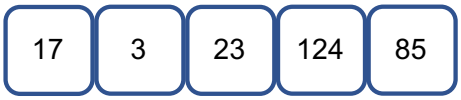
```
import numpy as np
```

Jetzt ist NumPy in unser Programm importiert und wir können seine Konstrukte und Werkzeuge nutzen. Eine detaillierte Beschreibung der Bestandteile des NumPy-Moduls findest du online im Reference Manual.

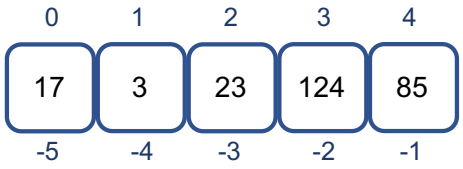
Quelle [3]

NumPy-Datenstruktur *ndarray*

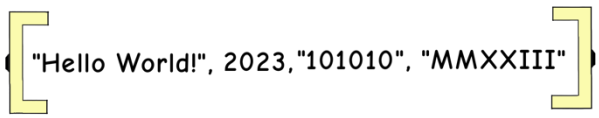
NumPy bietet eine eigene Datenstruktur namens *ndarray* an, die einer verschachtelten Liste ähnlich ist. *Array*s beinhalten wie Listen getrennte Datenobjekte, die an einem Ort gespeichert werden.



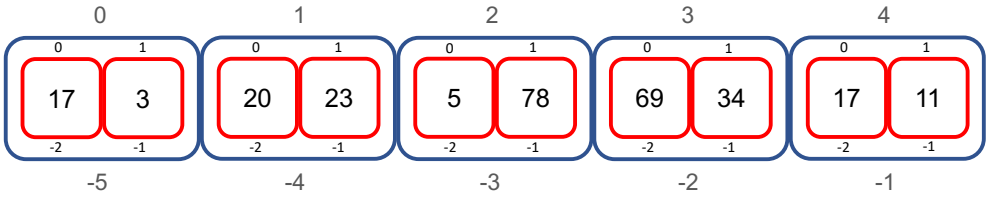
Datenobjekte innerhalb eines Arrays können entweder durch einen positiven oder durch einen negativen Index adressiert werden.



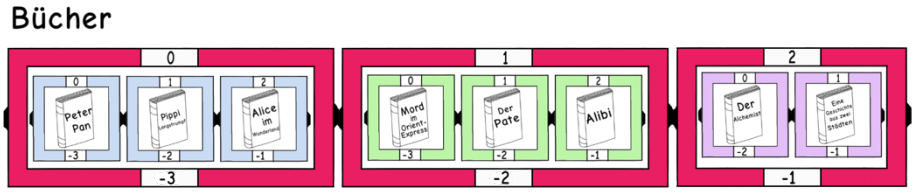
Der erste wesentliche Unterschied zu Listen besteht darin, dass ein Array nur Elemente vom selben Datentyp speichern kann. Zum Beispiel kannst du die Elemente dieser Liste in einem *ndarray* nicht speichern.



Ähnlich wie Listen von Listen, die du schon kennst, können auch Arrays verschachtelt werden. Das heißt, anstatt eines Datenobjektes kann ein Array als Element eines Arrays gespeichert werden.

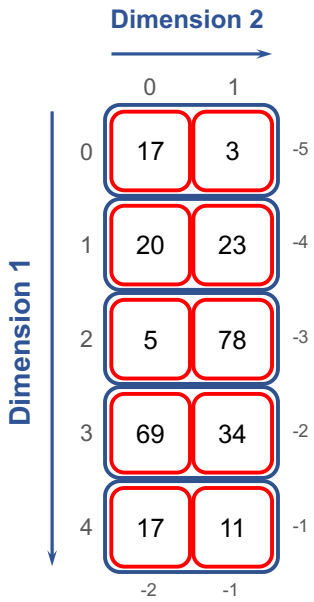


Auf diese Weise entstehen Arrays von Arrays, oder auch Arrays von Arrays von Arrays usw. Ein weiterer wichtiger Unterschied zu verschachtelten Listen besteht darin, dass Arrays auf derselben Hierarchieebene in einem verschachtelten Array die gleiche Anzahl von Elementen enthalten müssen. In diesem Beispiel enthält die verschachtelte Liste Unterlisten mit unterschiedlich vielen Elementen.



Das darf in einem verschachtelten Array nicht passieren.

Für eine bessere Übersicht werden die Arrays von Arrays in Form eines Rechtecks dargestellt. Die Elemente der ersten Hierarchieebene werden von oben nach unten und die Elemente der zweiten Hierarchieebene von links nach rechts angeordnet und durchnummeriert. Dank dieser Anordnung werden die Hierarchieebenen eines verschachtelten Arrays als *Dimensionen* bezeichnet.

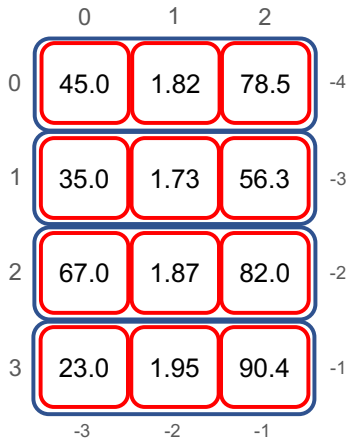


Deswegen verwendet man in Bezug auf verschachtelte Arrays häufiger die Bezeichnung *mehrdimensionale Arrays*. Beim *ndarrays* des NumPy-Moduls handelt es sich gerade um mehrdimensionale Arrays.

In *ndarrays* werden oft Werte einer Datentabelle gespeichert.

	Alter	Körpergröße	Gewicht
Patient 1	45	1.82	78.5
Patient 2	35	1.73	56.3
Patient 3	67	1.87	82
Patient 4	23	1.95	90.4

Wenn diese numerischen Werte unterschiedlichen Datentyps wie zum Beispiel ganze Zahlen und Fließkommazahlen enthält, dann empfiehlt es sich einen Datentyp mit einer höheren Präzision wie zum Beispiel *float*, also Fließkommazahlen, für die Speicherung der Werte im *ndarray* zu wählen.



Erzeugung von *ndarrays*

Wenn du bereits weißt, welche Werte du in einem *ndarray* speichern möchtest, kannst du das *ndarray* wie folgt erzeugen: Zuerst gibst du den Namen deines *ndarrays* an. Wir nennen unser *ndarray patienten_daten*. Nach dem Zuweisungsoperator muss der Modulname, also *np*, angegeben werden, weil wir die Datenstruktur dieses Moduls nutzen. Nach dem Punkt geben wir die Bezeichnung der Funktion an, mit deren Hilfe ein *ndarray* erzeugt werden kann. Die *ndarrays* werden mit der NumPy-Funktion *array()* erzeugt.

Quelle [4]

In der runden Klammer der Funktion führen wir die Werte aus unserer Datentabelle auf. Diese werden wie eine verschachtelte Liste angegeben. (Elemente eingeben.) Nach dem Komma kann der Datentyp der Elemente spezifiziert werden. Diese Angabe ist aber optional. Wenn wir den Datentyp nicht angeben, wird dieser automatisch bestimmt.

Wir haben unser *ndarray* erzeugt und können dieses mit dem *print()*-Befehl ausgeben.

```
patienten_daten = np.array([[45, 1.82, 78.5], [35, 1.73, 56.3],
                            [67, 1.87, 82], [23, 1.95, 90.4]])

print(patienten_daten)
```

Wie wir in der Ausgabe sehen können, wurden alle ganzen Zahlen automatisch in Fließkommazahlen umgewandelt. Wenn du aber den genauen Datentyp der Werte deines *ndarrays* wissen möchtest, kannst du es mit dem Befehl *dtype* herausfinden. Dafür gibst du den Namen deines *ndarrays* an, dann einen Punkt und danach den *dtype*-Befehl.

```
print("Datentyp der Elemente des ndarrays patienten_daten:",
      patienten_daten.dtype)
```

In der Ausgabe sehen wir, dass die Werte unseres *ndarrays* vom Typ *float64* sind. Wenn du wissen möchtest, was die Zahl hinter dem *float* bedeutet, kannst du es im Manual nachlesen.

Quelle [5]

Ein weiterer nützlicher Befehl für *ndarrays* ist *shape*. Dieser gibt die Dimensionen des *ndarrays* an, also die Anzahl der Elemente innerhalb der Arrays jeder Hierarchieebene. Um die Dimensionen unseres *ndarrays* herauszufinden, geben wir nach dem Arraynamen einen Punkt und die Anweisung *shape* an.

```
print("Dimension des ndarrays patienten_daten:",
      patienten_daten.shape)
```

Da wir jede der vier Zeilen unserer Datentabelle als ein Unterarray gespeichert haben, besteht unser *ndarray* aus vier Zeilen und drei Spalten, was der Anzahl der Features entspricht.

Quellen

Quelle [1] Python Software Foundation (2022). *The Python Programming Language, Version 3.10.9 [Source Code]* https://github.com/python/cpython/blob/9345eea2568bc590e1c2d4c418e3238ca7948ceb/Python/bltinmodule.c#L1945

Quelle [2] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. (2020). Array Programming with NumPy. *Nature 585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Quelle [3] NumPy (2022). *NumPy Reference, Release 1.23.* https://numpy.org/doc/1.23/reference/index.html

Quelle [4] NumPy (2022). The N-dimensional array (ndarray). In *NumPy Reference, Release 1.23*. https://numpy.org/doc/1.23/reference/arrays.ndarray.html

Quelle [5] NumPy (2022). Scalars. In *NumPy Reference, Release 1.23*. https://numpy.org/doc/1.23/reference/arrays.scalars.html#numpy.double

Weiterführendes Material

Schmitt, S. (2021). *Python Kompendium: Professionell Python Programmieren lernen.* BMU Media Verlag

Barry, P. (2017). *Python von Kopf bis Fuß.* O'Reilly

Disclaimer

Transkript zu dem Video „Woche 04: Programmierung - Module und Funktionen am Beispiel von NumPy (Teil 1)", Ludmila Himmelspach.