Einführung in das Programmieren

Grundlagen und erste Schritte in python

Christoph Draxler, Vanessa Reichel
draxler@phonetik.uni-muenchen.de
vanessa.reichel@campus.lmu.de

24. April 2023

Grundlegende Datentypen

Die meisten Programmiersprachen unterscheiden die folgenden *Datentypen*

Zahlen numerische Werte: natürliche, ganze oder Gleitkommazahlen, große, kleine, einfach oder doppelt genaue Zahlen, usw.

Zeichen Buchstaben und Zeichenketten

Wahrheitswerte die Werte *wahr* und *falsch*

null den *Nullwert*, einen undefinierten Wert

Datentypen

Ein Datentyp

- hat einen *Namen*, z.B. `str`, `int`, `float`, `bool`, `list`, o.ä.
- einen *Wertebereich*, z.B. die natürlichen Zahlen im Intervall von $0 \leq n < 65367$
- eine festgelegte Schreibweise, z.B. werden Zeichenketten in einfache (') oder doppelte (") Anführungszeichen geschrieben, also `'python-Kurs'` oder `"python-Kurs"`

In python kann man mit der Funktion `type()` den Datentyp eines Werts oder einer Variablen ermitteln.

Operationen

Jeder Datentyp hat spezifische *Operationen*, die durch *Operatoren* oder *Funktionen* ausgedrückt werden.

Zahlen	Addition, Subtraktion, Multiplikation, Division Wurzel, Potenz	$+$, $-$, $*$, $/$, $\sqrt{n}$, n^m

Zeichenketten	sortieren, verknüpfen
	suchen, ersetzen
	konvertieren
		`sort()`, `join()`
		`search()`, `replace()`
		`upper()`, usw.

Wahrheitswerte	UND, ODER, NICHT	$\vee, \wedge, \neg$

Weil die wenigsten Operatorsymbole auf der Tastatur verfügbar sind, werden sie in Programmiersprachen meist als Funktionen geschrieben, z.B. $\sqrt{n}$ als `math.sqrt(n)` (für engl. *square root*).[1]

[1] In python muss dafür die Library `math` importiert werden.

Variablen

Variablen sind benannte *Platzhalter* für Werte. Sie haben

Namen	möglichst sprechend und eindeutig
Gültigkeit	einen Bereich im Code, in dem sie gültig sind
Sichtbarkeit	einen Bereich im Code, in dem sie sichtbar sind

Moderne Programmiersprachen unterscheiden Groß- und Kleinschreibung. `anzahl`, `Anzahl` und `anZahl` sind also drei verschiedene Variablennamen.

Zuweisungen

Variablen erhalten ihre Werte durch *Zuweisung* per *Zuweisungsoperator*. In python ist dies '='.

$x = 4$	die Variable x erhält den Wert 4
$y = 5.12$	die Variable y erhält den Wert $5,12$
$z = x + y$	die Variable z erhält den Wert aus der Addition von x und y
$x = x + 4$	die Variable x erhält den Wert aus der Addition des alten Werts von x und 4

Was sind die Werte der drei Variablen, wenn die obigen 4 Zuweisungen in dieser Reihenfolge ausgeführt worden wären?

Erste Schritte

- Legen Sie ein Arbeitsverzeichnis an – das ist einfach ein Ordner im Dateisystem, z. B. `Code`
- Öffnen Sie auf Ihrem Rechner den Editor Visual Studio Code.
- Ziehen Sie den Ordner auf die linke Seitenleiste von Visual Studio Code.
- Klicken Sie auf den Reiter `Terminal` und geben Sie anschließend `python3` ein.

Wenn Sie am linken Fensterrand >>> sehen wartet python auf die Eingabe Ihrer Befehle.

Erste Eingaben

Führen Sie die folgenden Schritte in der python-Konsole aus und achten Sie auf die Reaktion der Konsole.

- Geben Sie rechts vom >>> eine einfache Zahl ein, z.B. 42
- Geben Sie rechts vom >>> den Befehl `x = 42` ein
- Geben Sie rechts vom >>> den Befehl `x` ein
- Geben Sie rechts vom >>> den Befehl `y = x * x` ein
- Geben Sie rechts vom >>> den Befehl `y = y * x` ein

Überlegen Sie vor dem Drücken der *Return*-Taste jeweils, welche Reaktion der Konsole Sie erwarten.

Erläuterungen

Mit den vorangegangenen Befehlen haben Sie Werte direkt eingegeben bzw. Variablen Werte zugewiesen.

- `x` und `y` sind Variablen
- `42` ist ein numerischer Wert
- `*` ist die Multiplikation

Welchen Wert haben `x` und `y` nach Ausführung der Befehle? Geben Sie einfach `x` oder `y` rechts vom > ein.

Funktion *definieren*

Mit einer Funktions*definition* kann man Folgen von Befehlen unter einem Namen zusammenfassen, um sie später wiederzuverwenden. Eine Funktionsdefinition besteht aus der *Signatur*, auch *Funktionskopf* genannt, und dem *Funktionsrumpf* mit der eigentlichen Definition.

Funktion definieren

Der Funktionskopf

- wird mit dem reservierten Wort `def` markiert,
- hat einen eindeutigen Namen,
- kann Argumente in *runden* Klammern haben,
- wird mit einem ':' abgeschlossen

Der Funktionsrumpf enthält

- einen oder mehrere Befehle, einen pro Zeile und nach rechts eingerückt
- und einem `return()` am Ende

Funktionen *ausführen*

Um eine Funktion auszuführen muss sie *aufgerufen* werden.

Dazu schreibt man den Funktionsnamen gefolgt von Argumentwerten oder Variablen in der Klammer.

Auch wenn eine Funktion keine Argumente benötigt, schreibt man die Klammern. In diesem Fall bleiben sie aber leer.

Beispiel: LMU Mailadressen

Legen Sie eine neue python-Datei namens `lmumail.py` an und schreiben Sie den folgenden Code dort hinein.

```
def lmumail (vorname, nachname):
    links =  vorname + "." + nachname
    rechts = "campus.lmu.de"

    mail = links + "@" + rechts
    return (mail)
```

Drücken Sie im Editorfenster oben rechts auf das dreieckige Symbol, um die Datei in der Konsole zu laden. Was passiert?

`lmumail()` *ausführen*

1. Wählen Sie im Editor die Zeilen von `def...` bis `return...` aus.
2. Drücken Sie gleichzeitig die Umschalt-Taste und die Return-Taste.
3. Die Datei wird in die Konsole geladen, und python wartet mit dem `>>>` auf Ihre nächste Eingabe.
4. Geben Sie `print(lmumail("Christoph", "Draxler"))` ein und drücken Sie die Return-Taste.
5. In der Konsole wird `Christoph.Draxler@campus.lmu.de` ausgegeben, danach wartet python wieder auf Ihre Eingabe.
6. Geben Sie `quit()` ein um den Ablauf zu beenden.

Was ist hier passiert?

Mit der Auswahl der Zeilen und dem gleichzeitigen Drücken der Umschalt- und der Return-Taste wurden die ausgewählten Zeilen in python eingelesen. In diesen Zeilen stand aber nur die Funktions*definition*, d. h. kein ausführbarer Code.
Erst mit der Eingabe des Befehls `print(lmumail("Christoph", "Draxler"))` wurde die Funktion *aufgerufen*, das Ergebnis berechnet und ausgegeben.

Analyse der Funktion

Die Funktion `lmumail()` führt folgende Schritte aus:

- den Variablen `links` und `rechts` werden Werte *zugewiesen*
  - `links` wird das Ergebnis des Zusammenfügens von `vorname`, `'.'` und `nachname` zugewiesen
  - `rechts` wird eine einfache Zeichenkette zugewiesen
- dann werden beide mit dem Operator + zusammengefügt, und das Ergebnis wird der Variablen `mail` zugewiesen
- `mail` wird zum Abschluss der Funktion *zurückgegeben*.

+ und `return()` sind in python vordefiniert. Beachten Sie die Verwendung sprechender Namen und die Leerzeile – dadurch ist der Code recht verständlich.

Beispiel: Addition zweier Zahlen

```
def summe (n, m):
    s = n + m
    return(s)
```

Rufen Sie die Funktion mit konkreten Werten oder Variablen auf:

```
x = summe(4, 5)
n = 21
m = 20
y = summe(n, m)
print(x, y)
```

Frage in die Runde

Warum sind die runden Klammern immer notwendig, auch wenn die Funktion keine Argumente hat?

Frage in die Runde

Warum sind die runden Klammern immer notwendig, auch wenn die Funktion keine Argumente hat?
Um bereits auf syntaktischer Ebene Funktionsaufrufe von Variablen unterscheiden zu können.

Analyse der Funktionsdefinition

`def`	vordefinierter Befehl in python
`summe`	selbstgewählter Name der Funktion,
`(n, m)`	zwei mit Komma getrennte Argumentvariablen `n` und `m`
`:`	Abschluss des Funktionskopfes
Leerzeichen	eingerückter Code
`s = n + m`	`s` bekommt den Wert der Addition von `n` und `m`
`return (s)`	Rückgabe des berechneten Werts
Leerzeile	Ende der Funktionsdefinition

Wie hätte man die Funktion auch etwas kürzer definieren können?

Anmerkungen zu Funktionen

- Argumente dienen dazu, Werte *in* die Funktion zu bringen
- `return()` dient dazu, Werte an den Aufrufer der Funktion *zurückzugeben*
- eine Funktion kann weitere Funktionen aufrufen
- Variablen, die nur innerhalb einer Funktionsdefinition stehen, sind *lokal*, d.h. außerhalb der Definition nicht sichtbar
- es ist guter Programmierstil, in Funktionen *ausschließlich* lokale Variablen zu verwenden

Achtung! python hindert einen nicht daran, in Funktionen nicht-lokale Variablen zu verwenden.

Anschaulicher Vergleich

Im Alltag gibt es etwas Ähnliches wie Funktionen auch – z. B. Terminvereinbarung via Telefon:

- Sie müssen wissen, dass es diese Funktion überhaupt gibt.
- Sie rufen an und geben die notwendigen Daten an: Name, Vorname, gewünschter Termin. . . – das sind die Argumente
- Sie warten, während Ihr Terminwunsch bearbeitet wird. Wie das genau geschieht, bleibt Ihnen verborgen – hier werden die notwendigen Schritte ausgeführt.
- Nach mehr oder weniger langer Zeit erhalten Sie eine Antwort, z. B. einen konkreten Termin – das ist die Rückgabe der Funktion.
- Mit dieser Antwort können Sie dann weitermachen, z. B. den Termin in Ihren Kalender eintragen.

Aufgabe

Gegeben sei folgender Code:

```
n = "Hello World."
m = 10

def summe (n, m):
    s = n + m
    return (s)
```

Was ergeben die folgenden Anweisungen?

- `x = summe(4, 5)`
- `y = summe(4, m)`
- `z = summe(n, m)` Error

Was passiert hier?

Kommentare

Kommentare sind umgangssprachlich formulierte Hinweise, was der Code machen soll.

- das Zeichen für einen Kommentar ist #
- alles, was nach dem # in einer Zeile steht, ist Teil des Kommentars
- Kommentare werden von python ignoriert, aber sie
- helfen Programmierern, den Code zu verstehen

Schreiben Sie vor jede Funktion, die Sie definieren, einen Kommentar, und außerdem immer dann, wenn der Code ansonsten unverständlich wird.

Kommentare für `lmumail()`

```
# ---
# lmumail(v, n) erzeugt eine Standard LMU-Mailadresse aus
# Vor- und Nachnamen.
# Chr. Draxler, 29.04.2015
# ---
def lmumail (vorname, nachname):
    links =  vorname + "." + nachname
    rechts = "campus.lmu.de"

    mail = links + "@" + rechts
    return (mail)
```

Schreibkonventionen

Schreibkonventionen ähneln Glaubenskriegen oder Mac vs. PC Diskussionen. Dabei ist es ganz einfach!

- sprechende Variablennamen
- nur ein Befehl pro Zeile
- nach jedem Doppelpunkt die folgenden zusammengehörenden Codezeilen nach rechts einrücken
- vor und nach einem Operator ein Leerzeichen
- Leerzeilen zum visuellen Gliedern des Codes in zusammenhängende Abschnitte

'Elegance is not optional' – nur schöner Code ist guter Code.

python zwingt einen zur Einhaltung gewisser Layouts, sonst funktioniert der Code nicht.