Python 3 Referenzkarte

Trennzeichen

Einzelne Befehle werden in Python durch den Zeilenumbruch getrennt. Kontrollstrukturen werden hinter dem Doppelpunkt `:` eingerückt.

Grundlegende Datentypen

`int` Ganze Zahlen: `-12, 5`
(können beliebige Länge haben)

`float` Kommazahlen: `4.5, -3.0E10, 0.3e-12`

`string` Zeichenketten: `"Hallo␣Welt"`

```
'Hallo␣Welt'
"""
  Hallo
  Welt!
  mehrzeilig
"""
```

`bool` Wahrheitswert: `"True,␣False"`

Variablen

- Variable muss mit einem Buchstaben oder dem Unterstrich (_) beginnen
- darf alle Buchstaben des Alphabets (auch Umlaute), den Unterstrich oder Ziffern enthalten
- Schlüsselworte sind nicht erlaubt

Schlüsselworte in Python:

`and,`	`else,`	`import,`	`return,`
`assert,`	`except,`	`in, is,`	`try,`
`break,`	`exec,`	`lambda,`	`while,`
`class,`	`finally,`	`not, or,`	`yield`
`continue,`	`for, from,`	`pass,`	
`def, del,`	`global,`	`print,`	
`elif,`	`if,`	`raise,`	

Wertzuweisung

`x=5` (Datentyp wird bei der Zuweisung festgelegt.)

Numerische Opertatoren

`+` Addition, Verkettung (Strings)
`-` Subtraktion
`*` Multiplikation
`/` Division
`//` ganzzahlige Division
`%` Modulo, Rest bei ganzzahliger Division
`**` Potenzieren

Vergleichsoperatoren

`>` größer
`>=` größer gleich
`==` gleich
`<=` kleiner gleich
`<` kleiner
`!=` ungleich

Logische Operatoren

`A and B` ist wahr wenn A und B wahr sind
`A or B` ist wahr wenn A oder B oder beide wahr sind
`not A` ist wahr, wenn A falsch ist

Typumwandlungen

`int()` wandelt einen Ausdruckin eine ganze Zahl um
`float()` wandelt einen Ausdruckin eine Kommazahl um
`str()` wandelt einen Ausdruck in eine Zeichenkette um

Einfache Aus- und Eingaben

`print()` Funktion zur Ausgabe auf der Kommandozeile

```
print("Hallo␣Welt!")
print("x=",x)
print(a,"␣+␣",b,"␣=␣",a+b)
```

`input()` unterbricht das Programm und wartet auf eine Eingabe (Typ `str`) von der Kommandozeile

```
eingabe=input()
x=int(input("x=")) (x= wird vor der Eingabe ausgegeben und die Eingabe wird in int umgewandelt.)
```

Kontrollstrukturen

Einseitige Alternative Wenn Bedingung erfüllt, dann führe aus...

```
if <bedingung>:
   ... anweisungen ...
```

Zweiseitige Alternative Wenn Bedingung erfüllt, dann führe aus... , anderenfalls führe aus ...

```
if <bedingung>:
   ... anweisungen ...
else:
   ... anweisungen ...
```

Mehrfachverzweigung Wenn Bedingung 1 erfüllt, dann führe aus..., Wenn Bedingung 2 erfüllt, dann führe aus..., anderenfalls führe aus...

```
if <bedingung 1>:
   ... anweisungen ...
elif <bedingung 2>:
   ... anweisungen ...
elif <bedingung 3>:
   ... anweisungen ...
...
else:
   ... anweisungen ...
```

Schleife mit Abbruchbedingung Solange Bedingung erfüllt, führe aus ...

```
while <bedingung>:
        ... Anweisungen ...
```

Weitere Schleifenanweisungen:

`break` bricht die Schleife ab
`continue` fährt mit dem nächsten Schleifendurchlauf fort

Zählschleife Für `zahl = anfangswert` bis `zahl = endwert` mit `schrittweite` wiederhole ...

```
for i in range(a,e+1,s):
    ...Anweisungen...
```

(a-Anfangswert, e-Endwert, s-Schrittweite)

Beispiele für `range` :

`range(5)` a=0,e=4,s=1 [0,1,2,3,4]

`range(1,5)` a=1,e=4,s=1 [1,2,3,4]

`range(1,5,2)` a=1,e=4,s=1 [1,3]

Listen

strukturierter Datentyp, der aus mehreren unterschiedlichen Werten (auch mit unterschiedlichen Datentypen) besteht

Beispiele:
`[1,23,7,4]`
`["Hallo",4,34.2,[1,2,3]]`

- Defintion von Listen:
  `liste1=[1,23,7,4]`
  `liste2=[] #leere Liste`
- Zugriff auf Listenelemente:
  `liste1=[1,23,21,23,12]`

```
>>> print(liste1[2])
21
```

(Zählung beginnt bei 0.)

- Teilbereiche von Listen - Slices

```
>>>liste1=[1,23,21,23,12]
>>> print(liste1[1:3])
[23, 21]
>>> print(liste1[:3])
[1, 23, 21]
>>> print(liste1[1:])
[23, 21, 23, 12]
```

- Mehrdimensionale Listen
  Definierte Liste:
  `liste1=[[1,2,3],[4,5,6],[7,8,9]]`

```
>>> print(liste1[1])
[4, 5, 6]
>>> print(liste1[1][2])
6
```

- Weitere Listenfunktionen
  Wir betrachten `liste=[1,23,21,23,12]`

```
>>> # Element ändern:
>>> liste1[1]=555
>>> print(liste1)
[1, 555, 21, 23, 12]

>>> # Element anhängen:
>>> liste1.append(555)
>>> print(liste1)
[1, 23, 21, 23, 12, 555]
>>> # Teilbereiche oder Elemente lö
    schen:
>>> del liste1[1:3]
>>> print(liste1)
[1, 23, 12]
```

- Verwendung der for-Anweisung auf Listen

```
# wert nimmt nacheinander den Wert
    jedes Listenelements an
for wert in liste:
    ...Anweisungen...
```

Zeichenketten (Strings)

Zeichenketten als strukturierter Datentyp

- Zugriff auf einzelne Zeichen

```
>>> zeichenkette = "Informatik"
>>> print("Erster␣Buchstabe:",
    zeichenkette[0])
Erster Buchstabe: I
>>> print("Wort␣im␣Wort:",
    zeichenkette[2:6])
Wort im Wort: form
```

- Operationen

```
>>> wort1 = "Hallo"
>>> wort2 = "Welt"
>>> nachricht = wort1 + "␣" + wort2
>>> print(nachricht)

Hallo Welt
>>> wort = "Hallo"
>>> satz = 5 * wort
>>> print(satz)
HalloHalloHalloHalloHallo
```

- Funktionen auf Zeichenketten
  Verwendung: `zeichenkette.funktion()`

  `len()` gibt die Länge (Anzahl der Zeichen) der Zeichenkette zurück
  `lower()` konvertiert alle Buchstaben zu Kleinbuchstaben
  `upper()` konvertiert alle Buchstaben zu Großbuchstaben
  `strip()` entfernt führende und folgende Leerzeichen
  `split(trennzeichen)` teilt die Zeichenkette anhand eines Trennzeichens in eine Liste auf
  `replace(wort1,wort2)` ersetzt wort1 in der Zeichenkette durch wort2
  `find(wort)` findet die Position von wort in der Zeichenkette
  `count(wort)` zählt, wie oft wort in der Zeichkette vorkommt
  `sorted()` gibt eine alphabetisch sortierte Liste der Zeichen zurück

Funktionen

Definition einer Funktion:

```
def funktionsname():
    ... Anweisungen ...
```

Aufrufen einer Funktion:

```
... Anweisungen ...
funktionsname()
... Anweisungen ...
```

Funktion mit Parametern und Rückgabewerten:

```
def funktionsname(p1, p2, ...):
    ... Anweisungen ...
        return wert
```

Aufruf:

```
... Anweisungen ...
variable = funktionsname(w1,w2,...)
... Anweisungen ...
```

Lokale und globale Variablen

- Lokale Variablen sind nur innerhalb einer Funktion gültig
- Globale Variablen sind im gesamten Pythonprogramm gültig
- Um eine globale Variable innerhalb einer Funktion zu ändern muss das Schlüsselwort `global` verwendet werden.

```
def funktion():
    global g # Zugriff auf globale
        Variable
    l = 3 # lokale Variable
    g = g + 3 # g wird um 3 erhöht
    print(l+g)

g = 3 # Definition globale Variable

funktion() # Ausgabe 9
```

Module

Funktionen können in Extradateien ausgelagert und als Module wieder eingebunden werden. Python besitzt einige Standardmodule.

Beispiel:

```
# Zugriff auf alle enthaltenden
    Funktionen
import module
module.function_name() #Aufruf der
    Funktion über Punktnotation

from module import *
function_name() #Aufruf der Funktion

#Es auch nur einzelne Funktionen eines
    Moduls importieren:
from module import function_name
function_name()
```

Klassen und Objekte

```
class Meineklasse():

        def __init__(self,a):
                self.x=a

        def meinemethode(self):
                <Anweisungen>

        def __str__(self):
                return "
                   Klasseninformation"
```

Erläuterung:

`class` Schlüsselwort zur Definition der Klasse
`self` Schlüsselwort für Klassenmethoden und -variablen
`__init__(self...)` Konstruktormethode der Klasse
`x` Klassenvariable
`meinemethode(self)` Klassenmethode
`__str__(self)` gibt eine Zeichenkette zurück, die bei `print(klassenname)` ausgegeben wird.

Benutzung von Klassen:

Erzeugen einer Instanz der Klasse:
`Meinobjekt=Meineklasse(5)`

Aufruf der KLassenmethode für `Meinobjekt`:
`Meinobjekt.meinemethode()`

Vererbung:

```
class Meineneueklasse(Meineklasse):

        ...Anweisungen...
```

`Meineneueklasse` erbt alle Klassenmethoden und Variablen von `Meineklasse`.

Copyright © November 12, 2026 Lutz Herrmann