

Kurzübersicht Python

Datentypen

Integer	i=1	Ganzzahl	unveränderlich (immutable)
Float	f=0.1	Gleitkommazahl	
String	s="Hallo"	Zeichenkette	
Tupel	t=(1, 2, 3)	Unveränderbare Liste mit Reihenfolge	
Dictionary	d={1: "eins", 2: "zwei", 3: "drei"}	Assoziatives Array	veränderlich (mutable)
Liste	l=[1, 2, 3]	Veränderbare Liste mit Reihenfolge	

Umwandlung von Datentypen

int(2.33) liefert 2
float(3) liefert 3.0
str(53) liefert die Zahl 53 als Zeichenkette

Ausdrücke und Operatoren

Klammern

gruppieren Operatoren und Operanden, um die Reihenfolge der Auswertung zu beeinflussen

Arithmetik

+ (plus), - (minus), * (mal), / (dividiert durch), ** (hoch), % (Modulo, Rest einer Division)

16/3 liefert 5 (Ganzzahldivision)

16.0/3 liefert 5.333333333333333 (Gleitkommazahldivision)

Vergleiche

== (gleich), != (ungleich), > (größer), < (kleiner), >= (größer oder gleich), <= (kleiner oder gleich)

Logik

true (wahr), false (falsch), and (und zugleich), or (oder), not (nicht)

Element-Beziehung

in (in einer Elementmenge), z.B.

3 in [1, 2, 3] (true)

4 not in [1, 2, 3] (true)

Zeichenketten

Zeichenketten können mit + verknüpft oder mit * wiederholt werden, z.B.

“abc“+“def“ liefert “abcdef“

5*“A“ liefert “AAAAA“

Sequenzen

Als Sequenzen werden Tupel (1, 2, 3) und Listen [1, 2, 3] bezeichnet. Im Gegensatz zu Dictionaries sind Sequenzen geordnet, d.h. jedes Element hat eine eindeutige Position ab Position 0 (Null) für das erste Element der Liste, 1 für das zweite usw.

-1 kann für das letzte Element gesetzt werden, -2 für das vorletzte usw.

z.B. (3, 4, 5)[0] liefert 3

(3, 4, 5)[-1] liefert 5

Listen-Generatoren

range(10) entspricht [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

range(1,4) entspricht [1,2,3]

Funktions-Definition

```
def quadrat(x):
```

```
    return x ** 2
```

quadrat(3) liefert 9

Befehle bei Zahlen und Zeichenketten

round(a [, n]) - rundet eine Fließkommazahl; optional kann die Anzahl der Nachkommastellen angegeben werden,

z.B. **round(5.59555, 2)** liefert 5.6

int() - in Ganzzahl konvertieren, **float()** - in Fließkommazahl konvertieren.

str() - in String konvertieren

Nicht nur Zahlen, sondern generell alle Objekte kann man mittels **str()** in einen String konvertieren, z.B.

print (str(167)[-1]) liefert 7

len() - Länge des Strings ermitteln, z.B.

test_string=“Heute“

len(test_string) liefert 5

Kontrollstrukturen

if-elif-else

Beispiel:

```
x = 8
if x > 5:
    print ("x ist größer als 5")
elif x < 5:
    print ("x ist kleiner als 5")
else:
    print ("x ist 5")
```

Ausgabe:

x ist größer als 5

while

Beispiel:

```
i = 0
while i < 4:
    print ("i ist", i)
    i = i + 1
```

Ausgabe:

i ist 0
i ist 1
i ist 2
i ist 3

for

Beispiel:

```
for i in range(10,21):
    print (i*i,end=' ')
```

Ausgabe:

100 121 144 169 196 225 256 289 324 361 400

break, continue und else

Beispiel:

```
for i in "Hallo Welt!":
    if i == " ":
        break
    print (i,end='')
```

Ausgabe:

Hallo

Schlüsselwörter

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

Höhere Datentypen

Tupel

Unveränderbare Liste mit Reihenfolge

Befehlsauszug

+	Verkettung von Tupeln	<code>("Das","ist") + ("eine", "Addition")</code>
*	Vervielfachung von Tupeln	<code>3 * ("Das","ist")</code>
[]	Zugriff auf ein Element entsprechend des Indexes	<code>zahlen = (1, 5, 3, 5, 22) print(zahlen[2])</code>
[a:b]	Zugriff auf einen Bereich zwischen den Indexangaben – $a \leq \text{Bereich} < b$	<code>zahlen = (1, 5, 3, 5, 22) print(zahlen[2:4])</code>
in not in	Prüft, ob ein Element enthalten bzw. nicht enthalten ist	<code>zahlen = (1, 5, 3, 5, 22) if 7 in zahlen:</code>
len	Liefert die Anzahl der Elemente im Tupel	<code>zahlen = (1, 5, 3, 5, 22) print(len(zahlen))</code>
count(o)	Zählt das Vorkommen eines Elements o im Tupel	<code>zahlen = (1, 5, 3, 5, 22) print(zahlen.count(5))</code>
index(o)	Liefert den Index des ersten Vorkommens eines Elements o im Tupel	<code>zahlen = (1, 5, 3, 5, 22) print(zahlen.index(5))</code>

Dictionary

Assoziatives Array

Befehlsauszug

[]	Lesender bzw. schreibender Zugriff auf einen Wert über den Schlüssel	<code>d = {"a": 10, "b": 20} print(d["a"])</code>
in not in	Prüft, ob ein Schlüssel enthalten ist bzw. nicht enthalten ist	<code>d = {"a": 10, "b": 20} if "a" in d:</code>
len	Liefert die Anzahl der Schlüssel-Werte-Paare im Dictionary	<code>d = {"a": 10, "b": 20} print(len(d))</code>
del []	Löscht einen Schlüssel mit dem zugehörigen Wert	<code>d = {"a": 10, "b": 20} del d["a"]</code>
copy()	Erzeugt eine Kopie	<code>d = {"a": 10, "b": 20} d2 = d.copy</code>
get(s,a)	Liefert den Wert zu einem Schlüssel s , falls s nicht existiert, wird a zurückgegeben	<code>d = {"a": 10, "b": 20} d.get("a","nicht da")</code>

pop(s)	Liefert den Wert zu einem Schlüssel und löscht beide aus dem Dictionary	d = {"a": 10, "b": 20} d.pop("a")
update()	Fügt dem Dictionary ein Dictionary hinzu – gleiche Schlüssel werden überschrieben	d = {"a": 10, "b": 20} d3 = {"x": 90, "b": 100} d.update[d3]

Liste

Veränderbare Liste mit Reihenfolge

Befehlsauszug

+	Verkettung von Listen	["Das","ist"] + ["eine", "Addition"]
*	Vervielfachung von Listen	3 * ["Das","ist"]
[]	Zugriff (lesend/schreibend) auf ein Element entsprechend des Indexes	zahlen = [1, 5, 3, 5, 22] print[zahlen[2]]
[a:b]	Zugriff (lesend/schreibend) auf einen Bereich zwischen den Indexangaben – a <= Bereich < b	zahlen = [1, 5, 3, 5, 22] print[zahlen[2:4]]
in not in	Prüft, ob ein Element enthalten bzw. nicht enthalten ist	zahlen = [1, 5, 3, 5, 22] if 7 in zahlen:
del	Löscht ein Element bzw. Elemente entsprechend der Index-Angabe	zahlen = [1, 5, 3, 5, 22] del zahlen[3]
count(o)	Zählt das Vorkommen eines Elements o im Tupel	zahlen = [1, 5, 3, 5, 22] print[zahlen.count[5]]
index(o)	Liefert den Index des ersten Vorkommens eines Elements o im Tupel	zahlen = [1, 5, 3, 5, 22] print[zahlen.index[5]]
append()	Hängt ein Element an die Liste an	zahlen = [1, 5, 3, 5, 22] zahlen.append(15)
extend()	Hängt eine Liste an die Liste an	zahlen = [1, 5, 3, 5, 22] zahlen.extend([2, 7])
insert(i, e)	Fügt an der Stelle i ein neues Element e ein	zahlen = [1, 5, 3, 5, 22] zahlen.insert(1, 20)
pop()	Entfernt ein Element an der Stelle i - ohne Index i wird das letzte Element entfernt	zahlen = [1, 5, 3, 5, 22] zahlen.pop(2) zahlen.pop()

remove()	Entfernt das erste Element aus der Liste, das identisch ist mit e	zahlen = [1, 5, 3, 5, 22] zahlen.remove(22)
reverse()	Dreht die Reihenfolge in der Liste um	zahlen = [1, 5, 3, 5, 22] zahlen.reverse()
sort()	Sortiert die Elemente in der Liste aufsteigend	zahlen = [1, 5, 3, 5, 22] zahlen.sort()
sort(reverse = True)	Sortiert die Elemente in der Liste absteigend	zahlen = [1, 5, 3, 5, 22] zahlen.sort(reverse=True)

Module

from ... import ...

z.B. from datetime import *

Bedeutung: Vom Modul datetime wird alles (*) importiert.

Ein- und Ausgabe

```
a = input("Text")      #Eingabe der Variablen a
print "Text", a        #Ausgabe von Text und Wert der Variablen a
```

Hinweise:

- **input** liefert immer eine Zeichenkette
- Zahleneingaben sind mit int oder float zu konvertieren

Formatierungsstring

f"normaler Text {Ausdruck} mehr normaler Text {weiterer Ausdruck:Format} ..."

- Ausdruck steht für Variable, Zahl, Berechnungsanweisung usw.
- der Ausdruck wird zuerst evaluiert (ausgerechnet) und dann aufbereitet
- die geschweiften Klammern werden nicht in den druckaufbereiteten Text übernommen
- Angabe der Ausgabelänge z.B. 20 f"...{variable:20}"
- Angabe der Ausrichtung
 - links f"...{variable:<20}"
 - rechts f"...{variable:>20}"
 - zentriert f"...{variable:^20}"
- Formatierungszeichen für Ganzzahlen (int)
 - vorzeichenbehaftet ausgeben f"|{var:+15}" oder f"|{var:-15}"
 - mit führenden Nullen f"|{var:+015}" oder f"|{var:-015}"
- dezimale Ausgabe (Standard) keine weitere Angabe erforderlich
- binäre Ausgabe f"|{var:+08b}"
- oktale Ausgabe f"|{var:+08o}"
- hexadezimale Ausgabe f"|{var:+08x}"
- Formatierungszeichen für Gleitpunktzahlen (float)
- Anzahl der Nachkommastellen f"|{var:+6.2f}"
- wissenschaftliche Schreibweise f"|{var:+6.2e}"

Beispiel:

```
i = 99
```

```
print (f"{i:3d}    {(1.0/i):1.6f}")
```

Ausgabe:

```
99    0.010101
```

Klassen

Klasse: Bauplan eines Objekts

Objekt: softwaretechn. Repräsentation eines Gegenstands oder Begriffs – ein Objekt wird instanziiert

Attribute: sind die Eigenschaften des Objekts

Methoden: beschreiben die Operationen, die mit dem Objekt durchgeführt werden können

Beispiel:

Definition der Klasse Fahrzeug mit

der Konstruktor-Methode `__init__()` und

der Methode `__str__()` zur Ausgabe der Attribute (Beschreibung des Objektes) und

der Methode `beschleunigen` zur Änderung der Geschwindigkeit

class Fahrzeug:

def `__init__`(self, h, m, g):

self.hersteller = h

self.modell = m

self.geschwindigkeit = 0

def `beschleunigen`(self, wert):

self.geschwindigkeit = wert

def `__str__`(self):

return f'{self.hersteller} {self.modell}: " \

f'Geschwindigkeit: {self.geschwindigkeit:3} km/h")

das Objekt mit dem Namen f1 instanziiieren

durch die Konstruktor-Methode `__init__()` werden die Attribute initialisiert

f1 = Fahrzeug("VW", "Golf", 0)

Ausgabe der Attribute – print verwendet die Methode `__str__()`

print(f1)

Geschwindigkeit auf 50 km/h erhöhen

f1.beschleunigen(50)

Ausgabe der Attribute – print verwendet die Methode `__str__()`

print(f1)

Ausgabe:

VW Golf: Geschwindigkeit: 0 km/h

VW Golf: Geschwindigkeit: 50 km/h