Cheat Sheet: Python mit TigerJython

Grundlagen und I/O

- Der Code wird über die Einrückung strukturiert. Code, der zu einer Kontrollstruktur oder Funktion gehört, muss gleichmässig eingerückt sein.
- Kommentare werden mit `#` eingeleitet und gehen bis zum Zeilenende.
- **`pass`** ist die leere Anweisung und dient als Platzhalter in Kontrollstrukturen.
- Am Ende der Anweisung (Zeile) darf ein Semikolon `;` stehen; es wird ignoriert.
- Ein Komma am Ende der **`print`**-Anweisung verhindert den Zeilenvorschub.

```
Variable = input(Frage)
Variable = inputInt(Frage) ★
Variable = inputFloat(Frage) ★
print Ausdruck
msgDlg(Ausruck) ★
```

```
# Ein Minimalprogramm
name = input("Wie heisst du?")
print "Hallo", name + "!"
```

Kontrollstrukturen

Verzweigungen Nach **`if`** können beliebig viele **`elif`** und/oder ein **`else`** folgen. Bedingungen werden bei Bedarf mit **`and`**, **`or`** oder **`not`** verknüpft. Vergleichsoperatoren sind u. a. `==` und `!=`.

```
if Bedingung:
    Code
elif Bedingung:
    Code
else:
    Code
```

```
if 0 < x < 10:
    print 0, x
elif 10 <= x < 100:
    zehner = x // 10
    einer = x % 10
    print zehner, einer
else:
    print "Ungültige Eingabe"
```

Schleifen Eine Schleife kann mit **`break`** jederzeit abgebrochen werden, **`continue`** springt zur nächsten Wiederholung.

```
while Bedingung:
    Code

for Variable in Liste:
    Code

repeat Anzahl: ★
    Code

repeat: ★
    Code
```

```
while x != 1 and x > 0:
    if x % 3 == 2:
        break
    x //= 3
```

Daten und Typen

In Python hat jeder Wert einen festen Typ. Variablen sind aber typenlos und verweisen auf Werte. Für Situationen ohne Werte kennt Python den Wert `None`.

`bool`	Wahrheitswert	`True, False`
`int`	Ganzzahl mit beliebiger Länge	`31536000, 0x1F`
`float`	Fliesskommazahl	`6.022e+23, 3.1415926`
`str`	Zeichenkette / String	`"an", 'Example\n', "X'"`
`list/tuple`	Liste oder Tupel	`[1, 2, 3], (4, 5, 6)`

Strings

- Strings werden entweder in einfache oder doppelte Anführungszeichen eingeschlossen und können Escapesequenzen wie `\\`, `\'`, `\"` oder `\n` (Zeilenvorschub) enthalten.
- `len("Abc")` ist die Länge des Strings.
- `ord("A")` liefert den ASCII-Code eines Zeichens, `chr(65)` den String zum Code.
- `"Abc".lower()` ergibt `"abc"` und `"Abc".upper()` ergibt `"ABC"`.
- `"31-Jan-1956".split("-")` ergibt `["31", "Jan", "1956"]`.
- `".".join(["31", "1", "56"])` ergibt `"31.1.56"`. In TigerJython auch `join(["31", "1", "56"], ".")` bzw. `join([31, 1, 56], ".")`. ★
- String Formatierung: `"{0} hat den Wert: {1}".format("Pi", 3.14159)`.

Mathematik

- Bei der Division wird zwischen der «genauen» Division `/` und der Ganzzahldivision `//` unterschieden. `6 / 4 = 1.5`; `6 // 4 = 1`. Divisionsrest: `6 % 4 = 2`.
- Potenzen werden mit `**` ausgedrückt: `3 ** 2 = 9` und `3 ** 0.5 = 1.732`.
- Viele Funktionen (`sqrt`, `sin`, `cos`, ...) und π (`pi`) sind im *math*-Modul, das zuerst importiert werden muss: **`import`** `math` oder **`from`** `math` **`import`** `sqrt, sin`.
- Python kann direkt mit komplexen Zahlen arbeiten: $(3 + 4j)^{**} 2 = -7 + 24j$.

```
from math import sqrt, pi
print sqrt(3)
print pi
```

```
import math
print math.sqrt(3)
print math.pi
```

★ Nur in TigerJython, nicht in Standard-Python

Listen

- Listen dürfen beliebige Typen enthalten und mischen: `[1, 2.5, "xy", (3, 4)]`.
- Tupel sind «unveränderliche Listen»: `(1, 2, "xy")`.
- Auf einzelne Elemente wird mit `liste[index]` zugegriffen. `liste[0]` ist das erste Element, `liste[-1]` gibt das letzte Element zurück.
- `liste[start:ende]` gibt eine Teilliste zurück: `"Beispiel"[1:4] = "eis"`.
- `3 in liste` prüft, ob 3 in der Liste vorkommt.
- Mit `del liste[index]` wird das Element an Stelle *index* gelöscht.
- `range(n)` erzeugt die Liste `[0, 1, 2, 3, ..., n-1]`, `range(3, n)` erzeugt die Liste `[3, 4, 5, 6, ..., n-1]`.
- `a, b = (1, 3); a, b = 1, 3` bzw. `a, b = [1, 3]` steht für `a = 1; b = 3` (sog. unpacking) und erlaubt z. B. auch `a,b = b,a` oder `a,b = a+b,a-b`.
- Listen können multipliziert werden: `[1, 2] * 3 = [1, 2, 1, 2, 1, 2]`.
- List comprehensions: `[x**2 for x in range(1, 5)]` erzeugt `[1, 4, 9, 16]`.
- List comprehensions mit Filter: `[x for x in range(1, 10) if x % 2 == 0]` erzeugt `[2, 4, 6, 8]`.

Methoden

`Liste.append(Element)`	Element am Ende anfügen
`Liste.index(Element)`	Position des Elements ermitteln
`Liste.insert(Index, Element)`	Element an Stelle Index einfügen
`Liste.remove(Element)`	Erstes Vorkommen des Elements entfernen
`Liste.sort()`	Liste sortieren

Funktionen

```
liste = [3, 1, 7, 9, 5]
len(liste)          5
sorted(liste)       [1, 3, 5, 7, 9]
max(liste)          9
min(liste)          1
sum(liste)          25
head(liste) ★       3                 liste[0]
tail(liste) ★       [1, 7, 9, 5]      liste[1:]
indices(liste) ★    [0, 1, 2, 3, 4]   range(len(liste))
```

Standard-Python:

Funktionen

Deklaration

```
def Name(Parameter):
    Code
```

- Funktionen können eine beliebig lange Liste von Parametern haben. Die Klammern sind auch nötig, wenn es keine Parameter gibt. Optionale Parameter haben bereits einen Standardwert, der in der Deklaration mit `Parameter=Wert` angegeben wird.
- `return` beendet die Funktion, `return Wert` beendet die Funktion und gibt *Wert* zurück. Eine Funktion muss kein `return` haben. Funktionen ohne Rückgabewert geben `None` zurück.
- Funktionen sind auch «Daten». Sie lassen sich an Variablen zuweisen oder als Parameter übergeben.
- Um globale Variablen zu verändern müssen diese mit `global Variable` in der Funktion deklariert werden. Ansonsten wird eine neue lokale Variable erzeugt.
- Kurze anonyme Funktionen werden mit `lambda Parameter: Ausdruck` deklariert. Bsp: `lambda x: x**2`.

```
def mini_sort(x, y):
    if x <= y:
        return x, y
    else:
        return y, x

print mini_sort(5, 2)
```

```
def mini_sort(x, y):
    if x > y:
        x, y = y, x
    return y, x

my_sort = mini_sort
print my_sort(y=2, x=5)
```

```
x = 3
def inc_x():
    global x
    x += 1
inc_x()
```

```
x = 3
def inc_x(delta = 1):
    global x
    x += delta
inc_x()
inc_x(-1)
```

```
def applyToList(aList, function):
    result = []
    for item in aList:
        result.append(function(item))
    return result
def sqr(x):
    return x**2
applyToList([1, 2, 3], sqr)
```

```
def applyToList(aList, function):
    return [function(item) for item in aList]
applyToList([1, 2, 3], lambda x: x**2)
```

★ Nur in TigerJython, nicht in Standard-Python