Python für Linguisten

Dozentin: Wiebke Petersen & Co-Dozentin: Esther Seyffarth

2. Foliensatz

Variablen, Datentypen, vordefinierte Operationen und Funktionen

Shell

- Die Python Shell meldet sich mit dem Prompt: >>>
- Jede Anweisung, die hinter den Prompt geschrieben wird, wird sofort ausgeführt.
  Beispiel:

  ```
  >>> name = "Anna"
  ```

  weist der Variablen `name` den Wert `"Anna"` zu.
- Mit `print` wird der Wert eines Ausdrucks ausgegeben.
  Beispiele:

  ```
  >>> print(4 + 3)
  7
  >>> print("wer"+ " will")
  wer will
  ```

  Was passiert, wenn man dieselben Ausdrücke ohne `print`-Anweisung eingibt?

Anweisungen

- Jede Anweisung startet in einem neuen Block.
- Ein neuer Block startet immer in einer neuen Zeile (bisher sind unsere Blöcke zumeist einzeilig).
- Diese Anweisungen kennen Sie bereits:
  - `variable_name = expression`
    weist der Variablen `variable_name` das Ergebnis der Auswertung des Ausdrucks `expression` zu.
    Beispiel: `>>> number = 5 + 7`
    weist der Variablen `number` den Wert von `5 + 7`, also 12, zu.
  - `print(expression)`
    gibt das Ergebnis der Auswertung des Ausdrucks `expression` auf dem Bildschirm aus.
    Beispiel: `>>> print(5 + 7)`
    gibt 12 aus.

Objekte & Ausdrücke

Zurück zu `>>> print(5 + 7)`:

- 5 und 7 sind Objekte in Python, sie erhalten einen festen Speicherplatz.
- Mit `>>> id(object)` können sie die interne ID des Objekts abfragen
- Testen Sie folgendes: `>>> id(2)`, `>>> x=2`, `>>> y=x`, `>>> id(x)`, `>>> id(y)`.
- Objekte können von unterschiedlichem Typ sein. 5 und 7 sind beide vom Typ `int` (integer).
- Testen Sie: `>>> type(2)`, `>>> type(2.0)`, `>>> type("two")`
- Mit `str(x)`,`int(x)`,`float(x)`,`list(x)` können sie den Typ eines Objekts ändern. Achtung, kann zu Problemen führen!
- + ist ein zweistelliger Operator. Die Typen der Inputobjekte müssen passen! Der Ausdruck `5 + 7` wird zu dem neuen Objekt 12 ausgewertet.
- + und − können auch als einstellige Operatoren verwendet werden.
- Sie können auf Klammern in Ausdrücken mit mehreren Operatoren verzichten, wenn Sie einen Ausdruck gemäß der üblichen Operatorenpräzedenz auswerten lassen wollen.
  Beispiel: `>>> 5 + 7 * -3` wird zu −16 ausgewertet.

Übersicht über die gängigen numerischen Operatoren

- `+` Addition ($3 + 2 = 5$)
- `-` Subtraktikon ($3 - 2 = 1$)
- `*` Multiplikation ($3 * 2 = 6$)
- `/` Division ($7/2 = 3.5$), Vorsicht: bei Python 2.7 müssen Sie zunächst die Anweisung `from __future__ import division` erteilen.
- `//` ganzzahlige Division ($7/2 = 3$)
- `%` Modulo, Rest bei ganzzahliger Division ($7\%2 = 1$)
- `**` Exponent ($2 * *3 = 8$)
  Achtung: ^ entspricht der Operation "bitweise XOR", hat also nichts mit Exponenten zu tun!

Operatorpräzedenz: $+, -$ < $*, /, //, \%$ < $**$ < -x,+x

Strings

- Strings sind Zeichenketten und müssen in einfache oder doppelte Anführungszeichen gesetzt werden.
  `>>> this_string = "hello"`, `>>> this_string = 'hello'`.
  Zeichenketten, die über mehrere Zeilen gehen oder andere Anführungszeichen einschließen sollen, werden von dreifachen Anführungszeichen umschlossen:

```
1    >>> x= ''' dies ist eine 'ziemlich'
2    lange
3    "Anweisung" mit ''Zeilenumbruch'' '''
4    >>> print(x)
```

- Sonderzeichen müssen escaped werden:
  Tabulator: `\t`
  Zeilenumbruch: `\n`
  Backslash: `\\`
  einfache oder doppelte Anführungszeichen: `\'`, `\"`

Stringoperationen

- Zuweisung von Strings zu Variablen: `>>> this_string = "hello"`
- Stringkonkatenation: Strings werden mit dem Operator + konkateniert (`"some" + "value"`)
- mehrfache Konkatenation mit Operator `*`: `string * int`, `int * string`
- Substringtest: `>>>'ell'in this_string`, `>>>'le'in this_string`
- Ein String wird intern über einen Index repräsentiert.

h	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

Stringoperationen: Slicing

- Ein String wird intern über einen Index repräsentiert.

h	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

- Auf den Index kann mit einer Integer in eckigen Klammern zugegriffen werden. Beispiel: `>>> this_string[0]` gibt 'h' aus
- Es kann auch ein Indexintervall angeben werden (Slicing):
  `>>> this_string[0:2]` Wichtig: **bis** drittes Element, also Index 2
- Dies kann auch als `>>> this_string[:2]` geschrieben werden.
- Testen Sie verschiedene Intervalle:
  `>>> this_string[2:], this_string[:],this_string[-4:]`...
- Bei "unmöglichen" Intervallen, z.B. `this_string[3:0]`, wird ein leerer String zurückgegeben.

Datentypen – Listen

- Eine leere Liste wird durch `>>> this_list = []` erstellt, alternativ kann auch `>>> this_list = list()` verwendet werden
- Eine Liste mit Elementen wird durch `>>> this_list = ['a','b','c']` erstellt. Alternative?
- Wie Strings verfügen Listen über einen Index: `>>> this_list[0]` gibt `'a'` zurück.

Syntax- & Semantikfehler

- Syntaxfehler treten auf, wenn die Anweisungen syntaktisch nicht wohlgeformt sind.
  Beispiel: `>>> 5*/3`
  Beispiel: `>>> 5+3) *2`
- Semantikfehler treten auf, wenn die Anweisungen zwar syntaktisch wohlgeformt sind, aber aufgrund von z.B. Objekten falschen Typs oder aufgrund einer falschen Zahl von Argumenten nicht interpretiert werden können (TypeError).
  Beispiel: `>>> 5/"hello"`
  Beispiel: `>>> type(4,5)`
- Der Python-Interpreter meldet Ihnen genau, wo und welche Art von Fehler auftritt.
- Weiteres Beispiel für einen Semantikfehler: `>>> 5/(4 % 2)` (was passiert hier?)
- Noch ein Beispiel für einen Semantikfehler: `>>> 'hello'[6]` (was passiert hier?)
- Lesen Sie bitte die Meldungen des Python-Interpreters immer und gründlich!

Funktionsaufruf

- Python bringt einige vordefinierte Funktionen mit: z.B. `round(number)`, `min(number1,number2)`, `type(object)`, `id(object)`.
- Ein Funktionsaufruf ist ein Ausdruck
  `function_name(arg1,arg2)`
  `arg1,arg2` sind die Argumente, mit denen die Funktion `function_name` aufgerufen wird.
- Jede Funktion hat eine festgelegte Stelligkeit (Zahl der Argumente).
- Bei der Auswertung eines Funktionsaufrufs werden zunächst die Argumente ausgewertet und anschließend wird die Funktion mit den sich ergebenden Werten ausgewertet. Genaugenommen sind Operatoren auch Funktionen (mit einer anderen Syntax).
- Funktionsaufrufe können als Input für andere Funktionen eingesetzt werden (die Funktionen werden verschachtelt)
  Beispiel: `type(id("hello"))`
  Beispiel: `min(round(4.3),4.2)`
  Beispiel: `round(min(4.3,4.2))`

vordefinierte Funktionen

- Mit `dir(__builtins__)` können Sie sich die vordefinierten Funktionen anzeigen lassen (es werden nicht nur Funktionen angezeigt).
- Mit `help(function_name)` erhalten Sie Hilfsinformationen zu der Funktion `function_name`
- Beim Verwenden von Funktionen können Sie sich auch am Tooltip orientieren, der in IDLE erscheint, wenn Sie nach der öffnenden Klammer der Funktion die Shift-Taste nicht sofort loslassen:

```
>>> print(
        print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Schauen Sie sich bitte die Definitionen der folgenden Funktionen an:

`len`, `round`, `abs`, `set`