```
HOWTO zu den Operatoren von Python

(C) 2016-2021 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
              OSTC Open Source Training and Consulting GmbH
              http://www.ostc.de

$Id: python-operator.txt,v 1.6 2021/07/28 08:29:35 tsbirn Exp $

Dieses Dokument beschreibt die 55 Operatoren von Python und ihren Vorrang.
Die Operatoren sind von oben nach unten mit fallendem Vorrang aufgelistet
(oben stehen die Operatoren mit hÃ¶chstem VORRANG, unten die mit niedrigstem).

Doku --> http://docs.python.org/3/reference/lexical_analysis.html#operators
     --> http://docs.python.org/3/reference/expressions.html#operator-precedence

+====+================================+=========================================+
|    | Operator                       | Beschreibung                            |
+====+================================+=========================================+
|    | ( EXPR )                       | Gruppierung (kein Komma!)               |
|    | ( EXPR,... )                   | Tupel (mind. 1 Komma notwendig)         |
| 17 | [ EXPR,... ]                   | Liste                                   |
|    | { KEY:VALUE,... }              | Dictionary                              |
|    | { EXPR,...}                    | Set                                     |
+----+--------------------------------+-----------------------------------------+
|    | X[INDEX]  X[INDEX:INDEX:STEP]  | Indizierung, Slicing                    |
| 16 | X(ARGS...)       X.ATTR        | Funktionsaufruf, Attribut-Referenz      |
|    | OBJ.X(ARGS...)                 | Methodenaufruf fÃ¼r Objekt               |
+----+--------------------------------+-----------------------------------------+
| 15 | await X                        | Await Ausdruck                   PY3.5  |
| 14 | **                             | Potenzierung ("hoch")                   |
| 13 | +X   -X   ~X                   | Vorzeichen, Bitweise NOT (Tilde)        |
+----+--------------------------------+-----------------------------------------+
|    | *   @                          | Multiplikation, Matrix-Multipl. @ PY3.5 |
| 12 | /   //                         | Division, Divison ohne Rest    // PY2.2 |
|    | %                              | Divisions-Rest (Modulo)                 |
+----+--------------------------------+-----------------------------------------+
| 11 | +   -                          | Addition, Subtraktion (Diff bei Set)    |
| 10 | <<   >>                        | Bitweise Shift nach links + rechts      |
|  9 | &                              | Bitweise AND (Schnittmenge bei Set!)    |
|  8 | ^                              | Bitweise XOR (Symm. Diff. bei Set!)     |
|  7 | |                              | Bitweise OR  (Vereinigung bei Set!)     |
+----+--------------------------------+-----------------------------------------+
|  6 | <   <=   >   >=   ==   !=      | Wert-Vergleich (Werte gleich)           |
|    | in    not in                   | Elem/String in Container/String enthalt.|
|    | is    is not                   | IdentitÃ¤ts-Vergleich (Obj. identisch)   |
+----+--------------------------------+-----------------------------------------+
|  5 | not X                          | Logisch NOT                             |
|  4 | and                            | Logisch AND                             |
|  3 | or                             | Logisch OR                              |
|  2 | EXPR1 if COND else EXPR2       | Bedingter Ausdruck                      |
|  1 | lambda EXPR                    | Lambda Ausdruck (unbenannte Funktion)   |
+----+--------------------------------+-----------------------------------------+
|  0 | :=                             | Zuweisungsausdruck ("Walrus")    PY3.8  |
+----+--------------------------------+-----------------------------------------+

* Alle Operatoren in obiger Tabelle sind BINÃM-^DR ausser ein X steht bei UNÃM-^DREN
  Operatoren (d.h. +X   -X   ~X   not X   await X)

* Vorrang: Gemischte Operatoren werden gemÃ¤ÃM-^_ der Vorrangtabelle ausgewertet:

    1 & 2 | ~3 < 1 + 2 * 4 ** 5              # ohne Klammern ergibt sich die
    ((1 & 2) | (~3)) < (1 + (2 * (3 ** 4)))  # Reihenfolge wie mit Klammern

* AssoziativitÃ¤t: ALLE Operatoren gruppieren von LINKS NACH RECHTS,
  AUSSER dem Potenz-Operator "**", er gruppiert von RECHTS NACH LINKS.

    1 + 2 + 3 + 4 + 5       # analog ((((1 + 2) + 3) + 4) + 5)
    1 - 2 - 3 - 4 - 5       # analog ((((1 - 2) - 3) - 4) - 5)
    1 * 2 * 3 * 4 * 5       # analog ((((1 * 2) * 3) * 4) * 5)
    1 / 2 / 3 / 4 / 5       # analog ((((1 / 2) / 3) / 4) / 5)
    1 ** 2 ** 3 ** 4 ** 5   # analog (1 ** (2 ** (3 ** (4 ** 5))))

* Zuweisung "=" und verkÃ¼rzte Zuweisung "OP=" sind KEINE Operatoren!
  Sie kÃ¶nnen daher auch KEIN Teil von AusdrÃ¼cken sein und haben KEINEN Vorrang!
  Durch sie wird ein Name per Referenz an ein Objekt/einen Wert gebunden.

  +------+-------------------------------------------------+
  |   =  | Zuweisung                                       |
  +------+-------------------------------------------------+
  |  +=  | Addition                                        |
  |  -=  | Subtraktion (auch Set!)                         |
  |  *=  | Multiplikation                                  |
```

/=	Division (FlieÃM-^_komma-Ergebnis)
//=	Division ohne Rest
%=	Divisions-Rest (Modulo)
**=	Potenzierung
|=	Bitweise ODER (auch Set, auch Dict seit PY3.9!)
&=	Bitweise AND (auch Set!)
^=	Bitweise XOR (auch Set!)
<<=	Bitweise Shift nach links
>>=	Bitweise Shift nach rechts
@=	Matrix-Multiplikation PY3.5

* Seit Python 3.8 ist der Operator ":=" (Assignment Expression, auch "Walrus"-Operator) als Zuweisungs-Operator in einem Ausdruck (Expression) erlaubt. D.h. folgende Konstrukte sind damit mÃ¶glich:

```
if fin := open("datei.txt", "r"):
    ...

while zeile := fin.readline():
    ...
```

* Folgende Syntax ist erlaubt/verboten:

```
a = 123                # OK --> 123
a == 123               # OK --> True (Wert wird verworten)
erg = a == 123         # OK --> True
erg == a = 123         # SyntaxError: cannot assign to comparison
erg = a + 3            # OK --> 126
erg := a + 3           # SyntaxError: invalid syntax
(erg := a + 3)         # OK --> 126
erg := (a + 3)         # SyntaxError: invalid syntax
print(erg := a + 3)    # OK --> 126
erg = erg := a + 3     # SyntaxError: invalid syntax
erg = (erg := a + 3)   # OK --> 126
erg = erg = a + 3      # OK --> 126
erg := erg = a + 3     # SyntaxError: invalid syntax
```

* HINWEIS: Neue Operatoren werden extrem selten zu Python hinzugefÃ¼gt, (entfernt werden Operatoren gar nicht). Bevor so etwas passiert, wird auf jeden Fall ein PEP erstellt, es findet eine Diskussion und eine Abstimmung/Entscheidung sowie die Festlegung der Python-Version statt, ab der dieses SchlÃ¼sselwort verfÃ¼gbar sein wird. D.h. man hat mehrere Jahre Zeit, um seinen Quellcode daran anzupassen.

* Folgende Sonderzeichen sind keine Operatoren, sondern "Delimiter":

,	Element-Trennzeichen Tuple/List/Dict, Funktion-Def/Aufruf, ...
;	Abschlusszeichen Anweisung (Ã¼berflÃ¼ssig)
:	Block-Beginn, Dict-Trennzeichen, Type Hint/Annotation VAR : TYP
->	Type Hint/Annotation Funktions-Ergebnis
.	Objekt.Attribut, Modul.SubModul.SubSubModul...
\	Zeilenfortsetzung (Line Continuation, Zeile in nÃ¤chster Zeile weiter)
#	Kommentarbeginn (bis Zeilenende)

* String-Begrenzer sind:

" '	String-Begrenzer (nur einzeilig)
""" '''	String--Begrenzer (mehrzeilig erlaubt)
b" b'	BinÃ¤r-String b"deadbeefaffe..."
f" f'	Format-String f"...{...}..."
r" r'	Raw-String (Escape-Sequenzen \X "as is")
u" u'	Unicode-String (Std in PY3, notwendig in PY2)
br" rb"	BinÃ¤r + Raw
fr" rf"	Format + Raw