HOWTO zu den Booleschen Wahrheitswerten in Python

$Id: python-boolean.txt,v 1.3 2020/03/31 00:59:56 tsbirn Exp $

Dieses Dokument beschreibt die Boolesche Logik der Werte in Python,
insbesondere die Booleschen Werte True und False sowie den Wert None.

INHALTSVERZEICHNIS

---

1) Boolesche Werte

Doku --> http://docs.python.org/3/library/stdtypes.html#boolean-values
http://docs.python.org/3/library/stdtypes.html#truth

Python kennt einen Booleschen Datentyp "bool" mit den beiden Werten "True" und "False", aber interpretiert auch JEDEN anderen Wert JEDES Typs (NoneType, int, long, float, complex, str, tuple, list, dict, ...) als logischen Wert. Dabei ist exakt definiert, welche Werte als logisch "False" und welche als logisch "True" interpretiert werden. Logisch "falsch" sind GENAU die folgenden 13/14 Werte (Konstanten definiert als "falsch", Wert 0 jedes numerischen Datentyps, leere Sequenzen und Collections)

Wert	Beschreibung
None	Undefinierter/ungÃ¼ltiger Wert
False	Logisch falscher Wert
0	Ganzzahl Null
0.0	FlieÃM-^_kommazahl Null
0l 0L	Ganzzahl Null (Typ "long", nur Python 2)
0+0j 0+0J	Komplexe Zahl Null (Realteil + ImaginÃ¤rteil)
Decimal(0)	Dezimalzahl 0 (interessant fÃ¼r Finanzbuchhaltung)
Fraction(0,1)	Bruch 0/1
""	Leere Zeichenkette (natÃ¼rlich auch '' r"" r'' ...)
()	Leerer Tupel
[]	Leere Liste
{}	Leeres Dictionary
set()	Leere Menge
range(0)	Leerer Wertebereich

ALLE ANDEREN Werte werden als logisch "True" interpretiert, z.B. folgende (Auswahl):

Wert	Bedeutung
True	Logisch wahrer Wert
1	Zahl 1
-1	Negative Zahl -1
0.1	FlieÃM-^_kommazahl 0.1
Decimal("0.1")	Dezimalzahl 0.1
Fraction(1,2)	Bruch 1/2
" "	Zeichenkette aus 1 Leerzeichen
" "	Zeichenkette aus 5 Leerzeichen
"\n"	Zeichenkette aus 1 Newline
"\r"	Zeichenkette aus 1 Carriage Return
"\t"	Zeichenkette aus 1 Tabulator
"None"	Zeichenkette aus Text "None"

"False"	Zeichenkette aus Text "False"
"0"	Zeichenkette aus 1 Zeichen "0"
"00"	Zeichenkette aus 2 Zeichen "00"
"+0"	Zeichenkette aus 2 Zeichen "+0"
"-0"	Zeichenkette aus 2 Zeichen "-0"
"0.0"	Zeichenkette aus 3 Zeichen "0.0"
"abc"	Beliebige Zeichenkette ab 1 Zeichen LÃ¤nge
(0,)	Tupel mit mind. 1 Wert
[0]	Liste mit mind. 1 Wert
{0:0}	Dictionary mit mind. 1 Wert
{0}	Set mit mind. 1 Wert
range(1)	Set mit mind. 1 Wert

1b) Boolescher Kontext in if/while

Die Anweisungen "if" und "while" erzwingen einen "Booleschen Kontext", in dem der Wert der steuernden Ausdrucks EXPR (Bedingung) als Boolescher Wert interpretiert wird:

```
if EXPR: ...       # entspricht if bool(EXPR): ...
while EXPR: ...    # entspricht while bool(EXPR): ...
```

1c) Zusammenhang mit Zahlen-Datentypen int/float/complex

Der Datentyp "bool" ist eine Spezialisierung des Datentyps "int", d.h. die Booleschen Werte "True/False" werden in RechenausdrÃ¼cken als 1/0 interpretiert. In Listen/Tupeln werden sie als Index 1/0 interpretiert. Ebenso werden sie bei Dictionaries/Sets mit den SchlÃ¼sseln 1/0 (int), 1.0/0.0 (float) und 1+0j/0+0j (complex) identifiziert.

2) Logische Operatoren

Die logischen Operatoren "and", "or" und "not" liefern angewendet auf Wahrheitswerte (oder beliebige andere Werte, die dann gemÃ¤Ã-^_ obiger Tabelle als Wahrheitswerte interpretiert werden) folgende Resultate:

Operator	Bedeutung
A and B	UND: Resultat "True" wenn BEIDE Operanden "True" Resultat "False" sonst
A or B	ODER: Resultat "False" wenn BEIDE Operanden "False" Resultat "True" sonst
not A	NICHT: Resultat "True" wenn Operand "False" Resultat "False" wenn Operand "True"

Bei Kombination von "and", "or" und "not" in einem Ausdruck gilt folgender Vorrang (wie in allen anderen Programmiersprachen auch):

not	hÃ¶chster
and	mittlerer
or	niedrigster

Die Booleschen Werte "True" und "False" werden bei der Ausgabe per "print(...)" als "True" und "False" dargestellt.

```
print(True)    # --> True
print(False)   # --> False
```

Die Booleschen Werte "True" und "False" werden bei formatierter Ausgabe per "print(...)" als "1" und "0" dargestellt.

```
print("%d" % True)            # --> 1
print("%d" % False)           # --> 0
print("{:d}".format(True))    # --> 1
print("{:d}".format(False))   # --> 0
print("{:s}".format(True))    # --> ValueError: Unknown format code 's' for
print("{:s}".format(False))   # -->             for object of type 'bool'
```

Der logische Operator "not" liefert IMMER "True" oder "False" zurÃ¼ck:

```
print(not True)    # --> False
print(not False)   # --> True
print(not "abc")   # --> False
```

```
  print(not "")      # --> True
  print(not 0)       # --> True
  print(not 1)       # --> False
  print(not [])      # --> True

Die logischen Operatoren "and" und "or" liefern IMMER den LETZTEN fÃ¼r das
Ergebnis ausgewerteten Wert (interpretierbar als Wahrheitswert) zurÃ¼ck:

  print(5 or "abc")   # --> 5    = True
  print(0 or "abc")   # --> abc  = True
  print(0 or "")      # -->      = False (leerer String)
  print(1 and 234)    # --> 234  = True
  print(9 and None)   # --> None = False
  print(0 and [])     # --> 0    = False

HINWEIS: Die aus C/C++/... bekannten Syntaxformen "&& || !" fÃ¼r die Booleschen
Operatoren "and or not" gibt es in Python nicht! Die Operatoren "& | ~" fÃ¼r die
bitweisen Operationen "and or not" gibt es hingegen schon!


3) AusdrÃ¼cke mit logischen Werten
---------------------------------
In AusdrÃ¼cken zÃ¤hlt "True" als Zahl 1 und "False" als Zahl 0.
D.h. Berechnungen mit Booleschen Werten sind erlaubt:

  1 - True               # --> 0
  False + 1              # --> 1
  True * True            # --> 1
  True * False           # --> 0
  5 * True - 3 * False   # --> 5
  tage = monat_laenge(mm) + schaltjahr(jahr)     # schaltjahr --> True/False


4) Short Cut/Circuit Evaluation
-------------------------------
Die logische Operatoren "and" und "or" werden VON LINKS NACH RECHTS ausgewertet
(unter BerÃ¼cksichtigung ihres Vorrangs). Ergibt sich dabei ein
Zwischenergebnis, bei dem auch das Endergebnis des gesamten logischen Ausdrucks
feststeht, wird die Auswertung an dieser Stelle abgebrochen und der Rest des
logischen Ausdrucks nicht mehr ausgewertet ("Short Cut Evaluation", "Short
Circuit Evaluation", verkÃ¼rzte Auswertung). Dieses Verhalten ist KEINE
Besonderheit von Python, sondern ist in vielen Programmiersprachen (z.B. C,
C++, Java, JavaScript, Perl, PHP, Ruby, Awk) genauso Ã¼blich.

* Folge von UND-VerknÃ¼pfungen ("and") bricht mit Gesamtergebnis "False" ab,
  sobald 1x "False" als Zwischenergebnis vorkommt.
* Folge von ODER-VerknÃ¼pfungen ("or") bricht mit Gesamtergebnis "True" ab,
  sobald 1x "True" als Zwischenergebnis vorkommt.

Beispiel:

  0 and print("KEINE Ausgabe")    # --> 0
  1 and print("Ausgabe")          # --> Ausgabe
  1 or  print("KEINE Ausgabe")    # --> 0
  0 or  print("Ausgabe")          # --> Ausgabe
  print(5 or "abc" or 7.5)       # --> 5    = True
  print(0 or "abc" or 7.5)       # --> abc  = True
  print(0 or "" or 7.5)          # --> 7.5  = True
  print(1 and "" and 3.14)       # -->      = False (leerer String "")
  print(1 and 234 and 74)        # --> 74   = True
  print(9 and None and 8)        # --> None = False
  print(0 or [] or {})           # --> {}   = False


5) Undefinierter Wert "None" versus definierte Werte
----------------------------------------------------
Ein ungÃ¼ltiger Wert kann in Python durch "None" dargestellt werden (hat den Typ
"NoneType" bzw. ab PY3.7 type(None)). Variablen mit diesem Wert sind
UNDEFINIERT (analog SQL-Wert "NULL"). ALLE anderen Werte sind DEFINIERT.

Der Typ "NoneType" und sein einziger Wert "None" kÃ¶nnen eigentlich miteinander
identifiziert werden. Ab PY3.7 gibt es daher den Bezeichner "NoneType" nicht
mehr, er kann ersatzweise durch type(None) dargestellt werden:

  NoneType = type(None)   # ab PY3.7

Folgende Tests prÃ¼fen, ob der Wert einer Variablen "var" DEFINIERT bzw.
UNDEFINIERT ist:

  if type(var) == NoneType: ...        # var UNDEFINIERT?
  if type(var) == type(None): ...      # var UNDEFINIERT? (ab PY3.7)
  if isinstance(var, NoneType): ...    # var UNDEFINIERT?
```

```
  if isinstance(var, type(None): ...   # var UNDEFINIERT? (ab PY3.7)
  if var is None: ...                  # var UNDEFINIERT?
  if var == None: ...                  # var UNDEFINIERT?

Berechnungen mit "None" generieren IMMER einen "TypeError":

  None + 3   # --> TypeError!
  None * 3   # --> TypeError!
  None / 3   # --> TypeError!
  None - 3   # --> TypeError!

Vergleiche mit "None" als Wert sind nur fÃ¼r Gleichheit/Ungleichheit mÃ¶glich,
fÃ¼r den Vergleich "==" mit "None" liefert nur "None" den Wahrheitswert "True".

  None == None   # --> True
  None == True   # --> False
  None != 123    # --> True
  None < ""      # --> TypeError!
  None > 3.14    # --> TypeError!
  None <= []     # --> TypeError!
  None >= 3+3j   # --> TypeError!


6) Test auf "True" oder "False"
-------------------------------
Folgende Tests prÃ¼fen, ob ein Variablen-Wert "True" bzw. "False" ist (nur die
beiden ersten Tests sind wirklich allgemein gÃ¼ltig, die anderen Tests prÃ¼fen
nur EINEN Wert, aber nicht alle Werte gemÃ¤Ã-^_ Tabelle --> 1) Boolesche Werte.

  if VAR:               # VAR True? (GUT)
  if not VAR:           # VAR False? (GUT)
  if VAR == True:       # IMMER "False" auÃ-^_er VAR hat Wert "True" (SCHLECHT!)
  if VAR == False:      # IMMER "False" auÃ-^_er VAR hat Wert "False" (SCHLECHT!)
  if VAR != True:       # IMMER "True" auÃ-^_er VAR hat Wert "True" (SCHLECHT!)
  if VAR != False:      # IMMER "True" auÃ-^_er VAR hat Wert "False" (SCHLECHT!)
  if VAR is True:       # IMMER "False" auÃ-^_er VAR hat Wert "True" (SCHLECHT!)
  if VAR is False:      # IMMER "False" auÃ-^_er VAR hat Wert "False" (SCHLECHT!)
  if VAR is not True:   # IMMER "True" auÃ-^_er VAR hat Wert "True" (SCHLECHT!)
  if VAR is not False:  # IMMER "True" auÃ-^_er VAR hat Wert "False" (SCHLECHT!)


6.1) True == 1/1.0 und False == 0/0.0
-------------------------------------
Aus historischen GrÃ¼nden (der Datentyp "bool" wurde erst spÃ¤ter als Subtyp
von "int" hinzugefÃ¼gt), liefern folgende Vergleiche alle den Wert "True"
(obwohl die Datentypen der verglichenen Werte unterschiedlich sind):

  True  == 1     # --> True
  True  == 1.0   # --> True
  False == 0     # --> True
  False == 0.0   # --> True

HINWEIS: Nur jeweils einer der folgenden SchlÃ¼ssel kann in einem Dictionary
vorhanden sein (die Zahlen 0 und 0.0 sowie 1 und 1.0 werden ebenfalls als
gleich betrachtet):

  True    1   1.0   1+0j   1.0+0.0j
  False   0   0.0   0+0j   0.0+0.0j

In RechenausdrÃ¼cken verhalten sich "True" und "False" wie die Werte 1 und 0:

  5 * True       # --> 5
  4 + False      # --> 4


7) Test auf leeren/gefÃ¼llten Container
--------------------------------------
Da JEDES Objekt im Booleschen Kontext einen Wahrheitswert ergibt, kann ein Test
auf leeren/gefÃ¼llten Container (str, tuple, list, dict, set, ...) folgendermaÃ-^_en
durchgefÃ¼hrt werden:

  if CONT:            # CONTAINER enthÃ¤lt mind. 1 Element  --> True/False
  if not CONT:        # CONTAINER leer                     --> True/False
  if len(CONT) != 0:  # CONTAINER enthÃ¤lt mind. 1 Element  --> True/False
  if len(CONT) == 0:  # CONTAINER leer                     --> True/False
  if TUPEL != ():     # Tupel enthÃ¤lt mind. 1 Element      --> True/False
  if TUPEL == ():     # Tupel leer                         --> True/False
  if LISTE != []:     # Liste enthÃ¤lt mind. 1 Element      --> True/False
  if LISTE == []:     # Liste leer                         --> True/False
  if DICT != {}:      # Dictionary enthÃ¤lt mind. 1 Eintrag --> True/False
  if DICT == {}:      # Dictionary leer                    --> True/False
  if STR != "":       # String enthÃ¤lt mind. 1 Zeichen     --> True/False
```

```
  if STR == "":       # String leer                          --> True/False

HINWEIS: Die beiden Varianten sind vom Standpunkt der Performance und des
Speicherverbrauchs aus die Geschicktesten!
```