(C) 2024 OSTC GmbH (http://www.ostc.de)

HOWTO zu den Booleschen Wahrheitswerten in 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-boolean.txt,v 1.4 2021/07/28 08:29:35 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
1b) Boolescher Kontext
1c) Zusammenhang mit Zahlen-Datentypen int/float/complex
2) Logische Operatoren
3) AusdrÃ¼cke mit logischen Werten
4) Short Cut/Circuit Evaluation
5) Undefinierter Wert None versus definierte Werte
6) Test auf True oder False
6.1) True == 1/1.0 und False == 0/0.0
7) Test auf leeren/gefÃ¼llten Container
8) 3-wertige Logik (analog SQL)

---

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 "falsch" und welche als logisch "wahr" interpretiert werden. Logisch "falsch" sind GENAU die folgenden 13/14 Werte (die Konstanten "falsch" und "undefiniert", 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 (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

HINWEIS: In JEDER Programmiersprache gibt es eine EXAKTE DEFINITION, welche Werte als logisch "Falsch" und welche als logisch "Wahr" interpretiert werden. Ã-^\blich ist, dass JEDER Wert IMPLIZIT als Boolescher Wert interpretiert wird. EXPLIZIT erfolgt eine Konvertierung in einen Booleschen Wert durch:

```
!!WERT         # 2x-ige Negation (C, C++, Java, ...)
not not WERT   # 2x-ige Negation (Python, SQL, ...)
bool(WERT)     # Konvertierung in Python
```

1b) Boolescher Kontext in if/while

Die Anweisungen "if" und "while" erzwingen einen "Booleschen Kontext", in dem der Wert der steuernden Ausdrucks EXPR (Expression/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 in 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 A und B True Resultat False sonst
A or B	ODER: Resultat False wenn beide Operanden A und B False Resultat True sonst
not A	NICHT: Resultat True wenn Operand A False Resultat False wenn Operand A 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

Der Wert None und die Booleschen Werte True und False werden bei der Ausgabe per print(...) als Text "None, "True" und "False" dargestellt.

```
print(None)    # --> "None"
print(True)    # --> "True"
print(False)   # --> "False"
```

Die Booleschen Werte True und False werden bei formatierter Ausgabe als 1 und 0 dargestellt.

```
print("%d" % True)            # --> 1
print("%d" % False)           # --> 0
```

```
  print("%d" % None)            # --> TypeError: %d format: a number is required, not NoneType
  print("{:d}".format(True))    # --> 1
  print("{:d}".format(False))   # --> 0
  print("{:d}".format(None))    # --> TypeError: unsupported format string passed to NoneType.__form
at__
  print(F"{True:d}")            # --> 1
  print(F"{False:d}")           # --> 0
  print(F"{None:d}")            # --> TypeError: unsupported format string passed to NoneType.__form
at__
  print("{:s}".format(True))    # --> ValueError: Unknown format code 's'
  print("{:s}".format(False))   # -->              for object of type 'bool'
  print("{:s}".format(None))    # --> TypeError: unsupported format string passed to NoneType.__form
at__

Der logische Operator not liefert IMMER True oder False zurÃ¼ck:

  not True    # --> False
  not False   # --> True
  not "abc"   # --> False
  not ""      # --> True
  not 0       # --> True
  not 1       # --> False
  not []      # --> True

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

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

HINWEIS: Die aus C/C++/... bekannten Syntaxformen && || ! fÃ¼r die Booleschen
Operatoren gibt es in Python nicht! Die Operatoren & | ~ ^ << >> fÃ¼r die
bitweisen Operationen and or not xor shift left/right 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 "KEINE Ausgabe"   # --> 0         = False
  1 and "Ausgabe"         # --> "Ausgabe" = True
  1 or  "KEINE Ausgabe"   # --> 1         = True
  0 or  "Ausgabe"         # --> "Ausgabe" = True
  5 or "abc" or 7.5       # --> 5         = True
  0 or "abc" or 7.5       # --> "abc"     = True
  0 or "" or 7.5          # --> 7.5       = True
  1 and "" and 3.14       # --> ""        = False (leerer String)
  1 and 234 and 74        # --> 74        = True
  9 and None and 8        # --> None      = False
  0 or [] or {}           # --> {}        = False (leeres dict)
```

```
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 (Singleton). Ab PY3.7 gibt es daher den Bezeichner
"NoneType" nicht mehr, er wird ersatzweise durch type(None) dargestellt:

  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 != None   # --> 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:        # --> False auÃ-^_er VAR hat Wert True/1/1.0 (SCHLECHT)
  if VAR == False:       # --> False auÃ-^_er VAR hat Wert False/0/0.0 (SCHLECHT)
  if VAR != True:        # --> True auÃ-^_er VAR hat Wert True (SCHLECHT)
  if VAR != False:       # --> True auÃ-^_er VAR hat Wert False (SCHLECHT)
  if VAR is True:        # --> False auÃ-^_er VAR hat Wert True (SCHLECHT)
  if VAR is False:       # --> False auÃ-^_er VAR hat Wert False (SCHLECHT)
  if VAR is not True:    # --> True auÃ-^_er VAR hat Wert True (SCHLECHT)
  if VAR is not False:   # --> 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
  True  == 1+0j  # --> True
  1     == 1.0   # --> True
  1     == 1+0j  # --> True
  1.0   == 1+0j  # --> True
  False == 0     # --> True
  False == 0.0   # --> True
  False == 0j    # --> True
  0     == 0.0   # --> True
  0     == 0j    # --> True
  0.0   == 0j    # --> True

HINWEIS: Nur jeweils einer der folgenden SchlÃ¼ssel kann in einem Dictionary
vorhanden sein (die Zahlen 0 und 0.0 und 0j sowie 1 und 1.0 und 1+0j 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, ...) folgendermaÃM-^_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 ersten Varianten sind vom Standpunkt der Performance und des Speicherverbrauchs aus die Geschicktesten!

8) 3-wertige Logik (analog SQL)

Die 3 Werte True, False und None bilden bgzl. der Operatoren and, or und not eine 3-wertige Logik (analog SQL TRUE, FALSE, NULL), wobei None bei der Auswertung als False zÃ¤hlt und daher als linker Wert von and und als rechter Wert von or erhalten bleibt bzw. bei not zu True wird (* = Abweichung).
(and, or verhalten sich NICHT kommutativ = un-symmetrisch).

* and: True links ergibt rechten Wert, False oder None links ergibt linken Wert
* or: True links ergibt True, False oder None links ergibt rechten Wert
* not: True ergibt False, False oder None ergibt True

a	b	and	or	not
T	T	T	T	F
T	F	F	T	F
T	N	N	T	F
F	T	F	T	T
F	F	F	F	T
F	N	F	N	T
N	T	N	T	T*
N	F	N*	F*	T*
N	N	N	N	T*
a	b	and	or	not

Die 3-wertige SQL-Logik mit TRUE, FALSE, NULL verhÃ¤lt sich etwas anders (NULL zÃ¤hlt nicht als FALSE, sondern als "undefiniert" und ist somit Ergebnis, falls kein anderweitiges eindeutiges Ergebnis mÃ¶glich ist (* = Abweichung)
(and, or, xor verhalten sich kommutativ = symmetrisch).

* AND: TRUE links ergibt rechten Wert, FALSE links oder rechts ergibt FALSE, sonst NULL
* OR: FALSE links ergibt rechten Wert, TRUE links oder rechts ergibt TRUE, sonst NULL
* XOR: Ein NULL ergibt NULL, zwei TRUE oder zwei FALSE ergeben FALSE, sonst TRUE
* NOT: NULL ergibt NULL, TRUE ergibt FALSE, FALSE ergibt TRUE

a	b	and	or	not	xor
T	T	T	T	F	F
T	F	F	T	F	T
T	N	N	T	F	N
F	T	F	T	T	T
F	F	F	F	T	F

(C) 2024 OSTC GmbH (http://www.ostc.de)

F	N	F	N	T	N
N	T	N	T	N*	N
N	F	F*	N*	N*	N
N	N	N	N	N*	N
a	b	and	or	not	xor