HOWTO zu Booleschen Wahrheitswerten in Perl

$Id: perl-true-false-HOWTO.txt,v 1.14 2025/02/23 20:14:55 tsbirn Exp $

Dieses Dokument beschreibt die Logik der Booleschen Werte TRUE/FALSE sowie des UNDEFINIERTEN Werts "undef" in Perl und ihre Verknuepfung mit den Booleschen Operatoren AND, OR, NOT und XOR.

INHALTSVERZEICHNIS

---

1) Boolesche Werte

Perl kennt keinen Booleschen Datentyp mit den Werten "TRUE" und "FALSE", sondern interpretiert JEDEN Wert (undef, Skalar, Array, Hash, Referenz) als logischen Wert. Dabei ist exakt definiert, welche Werte als logisch FALSE und welche als logisch TRUE interpretiert werden. Logisch FALSE sind GENAU die folgenden 5 Werte:

Wert	Beschreibung
undef	Undefinierter Wert
""	Leere Zeichenkette (natürlich auch '', q//, qq//)
0	Zahl Null (auch +0, -0 und 0.0)
"0"	Zeichenkette bestehend aus Zeichen Null (auch '0', Sonderfall!)
()	Leere Liste (natürlich auch qw//)

Sonderfall: Die Zeichenkette "0" wird auch als logisch FALSE interpretiert, weil sie nicht von der Zahl 0 unterscheidbar ist, falls Perl automatisch zwischen Strings und Zahlen hin- und herkonvertiert.

ALLE ANDEREN Werte werden als logisch TRUE interpretiert, insbesondere folgende:

Wert	Bedeutung
" "	Zeichenkette bestehend aus 1 Leerzeichen
"\n"	Zeichenkette bestehend aus 1 Newline
"\r"	Zeichenkette bestehend aus 1 Carriage Return
"+0"	Zeichenkette bestehend aus 2 Zeichen
"-0"	Zeichenkette bestehend aus 2 Zeichen
"00"	Zeichenkette bestehend aus 2 Null-Zeichen
"0.0"	Zeichenkette bestehend aus 3 Zeichen
-1	Negative Zahl
{}	Referenz auf leeren Hash
[]	Referenz auf leeres Array

Es gibt keine expliziten logischen Werte TRUE und FALSE, sondern TRUE wird oft als Zahl "1" und FALSE wird oft als "" (leere Zeichenkette) dargestellt (zumindest wenn man den Wert 2x per not negiert):

```
  $true  = 1;
  $false = "";
  print($true, not $true, $true);       # --> 11    (1 "" 1)
  print($false, not $false, $false);    # --> 1     ("" 1 "")
```

2) Logische Operatoren

Die logischen Operatoren &&, ||, ! sowie and, or, not, xor liefern angewendet auf Wahrheitswerte (oder andere Werte, die dann als Wahrheitswerte interpretiert werden) folgende Resultate:

Operator	Alternativ	Bedeutung
A && B	A and B	Resultat TRUE wenn beide Operanden TRUE
A || B	A or B	Resultat TRUE wenn einer der beiden Operanden TRUE
! A	not A	Resultat TRUE wenn Operand FALSE (negieren)
	A xor B	Resultat TRUE wenn ein Operand TRUE und einer FALSE

TRUE wird zwar oft als "1" und FALSE wird oft als "" (leere Zeichenkette) dargestellt, letztlich liefern die logischen Operatoren aber immer den LETZTEN AUSGEWERTETEN Wert als Ergebnis zurück:

```
  0 or "abc"     # --> "abc" = TRUE
  0 or ()        # --> ()    = FALSE
  1 and 234      # --> 234   = TRUE
  9 and undef    # --> undef = FALSE
  1 xor 234      # --> ""    = FALSE    # TODO
  9 xor undef    # --> undef = TRUE     # TODO
  undef xor 9    # --> 9     = TRUE     # TODO
  "" xor 0       # --> ""    = FALSE    # TODO
```

Der Vorrang von && und || liegt NACH den Zuweisungoperatoren (d.h. Klammern sind notwendig, wenn das Ergebnis zugewiesen werden soll). Der Vorrang von "and", "or", "not" und "xor" liegt NACH allen anderen Operatoren (auch nach ","), d.h. ist der geringst mögliche. Einfaches Austauschen von && und "and" bzw. || und "or" bzw. ! und "not" führt daher oft zu einer falschen Auswertungsreihenfolge!

```
  $var = 1 and 0          # TODO
  $var = (1 and 0)        # TODO
  $var = 0 or 1           # TODO
  $var = (0 or 1)         # TODO
  $var = 1 xor 2          # TODO
  $var = 0 xor 1          # TODO
  $var = not 1            # TODO
  $var = 1 && 2           # TODO
  $var = 0 && 1           # TODO
  $var = 0 || 1           # TODO
  $var = ! 1              # TODO
  $var = ! 0              # TODO
```

3) Short cut/circuit evaluation

Logische Operatoren werden von links nach rechts ausgewertet (unter Berücksichtigung des Vorrangs). Ergibt sich dabei ein Zwischenergebnis, bei dem das Endergebnis des gesamten logischen Ausdrucks sofort feststeht, wird die Auswertung an dieser Stelle sofort abgebrochen und der Rest des logischen Ausdrucks nicht mehr ausgewertet (sogenannte "Short cut evaluation", "Short circuit evaluation", verkürzte Auswertung). Dieses Verhalten ist keine Perl-Besonderheit, sondern ist in vielen Programmiersprachen (z.B. C, C++, Java, JavaScript, PHP, Python, Ruby, Awk) zu beobachten.

Insbesondere wird eine Folge von UND-Verknüpfungen ("and", "&&") mit dem Gesamtergebnis FALSE abgebrochen, sobald 1x FALSE als Zwischenergebnis vorkommt. Eine Folge von ODER-Verknüpfungen ("or", "||") wird mit dem Gesamtergebnis TRUE abgebrochen, sobald 1x TRUE als Zwischenergebnis vorkommt. Eine Folge von XOR-Verknüfungen wird nicht abgebrochen, da das Endergebnis von ALLEN Werten abhängt.

Beispiel:

```
  0 and print "Nicht ausgegeben";         # --> 0 und nichts ausgegeben
  1 or  print "Auch nicht ausgegeben";    # --> 1 und nichts ausgegeben
  0 or "abc" or 3.14                      # --> "abc"
  1 and "" and 3.14                       # --> ""
  0 xor "abc" xor 3.14                    # --> ""   TODO
  0 xor "abc" xor 0                       # --> 0   TODO
  1 xor "abc" xor 2                       # --> 0   TODO
  2 xor "abc" xor 1                       # --> 0   TODO
  1 xor "" xor 3.14                       # --> "3.14"   TODO
```

4) "undef" versus definierte Werte

Nicht initialisierte Variablen haben in Perl den Wert "undef" und sind somit UNDEFINIERT (analog SQL-Wert "NULL"). ALLE anderen Werte sind DEFINIERT und können mit dem Operator "defined" vom Wert "undef" unterschieden werden.

Wert	TRUE	FALSE	defined	!defined
undef	Nein	Ja	Nein	Ja
""	Nein	Ja	Ja	Nein
0	Nein	Ja	Ja	Nein
"0"	Nein	Ja	Ja	Nein
()	Nein	Ja	Ja	Nein
" "	Ja	Nein	Ja	Nein
"\n"	Ja	Nein	Ja	Nein
"\r"	Ja	Nein	Ja	Nein
"+0"	Ja	Nein	Ja	Nein
"-0"	Ja	Nein	Ja	Nein
"00"	Ja	Nein	Ja	Nein
"0.0"	Ja	Nein	Ja	Nein
-1	Ja	Nein	Ja	Nein
{}	Ja	Nein	Ja	Nein
[]	Ja	Nein	Ja	Nein

Folgende Tests prüfen, ob der Wert einer Variablen $var DEFINIERT bzw. UNDEFINIERT ist:

```
  if (defined $var)     { ... }       # $var DEFINIERT?
  if (not defined $var) { ... }       # $var UNDEFINIERT?
  if (! defined $var)   { ... }       # $var UNDEFINIERT?
  unless (defined $var)   { ... }     # $var UNDEFINIERT?
```

Folgende Tests prüfen, ob der Wert einer Variablen $var TRUE bzw. FALSE ist:

```
  if ($var)     { ... }      # $var TRUE?
  if (not $var) { ... }      # $var FALSE?
  if (! $var)   { ... }      # $var FALSE?
  unless ($var)   { ... }    # $var FALSE?
```

5) Spezielle Operatoren

```
----------------------
```
Der Operator "//" (defined-or) prüft, ob der linke Wert DEFINIERT ist: Wenn ja,
gibt er ihn zurück, sonst wird der rechte Wert zurückgegeben:

```
  $var = $ARGV[0] // "";       # -->              TODO
  $var = undef // "default";   # --> "default"    TODO
  $var = 1 // "default";       # --> 1            TODO
```

Der Operator "||" (or) prüft, ob der linke Wert TRUE ist: Wenn ja, gibt er ihn
zurück, sonst wird der rechte Wert zurückgegeben:

```
  $var = $ARGV[0] || "1";      # --> "default"    TODO
  $var = undef || "default";   # --> "default"    TODO
  $var = 0 || "default";       # --> "default"    TODO
  $var = "" || "default";      # --> "default"    TODO
  $var = 1 || "default";       # --> "default"    TODO
```

Von beiden Operatoren gibt es auch die verkürzte Form "//=" und "||=":

```
  $var //= 123                 # --> 123 falls $var undefiniert, sonst $var unverändert
  $var ||= 123                 # --> 123 falls $var falsch, sonst $var unverändert
```