

## Escape-Sequenzen in Python-Strings

(C) 2016-2020 T.Birnthaler OSTC GmbH

Doku --> [http://docs.python.org/3/reference/lexical\\_analysis.html#literals](http://docs.python.org/3/reference/lexical_analysis.html#literals)

Das Zeichen "\" (Backslash) dient in Python innerhalb von Strings als "Fluchtzeichen" (Escape-Zeichen), das zusammen mit dem DIREKT FOLGENDEN Zeichen eine spezielle Bedeutung hat. Darüber werden z.B. Steuerzeichen (Strg-Z) und Unicode-Zeichen über ihre Codierung oder ihren Namen angesprochen.

Folgende Kombinationen sind möglich:

Zeichen	Bedeutung	
\\	Backslash	[backslash]
\'	Hochkomma (in '...' notwendig)	[single quote]
\"	Gänsefüßchen (in "... " notwendig)	[double quote]
\a	Klingel (BEL)	[alert]
\b	Rückschritt (BS)	[backspace]
\e	Escape (ESC)	[escape]
\f	Seitenvorschub (FF)	[form feed]
\n	Zeilenumbruch (LF)	[newline, linefeed]
\r	Wagenrücklauf (CR)	[carriage return]
\t	Horizontaler Tabulator (HT)	[tabulator]
\v	Vertikaler Tabulator (VT)	[vertical tabulator]
\0	Null Byte (NUL)	[null]
\OOO	Zeichen mit Oktalcode "OOO" (000-777)	[octal]
\xHH	Zeichen mit Hexadezimalcode "HH" (00-ff)	[hexadezimal]
\N{name}	Zeichen mit Unicode-Name "name"	[Name]
\uXXXX	Zeichen mit 16-bit Unicode-Wert "XXXX"	[unicode]
\UXXXXXXXX	Zeichen mit 32-bit Unicode-Wert "XXXXXXXX"	[Unicode]
<NEWLINE>	Zeilenende wird ignoriert	[line continuation]
<SONST>	Bleibt stehen inkl. Backslash	

## HINWEISE:

- \* Innerhalb von RAW-STRINGS r"..." werden Escape-Sequenzen nicht interpretiert sondern bleiben stehen.
- \* Ein Backslash am Zeilenende ("Line Continuation") verknüpft die aktuelle Zeile mit der folgenden zu einer "logischen Zeile" und erlaubt so eine beliebige Unterteilung von langen Zeilen in kurze Stücke. Die Einrückung der Folgezeile ist dabei irrelevant und muss nicht den Einrückungsregeln von Python gehorchen.

```
erg = 1 + \\
      2 + \\
      3
```

Zeichenketten werden durch folgende Syntax definiert (quotiert):

Syntax		Bedeutung
"..."	E 1	Alle Zeichen (außer ^_er " und Zeilenvorschub)
'...'	E 1	Alle Zeichen (außer ^_er ' und Zeilenvorschub)
"""..."""	E m	Alle Zeichen (inklusive " ' und Zeilenvorschub)
'''...'''	E m	Alle Zeichen (inklusive " ' und Zeilenvorschub)
b"..." B"..."	E 1	Binär-String (STD in P2)
f"..." F"..."	E 1	Format-String mit Platzhaltern der Form {...}
r"..." R"..."	- 1	Raw-String (Escape-Sequenz nicht interpretiert)
u"..." U"..."	E 1	Unicode-String (STD in P3)
ur"..." UR"..."	- 1	Raw-String im Unicode-Format

E = Escape-Sequenz \X wird interpretiert  
 - = Escape-Sequenz \X bleibt stehen (2 Zeichen)  
 1 = Nur einzelzeiliger String möglich  
 m = Mehrzeiliger String sowie ' und " im String erlaubt

## HINWEIS:

- \* String-Literale, die DIREKT aufeinander folgen, werden automatisch verkettet (das Verkettungszeichen "+" dazwischen ist nicht notwendig):

```
"hallo" "welt" 'usw.' # --> 'halloweltusw.'
"hallo" + "welt" + 'usw.' # --> 'halloweltusw.'
```

```
"hallo" ' ' "welt" ' ' "usw." # --> 'hallo welt usw.'
```

Dies gilt auch dann, wenn die String-Literale auf mehrere Zeilen verteilt sind:

```
"hallo" # --> "halloweltusw."
"welt"
'usw.'
```

## BEISPIELE:

```
print("Hallll\b\bo\t\\Welt") # --> 'Hallo \Welt'
print('Hallll\b\bo\t\\Welt') # --> 'Hallo \Welt'
print('\'Hallll\b\bo\t\\Welt\'') # --> 'Hallo \Welt'
print("""Hallll\b\bo\t\\Welt""") # --> 'Hallo \Welt'
print(r"Hallll\b\bo\t\\Welt") # --> 'Hallll\b\bo\t\\Welt' raw-String
print(R"Hallll\b\bo\t\\Welt") # --> 'Hallll\b\bo\t\\Welt' raw-String
print("Hallo \
      Welt") # Zeilenfortsetzung im String
              # (\n unterdrückt, Präfix 2. Zeile dabei)
print("Hallo " + \
      "Welt") # Zeilenfortsetzung im Code
              # (\n und Präfix 2. Zeile unterdrückt)
print("Hallo " "Welt") # --> 'Hallo Welt' (automatische Verkettung)
print("Hallo " + "Welt") # --> 'Hallo Welt' (explizite Verkettung)

"\u03C0 \U000003C0 \N{GREEK SMALL LETTER PI}" # --> 'ïM-^@ ïM-^@ ïM-^@'
"\u03A0 \U000003A0 \N{GREEK CAPITAL LETTER PI}" # --> 'Î Î Î'
"\u00E4 \U000000E4 \N{LATIN SMALL LETTER A WITH DIAERESIS}" # --> 'Ä Ä Ä'
"\u00F6 \U000000F6 \N{LATIN SMALL LETTER O WITH DIAERESIS}" # --> 'Ö Ö Ö'
"\u00FC \U000000FC \N{LATIN SMALL LETTER U WITH DIAERESIS}" # --> 'Û Û Û'
"\u00DF \U000000DF \N{LATIN SMALL LETTER SHARP S}" # --> 'Š Š Š'
"\u00C4 \U000000C4 \N{LATIN CAPITAL LETTER A WITH DIAERESIS}" # --> 'Ä Ä Ä'
"\u00D6 \U000000D6 \N{LATIN CAPITAL LETTER O WITH DIAERESIS}" # --> 'Ö Ö Ö'
"\u00DC \U000000DC \N{LATIN CAPITAL LETTER U WITH DIAERESIS}" # --> 'Û Û Û'
"\u00D7 \U000000D7 \N{MULTIPLICATION SIGN}" # --> 'Š Š Š'

hex(ord('ïM-^@')) # --> '03c0'
hex(ord('\u03C0')) # --> '03c0'
ord('ïM-^@') # --> 960
ord('\u03C0') # --> 960
chr(0x03C0) # --> 'ïM-^@'
chr(960) # --> 'ïM-^@'
chr(int('03C0', 16)) # --> 'ïM-^@'
chr(int('0x03C0', 0)) # --> 'ïM-^@'

import unicodedata # Modul importieren
unicodedata.name('\u03A0') # --> GREEK CAPITAL LETTER PI
unicodedata.name('\u03C0') # --> GREEK SMALL LETTER PI
```