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

Escape-Sequenzen in Python-Strings (C) 2016-2021 T.Birnthaler OSTC GmbH
==========

Doku --> http://docs.python.org/3/reference/lexical_analysis.html#literals
http://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals

Das Zeichen "\" (Backslash) dient in Python innerhalb von Strings als Escape-Zeichen ("Fluchtzeichen"), das zusammen mit dem DIREKT FOLGENDEN Zeichen eine spezielle Bedeutung hat. DarÃ¼ber werden z.B. Steuerzeichen (Strg-Z) und Unicode-Zeichen Ã¼ber ihre CODIERUNG oder ihren NAMEN angesprochen.

Folgende Kombinationen sind mÃ¶glich:

Zeichen	Bedeutung	
\\	Backslash selbst (immer verdoppeln!)	[backslash]
\'	Hochkomma (in '...' notwendig)	[single quote]
\"	GÃ¤nsefÃ¼Ã-^_chen (in "..." notwendig)	[double quote]
\a	Klingel (BEL)	[alert]
\b	RÃ¼ckschritt (BS)	[backspace]
\e	Escape (ESC)	[escape]
\f	Seitenvorschub (FF)	[form feed]
\n	Zeilenumbruch (LF)	[newline, linefeed]
\r	WagenrÃ¼cklauf (CR)	[carriage return]
\t	Horizontaler Tabulator (HT)	[tabulator]
\v	Vertikaler Tabulator (VT)	[vertical tabulator]
\0	Null-Byte (NUL)	[null]
\OOO	Zeichen mit Oktalcode "OOO" (000-377)	[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	[FEHLER]

HINWEISE:

* Innerhalb von RAW-STRINGS r"..." oder R"..." werden Escape-Sequenzen nicht interpretiert, der Backslash zÃ¤hlt als normales Zeichen.

* Ein Backslash am Zeilenende ("Line Continuation") verknÃ¼pft die aktuelle Zeile mit der folgenden zu einer LOGISCHEN ZEILE und erlaubt so eine beliebige Unterteilung von langen Zeilen in kurze StÃ¼cke. Die EinrÃ¼ckung der Folgezeile ist dabei irrelevant und muss nicht den EinrÃ¼ckungsregeln von Python gehorchen.

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

* Zeichenketten werden durch folgende Syntax definiert (quotiert):

Syntax		Bedeutung
"..."	E 1	Alle Zeichen (auÃ-^_er " und Zeilenvorschub)
'...'	E 1	Alle Zeichen (auÃ-^_er ' und Zeilenvorschub)
""".."""	E m	Alle Zeichen (inklusive " ' und Zeilenvorschub)
'''...'''	E m	Alle Zeichen (inklusive " ' und Zeilenvorschub)
b"..." B"..."	E 1	BinÃ¤r-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
ru"..." RU"..."	- 1	Raw-String im Unicode-Format
fr"..." FR"..."	- 1	Raw-Format-String
rf"..." RF"..."	- 1	Raw-Format-String
br"..." BR"..."	- 1	Raw-BinÃ¤r-String
rb"..." RB"..."	- 1	Raw-BinÃ¤r-String

E = Escape-Sequenz \X wird interpretiert
\- = Escape-Sequenz \X bleibt stehen (2 Zeichen)
1 = Nur einzeiliger String mÃ¶glich
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.'

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

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

  Funktioniert allerdings NICHT mit Variablen, Funktionen, Konstanten, ...

    text                             # text = "hallo"
    "welt".upper()                   # Funktion auf String "welt" anwenden
    string.ascii_lowercase           # Konstante

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 \                    # Zeilenfortsetzung im String
         Welt")                      # (\n unterdrÃ¼ckt, PrÃ¤fix 2. Zeile dabei)
  print("Hallo " + \                 # Zeilenfortsetzung im Code
        "Welt")                      # (\n und PrÃ¤fix 2. Zeile unterdrÃ¼ckt)
  print("Hallo " "Welt")             # 'Hallo Welt' (automatische Verkettung)
  print("Hallo " + "Welt")           # 'Hallo Welt' (explizite Verkettung)

Unicode-Zeichen per Code und per Name ansprechen:

  print(
      "\x41 \101 \u0041 \U00000041 \N{LATIN CAPITAL LETTER A}",         # A A A
      "\x61 \141 \u0061 \U00000061 \N{LATIN SMALL LETTER A}",           # a a a
      "\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}",      # Ã¤ Ã¤ Ã¤
      "\u00F6 \U000000F6 \N{LATIN SMALL LETTER O WITH DIAERESIS}",      # Ã¶ Ã¶ Ã¶
      "\u00FC \U000000FC \N{LATIN SMALL LETTER U WITH DIAERESIS}",      # Ã¼ Ã¼ Ã¼
      "\u00DF \U000000DF \N{LATIN SMALL LETTER SHARP S}",               # ÃM-^_ ÃM-^_ ÃM-^_
      "\u00C4 \U000000C4 \N{LATIN CAPITAL LETTER A WITH DIAERESIS}",    # ÃM-^D ÃM-^D ÃM-^D
      "\u00D6 \U000000D6 \N{LATIN CAPITAL LETTER O WITH DIAERESIS}",    # ÃM-^V ÃM-^V ÃM-^V
      "\u00DC \U000000DC \N{LATIN CAPITAL LETTER U WITH DIAERESIS}",    # ÃM-^\ ÃM-^\ ÃM-^\
      "\u00D7 \U000000D7 \N{MULTIPLICATION SIGN}",                      # ÃM-^W ÃM-^W ÃM-^W
      "\u4e14 \U00004e14 \N{CJK UNIFIED IDEOGRAPH-4E14}",               # ä¸M-^T ä¸M-^T ä¸M-^T
      sep="\n",
  )

HINWEIS: Bei Thonny gibt es leider Schwierigkeiten mit Unicode-Zeichen
         ab "\U0010000", diese fÃ¼hren zum "Einfrieren" der GUI.

      "\U0001f970 \N{SMILING FACE WITH SMILING EYES AND THREE HEARTS}", # ðM-^_¥° ðM-^_¥° ðM-^_¥°

Konvertierungs-Funktionen:

  print(
    hex(ord('ÏM-^@')),               # '03c0' (hexadecimal, ordinal)
    hex(ord('\u03C0')),           # '03c0'
    ord('ÏM-^@'),                    # 960
    ord('\u03C0'),                # 960
    chr(0x03C0),                  # 'ÏM-^@'    (character)
    chr(960),                     # 'ÏM-^@'
    chr(int('03C0', 16)),         # 'ÏM-^@'
    chr(int('0x03C0', 0)),        # 'ÏM-^@'
    sep="\n",
  )

  import unicodedata            # Modul importieren
  print(
    unicodedata.name('\u03A0'),    # 'GREEK CAPITAL LETTER PI'
    unicodedata.name('\u03C0'),    # 'GREEK SMALL LETTER PI'
    sep="\n",
  )
```