String

Following table is a list of escape or non-printable characters that can be represented with backslash notation.

An escape character gets interpreted; in a single quoted as well as double quoted strings.

|  |  |  |
| --- | --- | --- |
| **Backslash notation** | **Hexadecimal character** | **Description** |
| \a | 0x07 | Bell or alert |
| \b | 0x08 | Backspace |
| \cx |  | Control-x |
| \C-x |  | Control-x |
| \e | 0x1b | Escape |
| \f | 0x0c | Formfeed |
| \M-\C-x |  | Meta-Control-x |
| \n | 0x0a | Newline |
| \nnn |  | Octal notation, where n is in the range 0.7 |
| \r | 0x0d | Carriage return |
| \s | 0x20 | Space |
| \t | 0x09 | Tab |
| \v | 0x0b | Vertical tab |
| \x |  | Character x |
| \xnn |  | Hexadecimal notation, where n is in the range 0.9, a.f, or A.F |

## String Formatting Operator

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family. Following is a simple example −

#!/usr/bin/python

print "My name is %s and weight is %d kg!" % ('Zara', 21)

When the above code is executed, it produces the following result −

My name is Zara and weight is 21 kg!

Here is the list of complete set of symbols which can be used along with % −

|  |  |
| --- | --- |
| **Format Symbol** | **Conversion** |
| %c | character |
| %s | string conversion via str() prior to formatting |
| %i | signed decimal integer |
| %d | signed decimal integer |
| %u | unsigned decimal integer |
| %o | octal integer |
| %x | hexadecimal integer (lowercase letters) |
| %X | hexadecimal integer (UPPERcase letters) |
| %e | exponential notation (with lowercase 'e') |
| %E | exponential notation (with UPPERcase 'E') |
| %f | floating point real number |
| %g | the shorter of %f and %e |
| %G | the shorter of %f and %E |

Other supported symbols and functionality are listed in the following table −

|  |  |
| --- | --- |
| **Symbol** | **Functionality** |
| \* | argument specifies width or precision |
| - | left justification |
| + | display the sign |
| <sp> | leave a blank space before a positive number |
| # | add the octal leading zero ( '0' ) or hexadecimal leading '0x' or '0X', depending on whether 'x' or 'X' were used. |
| 0 | pad from left with zeros (instead of spaces) |
| % | '%%' leaves you with a single literal '%' |
| (var) | mapping variable (dictionary arguments) |
| m.n. | m is the minimum total width and n is the number of digits to display after the decimal point (if appl.) |

**Date & Time**

A Python program can handle date and time in several ways. Converting between date formats is a common chore for computers. Python's time and calendar modules help track dates and times.

**What is Tick?**

Time intervals are floating-point numbers in units of seconds. Particular instants in time are expressed in seconds since 12:00am, January 1, 1970(epoch).

**What is TimeTuple?**

Many of Python's time functions handle time as a tuple of 9 numbers, as

|  |  |  |
| --- | --- | --- |
| **Index** | **Field** | **Values** |
| 0 | 4-digit year | 2008 |
| 1 | Month | 1 to 12 |
| 2 | Day | 1 to 31 |
| 3 | Hour | 0 to 23 |
| 4 | Minute | 0 to 59 |
| 5 | Second | 0 to 61 (60 or 61 are leap-seconds) |
| 6 | Day of Week | 0 to 6 (0 is Monday) |
| 7 | Day of year | 1 to 366 (Julian day) |
| 8 | Daylight savings | -1, 0, 1, -1 means library determines DST |

The above tuple is equivalent to **struct\_time** structure. This structure has following attributes –

|  |  |  |
| --- | --- | --- |
| **Index** | **Attributes** | **Values** |
| 0 | tm\_year | 2008 |
| 1 | tm\_mon | 1 to 12 |
| 2 | tm\_mday | 1 to 31 |
| 3 | tm\_hour | 0 to 23 |
| 4 | tm\_min | 0 to 59 |
| 5 | tm\_sec | 0 to 61 (60 or 61 are leap-seconds) |
| 6 | tm\_wday | 0 to 6 (0 is Monday) |
| 7 | tm\_yday | 1 to 366 (Julian day) |
| 8 | tm\_isdst | -1, 0, 1, -1 means library determines DST |

strftime()

The method **strftime()** converts a tuple or struct\_time representing a time as returned by gmtime() or localtime() to a string as specified by the format argument.

If t is not provided, the current time as returned by localtime() is used. format must be a string. An exception ValueError is raised if any field in t is outside of the allowed range.

## Syntax

Following is the syntax for **strftime()** method:

## Directive

* %a - abbreviated weekday name
* %A - full weekday name
* %b - abbreviated month name
* %B - full month name
* %c - preferred date and time representation
* %C - century number (the year divided by 100, range 00 to 99)
* %d - day of the month (01 to 31)
* %D - same as %m/%d/%y
* %e - day of the month (1 to 31)
* %g - like %G, but without the century
* %G - 4-digit year corresponding to the ISO week number (see %V).
* %h - same as %b
* %H - hour, using a 24-hour clock (00 to 23)
* %I - hour, using a 12-hour clock (01 to 12)
* %j - day of the year (001 to 366)
* %m - month (01 to 12)
* %M - minute
* %n - newline character
* %p - either am or pm according to the given time value
* %r - time in a.m. and p.m. notation
* %R - time in 24 hour notation
* %S - second
* %t - tab character
* %T - current time, equal to %H:%M:%S
* %u - weekday as a number (1 to 7), Monday=1. Warning: In Sun Solaris Sunday=1
* %U - week number of the current year, starting with the first Sunday as the first day of the first week
* %V - The ISO 8601 week number of the current year (01 to 53), where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the week
* %W - week number of the current year, starting with the first Monday as the first day of the first week
* %w - day of the week as a decimal, Sunday=0
* %x - preferred date representation without the time
* %X - preferred time representation without the date
* %y - year without a century (range 00 to 99)
* %Y - year including the century
* %Z or %z - time zone or name or abbreviation
* %% - a literal % character