Python Time Module

There is a popular time module available in Python which provides functions for working with times and for converting between representations.

Compiled By,

Md Farmanul Haque,
Technical Trainer,
GLA University,
Mathura

altzone:

The method altzone() is the attribute of the time module. This returns the offset of the local DST timezone, in seconds west of UTC, if one is defined. This is negative if the local DST timezone is east of UTC (as in Western Europe, including the UK). Only use this if daylight is nonzero.

Syntax

Following is the syntax for altzone() method −

```
Time.altzone
```

```
import time

print ("time.altzone : ", time.altzone)
time.altzone :  -23400
```

asctime():

The method asctime() converts a tuple or struct_time representing a time as returned by gmtime() or localtime() to a 24-character string of the following form − 'Tue Feb 17 23:21:05 2009'.

Syntax

Following is the syntax for asctime() method −

```
time.asctime([t]))
```

Parameters

t − This is a tuple of 9 elements or struct_time representing a time as returned by gmtime() or localtime() function.

Return Value

This method returns 24-character string of the following form − 'Tue Feb 17 23:21:05 2009'.

Example

The following example shows the usage of asctime() method.

```
import time

t = time.localtime()
print ("asctime : ",time.asctime(t))
```

Result

When we run the above program, it produces the following result −

```
asctime :  Mon Feb 15 09:46:24 2016
```

clock():

The method clock() returns the current processor time as a floating point number expressed in seconds on Unix. The precision depends on that of the C function of the same name, but in any case, this is the function to use for benchmarking Python or timing algorithms.

On Windows, this function returns wall-clock seconds elapsed since the first call to this function, as a floating point number, based on the Win32 function QueryPerformanceCounter.

Syntax

Following is the syntax for clock() method −

```
time.clock()
```

Return Value

This method returns the current processor time as a floating point number expressed in seconds on Unix and in Windows it returns wall-clock seconds elapsed since the first call to this function, as a floating point number.

Example

The following example shows the usage of clock() method.

```
import time

def procedure():
   time.sleep(2.5)

# measure process time
t0 = time.clock()
procedure()
print (time.clock() - t0, "seconds process time")

# measure wall time
t0 = time.time()
procedure()
print (time.time() - t0, "seconds wall time")
```

Result

When we run the above program, it produces the following result −

```
2.4993855364299096 seconds process time
2.5 seconds wall time
```

Note − Not all systems can measure the true process time. On such systems (including Windows), clock usually measures the wall time since the program was started.

ctime():

The method ctime() converts a time expressed in seconds since the epoch to a string representing local time. If secs is not provided or None, the current time as returned by time() is used. This function is equivalent to asctime(localtime(secs)). Locale information is not used by ctime().

Syntax

Following is the syntax for ctime() method −

```
time.ctime([ sec ])
```

Parameters

sec − These are the number of seconds to be converted into string representation.

Return Value

This method does not return any value.

Example

The following example shows the usage of ctime() method.

```
import time

print ("ctime : ", time.ctime())
```

Result

When we run the above program, it produces the following result −

```
ctime :  Mon Feb 15 09:55:34 2016
```

gmtime():

The method gmtime() converts a time expressed in seconds since the epoch to a struct_time in UTC in which the dst flag is always zero. If secs is not provided or None, the current time as returned by time() is used.

Syntax

Following is the syntax for gmtime() method −

```
time.gmtime([ sec ])
```

Parameters

sec − These are the number of seconds to be converted into structure struct_time representation.

Return Value

This method does not return any value.

Example

The following example shows the usage of gmtime() method.

```
import time

print ("gmtime :", time.gmtime(1455508609.34375))
```

Result

When we run the above program, it produces the following result −

```
gmtime : time.struct_time(tm_year = 2016, tm_mon = 2, tm_mday = 15,
tm_hour = 3,
   tm_min = 56, tm_sec = 49, tm_wday = 0, tm_yday = 46, tm_isdst =
0)
```

gmtime():

The method gmtime() converts a time expressed in seconds since the epoch to a struct_time in UTC in which the dst flag is always zero. If secs is not provided or None, the current time as returned by time() is used.

Syntax

Following is the syntax for gmtime() method −

```
time.gmtime([ sec ])
```

Parameters

sec − These are the number of seconds to be converted into structure struct_time representation.

Return Value

This method does not return any value.

Example

The following example shows the usage of gmtime() method.

```
import time

print ("gmtime :", time.gmtime(1455508609.34375))
```

Result

When we run the above program, it produces the following result −

```
gmtime : time.struct_time(tm_year = 2016, tm_mon = 2, tm_mday = 15,
tm_hour = 3,
   tm_min = 56, tm_sec = 49, tm_wday = 0, tm_yday = 46, tm_isdst =
0)
```

localtime():

The method localtime() is similar to gmtime() but it converts number of seconds to local time. If secs is not provided or None, the current time as returned by time() is used. The dst flag is set to 1 when DST applies to the given time.

Syntax

Following is the syntax for localtime() method −

```
time.localtime([ sec ])
```

Parameters

sec − These are the number of seconds to be converted into structure struct_time representation.

Return Value

This method does not return any value.

Example

The following example shows the usage of localtime() method.

```
import time

print ("time.localtime() : %s" , time.localtime())
```

Result

When we run the above program, it produces the following result:

```
time.localtime() : time.struct_time(tm_year = 2016, tm_mon = 2,
tm_mday = 15,
   tm_hour = 10, tm_min = 13, tm_sec = 50, tm_wday = 0, tm_yday =
46, tm_isdst = 0)
```

mktime():

The method mktime() is the inverse function of localtime(). Its argument is the struct_time or full 9-tuple and it returns a floating point number, for compatibility with time().

If the input value cannot be represented as a valid time, either OverflowError or ValueError will be raised.

Syntax

Following is the syntax for mktime() method −

```
time.mktime(t)
```

Parameters

t − This is the struct_time or full 9-tuple.

Return Value

This method returns a floating point number, for compatibility with time().

Example

The following example shows the usage of mktime() method.

```
import time

t = (2016, 2, 15, 10, 13, 38, 1, 48, 0)
d = time.mktime(t)
print ("time.mktime(t) : %f" %  d)
print ("asctime(localtime(secs)): %s" %
time.asctime(time.localtime(d)))
```

Result

When we run the above program, it produces the following result −

```
time.mktime(t) : 1455511418.000000
asctime(localtime(secs)): Mon Feb 15 10:13:38 2016
```

sleep():

The method sleep() suspends execution for the given number of seconds. The argument may be a floating point number to indicate a more precise sleep time.

The actual suspension time may be less than that requested because any caught signal will terminate the sleep() following execution of that signal's catching routine.

Syntax

Following is the syntax for sleep() method −

```
time.sleep(t)
```

Parameters

t − This is the number of seconds execution to be suspended.

Return Value

This method does not return any value.

Example

The following example shows the usage of sleep() method.

```
import time

print ("Start : %s" % time.ctime())
time.sleep( 5 )
print ("End : %s" % time.ctime())
```

Result

When we run the above program, it produces the following result −

```
Start : Mon Feb 15 12:08:42 2016
End : Mon Feb 15 12:08:47 2016
```

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 −

```
time.strftime(format[, t])
```

Parameters

- **t − This is the time in number of seconds to be formatted.**
- **format − This is the directive which would be used to format given time.**

Directive

The following directives can be embedded in the format string −

- **%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**

Return Value

This method does not return any value.

Example

The following example shows the usage of strftime() method.

```
import time


t = (2015, 12, 31, 10, 39, 45, 1, 48, 0)
t = time.mktime(t)
print (time.strftime("%b %d %Y %H:%M:%S", time.localtime(t)))
```

Result

When we run the above program, it produces the following result −

```
Dec 31 2015 10:39:45
```

strptime():

The method strptime() parses a string representing a time according to a format. The return value is a struct_time as returned by gmtime() or localtime().

The format parameter uses the same directives as those used by strftime(); it defaults to "%a %b %d %H:%M:%S %Y" which matches the formatting returned by ctime().

If string cannot be parsed according to format, or if it has excess data after parsing, ValueError is raised.

Syntax

Following is the syntax for strptime() method −

```
time.strptime(string[, format])
```

Parameters

- **string − This is the time in string format which would be parsed based on the given format.**
- **format − This is the directive which would be used to parse the given string.**

Directive

The following directives can be embedded in the format string −

- **%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**

Return Value

This return value is struct_time as returned by gmtime() or localtime().

Example

The following example shows the usage of strptime() method.

```
import time


struct_time = time.strptime("30 12 2015", "%d %m %Y")

print ("tuple : ", struct_time)
```

Result

When we run the above program, it produces the following result −

```
tuple :  time.struct_time(tm_year = 2015, tm_mon = 12, tm_mday =
30,

   tm_hour = 0, tm_min = 0, tm_sec = 0, tm_wday = 2, tm_yday = 364,
tm_isdst = -1)
```

time():

The method time() returns the time as a floating point number expressed in seconds since the epoch, in UTC.

Note − Even though the time is always returned as a floating point number, not all systems provide time with a better precision than 1 second. While this function normally returns non-decreasing values, it can return a lower value than a previous call if the system clock has been set back between the two calls.

Syntax

Following is the syntax for time() method −

```
time.time()
```

Return Value

This method returns the time as a floating point number expressed in seconds since the epoch, in UTC.

Example

The following example shows the usage of time() method.

```
import time


print ("time.time(): %f " %  time.time())

print (time.localtime( time.time() ))

print (time.asctime( time.localtime(time.time()) ))
```

Result

When we run the above program, it produces the following result −

```
time.time(): 1455519806.011433

time.struct_time(tm_year = 2016, tm_mon = 2, tm_mday = 15, tm_hour
= 12, tm_min = 33,

   tm_sec = 26, tm_wday = 0, tm_yday = 46, tm_isdst = 0)

Mon Feb 15 12:33:26 2016
```

tzset():

The method tzset() resets the time conversion rules used by the library routines. The environment variable TZ specifies how this is done.

The standard format of the TZ environment variable is (whitespace added for clarity) −

```
std offset [dst [offset [,start[/time], end[/time]]]]
```

std and dst − Three or more alphanumerics giving the timezone abbreviations. These will be propagated into time.tzname.

offset − The offset has the form: hh[:mm[:ss]]. This indicates the value added the local time to arrive at UTC. If preceded by a '-', the timezone is east of the Prime Meridian; otherwise, it is west. If no offset follows *dst*, summer time is assumed to be one hour ahead of standard time.

start[/time], end[/time] − Indicates when to change to and back from DST. The format of the start and end dates are one of the following −

- **Jn − The Julian day n (1 <= n <= 365). Leap days are not counted, so in all years February 28 is day 59 and March 1 is day 60.**
- **n − The zero-based Julian day (0 <= n <= 365). Leap days are counted, and it is possible to refer to February 29.**
- **Mm.n.d − The d'th day (0 <= d <= 6) or week n of month m of the year (1 <= n <= 5, 1 <= m <= 12, where week 5 means 'the last d day in month m' which may occur in either the fourth or the fifth week). Week 1 is the first week in which the d'th day occurs. Day zero is Sunday.**
- **time − This has the same format as offset except that no leading sign ('-' or '+') is allowed. The default, if time is not given, is 02:00:00.**

Syntax

Following is the syntax for tzset() method −

```
time.tzset()
```

Return Value

This method does not return any value.

Example

The following example shows the usage of tzset() method.

```
import time
import os

os.environ['TZ'] = 'EST+05EDT,M4.1.0,M10.5.0'
time.tzset()
print (time.strftime('%X %x %Z'))

os.environ['TZ'] = 'AEST-10AEDT-11,M10.5.0,M3.5.0'
time.tzset()
print (time.strftime('%X %x %Z'))
```

Result

When we run the above program, it produces the following result −

```
13:00:40 02/17/09 EST
05:00:40 02/18/09 AEDT
```