PYTHON 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*.

There is a popular **time** module available in Python which provides functions for working with times, and for converting between representations. The function *time.time* returns the current system time in ticks since 12:00am, January 1, 1970*epoch*.

Example

```
#!/usr/bin/python
import time;  # This is required to include time module.

ticks = time.time()
print "Number of ticks since 12:00am, January 1, 1970:", ticks
```

This would produce a result something as follows −

```
Number of ticks since 12:00am, January 1, 1970: 7186862.73399
```

Date arithmetic is easy to do with ticks. However, dates before the epoch cannot be represented in this form. Dates in the far future also cannot be represented this way - the cutoff point is sometime in 2038 for UNIX and Windows.

What is TimeTuple?

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

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

Getting current time

To translate a time instant from a *seconds since the epoch* floating-point value into a time-tuple, pass the floating-point value to a function $e.g.$, $localtime$ that returns a time-tuple with all nine items valid.

```
#!/usr/bin/python
import time;

localtime = time.localtime(time.time())
print "Local current time :", localtime
```

This would produce the following result, which could be formatted in any other presentable form −

```
Local current time : time.struct_time(tm_year=2013, tm_mon=7,
tm_mday=17, tm_hour=21, tm_min=26, tm_sec=3, tm_wday=2, tm_yday=198, tm_isdst=0)
```

Getting formatted time

You can format any time as per your requirement, but simple method to get time in readable format is asctime −

```
#!/usr/bin/python
import time;

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

This would produce the following result −

```
Local current time : Tue Jan 13 10:17:09 2009
```

Getting calendar for a month

The calendar module gives a wide range of methods to play with yearly and monthly calendars. Here, we print a calendar for a given month $Jan 2008$ −

```
#!/usr/bin/python
import calendar

cal = calendar.month(2008, 1)
print "Here is the calendar:"
print cal
```

This would produce the following result −

```
Here is the calendar:
    January 2008
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

The *time* Module

There is a popular **time** module available in Python which provides functions for working with times and for converting between representations. Here is the list of all available methods −

SN	Function with Description
1	time.altzone 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 *asinWesternEurope, includingtheUK*. Only use this if daylight is nonzero.
2	time.asctime[*tupletime*] Accepts a time-tuple and returns a readable 24-character string such as 'Tue Dec 11 18:07:14 2008'.
3	time.clock Returns the current CPU time as a floating-point number of seconds. To measure computational costs of different approaches, the value of time.clock is more useful than that of time.time.
4	time.ctime[*secs*] Like asctime*localtime*(*secs*) and without arguments is like asctime
5	time.gmtime[*secs*] Accepts an instant expressed in seconds since the epoch and returns a time-tuple t with the UTC time. Note : t.tm_isdst is always 0
6	time.localtime[*secs*] Accepts an instant expressed in seconds since the epoch and returns a time-tuple t with the local time $t.\ tm_isdstis0or1,\ dependingonwhetherDSTappliestoinstantsecsbylocalrules.$
7	time.mktime*tupletime* Accepts an instant expressed as a time-tuple in local time and returns a floating-point value with the instant expressed in seconds since the epoch.
8	

	time.sleep*secs* Suspends the calling thread for secs seconds.
9	time.strftime*fmt*[, *tupletime*] Accepts an instant expressed as a time-tuple in local time and returns a string representing the instant as specified by string fmt.
10	time.strptime$str, fmt = '$ Parses str according to format string fmt and returns the instant in time-tuple format.
11	time.time Returns the current time instant, a floating-point number of seconds since the epoch.
12	time.tzset Resets the time conversion rules used by the library routines. The environment variable TZ specifies how this is done.

Let us go through the functions briefly −

There are following two important attributes available with time module:

SN	Attribute with Description
1	**time.timezone** Attribute time.timezone is the offset in seconds of the local time zone *withoutDST* from UTC $> 0 intheAmericas; <= 0 inmostofEurope, Asia, Africa.$
2	**time.tzname** Attribute time.tzname is a pair of locale-dependent strings, which are the names of the local time zone without and with DST, respectively.

The *calendar* Module

The calendar module supplies calendar-related functions, including functions to print a text calendar for a given month or year.

By default, calendar takes Monday as the first day of the week and Sunday as the last one. To change this, call calendar.setfirstweekday function.

Here is a list of functions available with the *calendar* module:

SN	Function with Description
1	**calendar.calendar**$year, w = 2, l = 1, c = 6$ Returns a multiline string with a calendar for year year formatted into three columns

separated by c spaces. w is the width in characters of each date; each line has length 21*w+18+2*c. l is the number of lines for each week.

2 **calendar.firstweekday**

Returns the current setting for the weekday that starts each week. By default, when calendar is first imported, this is 0, meaning Monday.

3 **calendar.isleap*year***

Returns True if year is a leap year; otherwise, False.

4 **calendar.leapdays*y*1, *y*2**

Returns the total number of leap days in the years within range*y*1, *y*2.

5 **calendar.month*year, month, w* = 2, *l* = 1**

Returns a multiline string with a calendar for month month of year year, one line per week plus two header lines. w is the width in characters of each date; each line has length 7*w+6. l is the number of lines for each week.

6 **calendar.monthcalendar*year, month***

Returns a list of lists of ints. Each sublist denotes a week. Days outside month month of year year are set to 0; days within the month are set to their day-of-month, 1 and up.

7 **calendar.monthrange*year, month***

Returns two integers. The first one is the code of the weekday for the first day of the month month in year year; the second one is the number of days in the month. Weekday codes are 0 *Monday* to 6 *Sunday*; month numbers are 1 to 12.

8 **calendar.prcal*year, w* = 2, *l* = 1, *c* = 6**

Like print calendar.calendar*year, w, l, c*.

9 **calendar.prmonth*year, month, w* = 2, *l* = 1**

Like print calendar.month*year, month, w, l*.

10 **calendar.setfirstweekday*weekday***

Sets the first day of each week to weekday code weekday. Weekday codes are 0 *Monday* to 6 *Sunday*.

11 **calendar.timegm*tupletime***

The inverse of time.gmtime: accepts a time instant in time-tuple form and returns the same instant as a floating-point number of seconds since the epoch.

12 **calendar.weekday*year, month, day***

Returns the weekday code for the given date. Weekday codes are 0 *Monday* to 6 *Sunday*; month numbers are 1 *January* to 12 *December*.

Other Modules & Functions:

If you are interested, then here you would find a list of other important modules and functions to play with date & time in Python:

- The *datetime* Module
- The *pytz* Module
- The *dateutil* Module

Processing math: 100%