Date and Time

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Python

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).

What is Tick? (Cont.)

```
>>>
>>> import time
>>>
>>> ticks = time.time()
>>> print ("Number of ticks since 12:00am, January 1, 1970 : ", ticks)
Number of ticks since 12:00am, January 1, 1970 :  1503936098.8157136
>>>
```

What is Tick? (Cont.)

- 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 the Python's time functions handle time as a tuple of 9 numbers, as shown below :

Field	Values
4-digit year	2017
Month	1 to 12
Day	1 to 31
Hour	0 to 23
Minute	0 to 59
Second	0 to 61 (60 or 61 are leap-seconds)
Day of Week	0 to 6 (0 is Monday)
Day of year	1 to 366 (Julian day)
Daylight savings	-1, 0 ,1, -1 means library determines DST

What is TimeTuple? (Cont.)

- For example:

```
>>> import time
>>>
>>> print (time.localtime())
time.struct_time(tm_year=2017, tm_mon=8, tm_mday=29, tm_hour=1, tm_min=8, tm_sec=48, tm_wday
=1, tm_yday=241, tm_isdst=0)
>>>
```

What is TimeTuple? (Cont.)

- `struct_time()`

Attributes	Values
`tm_year`	2017
`tm_mon`	1 to 12
`tm_mday`	1 to 31
`tm_hour`	0 to 23
`tm_min`	0 to 59
`tm_sec`	0 to 61 (60 or 61 are leap-seconds)
`tm_wday`	0 to 6 (0 is Monday)
`tm_yday`	1 to 366 (Julian day)
`tm_isdst`	-1, 0 ,1, -1 means library determines DST

Getting formatted time

- Can format any time as per your requirement, but a simple method to get time in a readable format is **`asctime()`**.

```
>>> import time
>>>
>>> localtime = time.asctime( time.localtime(time.time()))
>>> print ("Local current time : ", localtime)
Local current time :  Tue Aug 29 01:16:10 2017
>>>
```

Getting calendar for a month

- The calendar module gives a wide range of methods to play with yearly and monthly calendars.

```
>>> import calendar
>>>
>>>
>>>
>>> cal = calendar.month(2017, 8)
>>> print ("Hear is the calendar : ")
Hear is the calendar :
>>> print (cal)
    August 2017
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
```

```
1 import calendar
2
3 cal = calendar.month(2017, 8)
4 print ("Here is the calendar : ")
5 print (cal)
```

Console

<terminated> Hello.py [/usr/bin/python3.6]

```
Here is the calendar :
    August 2017
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

- `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 (as in Western Europe, including the UK).
  - Use this if the daylight is nonzero.

The Time Module (Cont.)

- `time.asctime([tupletime])`
  - Accepts a time-tuple and returns a readable 24-character string such as 'Tue Dec 11 18:07:14 2008'.

The Time Module (Cont.)

- `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()`.

The Time Module (Cont.)

- `time.ctime([secs])`
  - Like `asctime(localtime(secs))` and without arguments is like `asctime( )`.

The Time Module (Cont.)

- `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`.

The Time Module (Cont.)

- `time.localtime([secs])`
  - Accepts an instant expressed in seconds since the epoch and returns a time-tuple t with the local time (`t.tm_isdst` is `0` or `1`, depending on whether DST applies to instant secs by local rules).

The Time Module (Cont.)

- `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.

The Time Module (Cont.)

- `time.sleep(secs)`
  - Suspends the calling thread for `secs` seconds.

The Time Module (Cont.)

- **`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`**.

The Time Module (Cont.)

- **`time.strptime(str, fmt='%a %b %d %H:%M:%s %Y')`**
  - Parses str according to format string **`fmt`** and returns the instant in time-tuple format.

The Time Module (Cont.)

- `time.time()`
  - Returns the current time instant, a floating-point number of seconds since the epoch.

The Time Module (Cont.)

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

The Calendar Module

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

The Calendar Module (Cont.)

- `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.

The Calendar Module (Cont.)

- `calendar.isleap(year)`
  - Returns `True` if year is a leap year; otherwise, `False`.

The Calendar Module (Cont.)

- `calendar.leapdays(y1, y2)`
  - Returns the total number of leap days in the years within `range(y1,y2)`.

The Calendar Module (Cont.)

- `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.

The Calendar Module (Cont.)

- `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.

The Calendar Module (Cont.)

- `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.

The Calendar Module (Cont.)

- `calendar.prcal(year, w = 2, l = 1, c = 6)`
  - Like print `calendar.calendar(year,w,l,c)`.

The Calendar Module (Cont.)

- `calendar.prmonth(year, w = 2, l = 1)`
  - Like print `calendar.month(year,month,w,l)`.

The Calendar Module (Cont.)

- `calendar.setfirstweekday(weekday)`
  - Sets the first day of each week to weekday code weekday. Weekday codes are 0 (Monday) to 6 (Sunday).

The Calendar Module (Cont.)

- **`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.

The Calendar Module (Cont.)

- `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).

Lab

```
1 from time import localtime
2
3 t = localtime()
4 start_day = '%d-01-01' %t.tm_year
5 elapsed_day = t.tm_yday
6
7 print('오늘은 [%s]이후 [%d]일째 되는 날입니다.' %(start_day, elapsed_day))
8
```

Lab (Cont.)

```
1 from time import localtime
2
3 weekdays = ['월요일', '화요일', '수요일', '목요일', '금요일', '토요일', '일요일']
4
5 t = localtime()
6 today = '%d-%d-%d' %(t.tm_year, t.tm_mon, t.tm_mday)
7 week = weekdays[t.tm_wday]
8
9 print('[%s] 오늘은 [%s]입니다.' %(today, week))
10
```

Lab (Cont.)

```
1  from datetime import datetime
2
3  start = datetime.now()
4  print('1에서 백만까지 더합니다.')
5  ret = 0
6  for i in range(1000000):
7      ret += i
8  print('1에서 백만까지 더한 결과: %d' %ret)
9  end = datetime.now()
10 elapsed = end - start
11 print('총 계산 시간: ', end='');print(elapsed)
12 elapsed_ms = int(elapsed.total_seconds()*1000)
13 print('총 계산 시간: %dms' %elapsed_ms)
14
```