Working with Time

Notably missing from this page are the capabilities of the Python datetime module, which was introduced in 2.3 and is very powerful.

This page gives some basics of working with time in Python. More detail can be found in the time Module Documentation.

https://wiki.python.org/moin/WorkingWithTime

Formats

There are several ways to work with time:

format	Python
seconds since the "Epoch"	`time.time()`
tuple	`time.gmtime()`
string	`time.ctime()`

The Epoch is January 1st, 1970, midnight, on UNIX systems. On other systems, look at the results of `time.gmtime(0)` to discover the date of the Epoch.

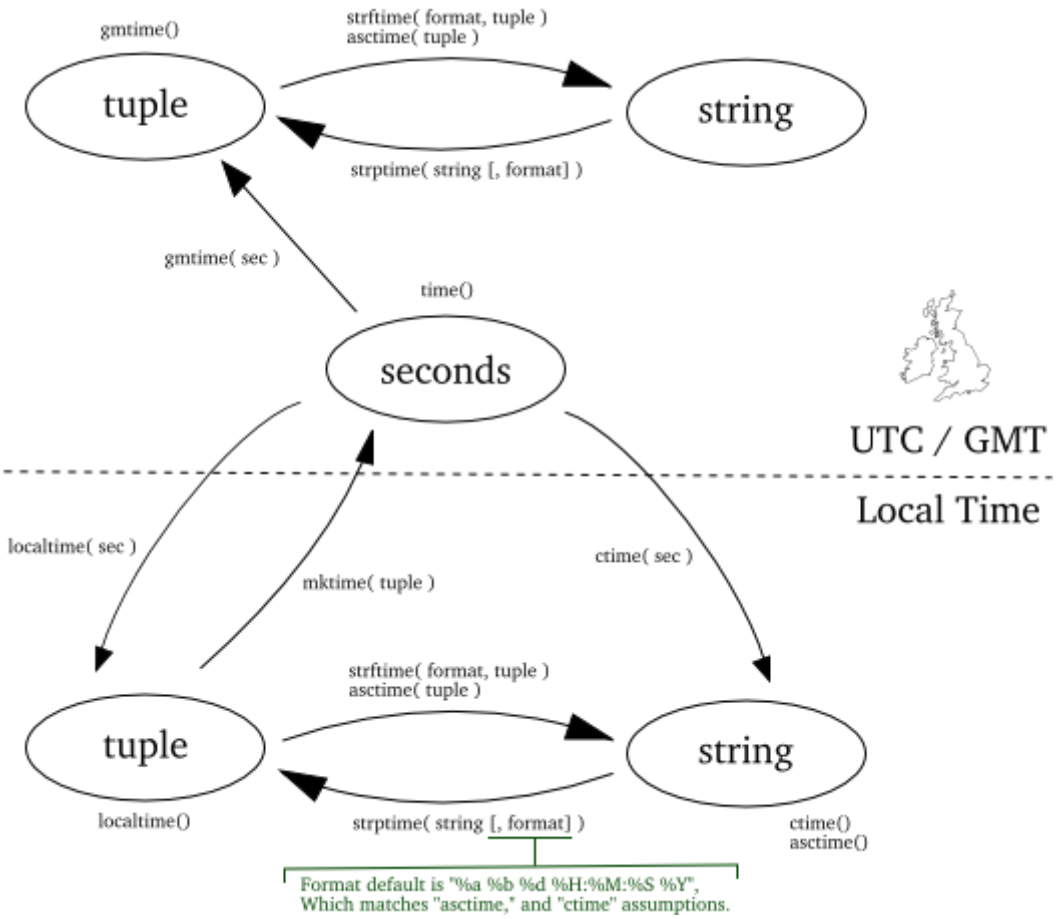
Measuring time in seconds since the Epoch is convenient for storage and comparing dates, because you only have a single number to consider.

The **tuple** contains several values, arranged the following way: year, month 1-12, day 1-31, hour 0-23, minutes 0-59, seconds 0-61, day 0-6 (Mon-Sun), day 1-366, and DST -1,0,+1. "DST" means "daylight savings time." For more information, see time Module Documentation.

Using a tuple is convenient, because you can access particular numbers by index, and because you can easily compare dates. Remember that Python compares tuple data from front to back. The data is indexed so that comparing tuple times is intuitive.

The **string** format reads something like "Mon Feb 16 16:04:25 2004". You can't really compare these usefully, but they're what you need to display things to the user.

Converting Between Formats



(see TimeTransitionsImage for image details)

mock ISO 8601

You can construct ISO 8601 with the following:

Toggle line numbers

```
1 iso_time = time.strftime("%Y-%m-%dT%H:%M:%S", tuple_time)
```

You can revert the construction with the following:

Toggle line numbers

```
1 tuple_time = time.strptime("2004-06-03T00:44:35", "%Y-%m-%dT%H:%M:%S")
```

The problem with this, though, is that if the string isn't formatted *exactly like that*, it won't be read. For instance, `20040603T00:44:35` is valid ISO8601, as is `20040603` for that matter, but neither will be read by the code above.

To account for both common inputs, you could use:

Toggle line numbers

```
1 tuple_time = time.strptime(iso_time.replace("-", ""),
"%Y%m%dT%H:%M:%S")
```

I seem to require receiving both while working with XmlRpc produced by a mix of both Python and Perl code. The XmlRpc specification is not very particular about the ISO8601 variant used for it's DateTime.

Note that this method does not handle time zones, so trying to parse `2004-06-03T12:34:56-0700` or `2004-06-03T12:34:56Z` will fail. This is a fundamental limitation of `time.strptime` for which there is no easy workaround.

Parsing ISO 8601 with xml.utils.iso8601

If you have `xml.utils.iso8601`, you can do the following:

Toggle line numbers

```
1 import xml.utils.iso8601
2 import time
3
4 # Present time, in ISO8601
5 print xml.utils.iso8601.tostring(time.time()) # returns "2004-04-
10T04:44:08.19Z"
6 # From ISO8601 string, to seconds since epoch UTC
7 print xml.utils.iso8601.parse("2004-04-09T21:39:00-08:00") # -8:00 for
Seattle, WA
8 # The epoch, in ISO8601
9 print xml.utils.iso8601.ctime(0) # returns "1969-12-31T16:00-08:00"
```

The module isn't perfect; If you omit the dashes, (perfectly valid ISO8601,) you'll get a `ValueError` exception. However, unlike `time.strptime`, `xml.utils.iso8601.parse` will handle the timezone specification correctly.

xml.utils is distributed in *python-xml* in debian and ubuntu, but appears to be unmaintained.

Parsing ISO 8601 with pyiso8601

http://pypi.python.org/pypi/iso8601/ provides a standalone ISO8601 parser.

Parsing ISO 8601 with python-dateutil

Although this isn't advertised as ISO8601 support, python-dateutil seems to deal with the format correctly (for example, it has time zone support).

```
import dateutil.parser
dateutil.parser.parse('2008-08-09T18:39:22Z')
```

Parsing RFC 3339 with the rfc3339 module

RFC 3339 is the standard datetime format for most internet and web standards. It is a stricter ISO8601 profile, restricted to datetimes of second resolution or more precise. It requires explicit timezones as offsets from UTC.

python-rfc3339 (announcement) is an RFC 3339 parser. It also implements fixed-offset timezones and a formatter. It lacks support for leap seconds, as does the standard library's datetime module.

ISO8601 and Python in general

All in all, *reading* ISO8601 time values is non-trivial. For example, it is possible that someone can specify a full day, or a full week even, in ISO format. You can also specify days by number, if I recall correctly.

Generally, programs conceed to accept only a limited subset of ISO8601. That subset tends to be "`yyyy-mm-ddThh:mm:ss`" and then either ".mmmmmm" sub-seconds or "+hh:00"/"-hh:00" for time zone.

Time Zones

(nothing yet)

See Also

This page is meant just to give you the basics. You may also want to read:

- the official Python time module documentation.
- Coordinated_Universal_Time ("UTC"), Greenwich_Mean_Time ("GMT")
- Official nomencalture and description for 8601: http://www.standardsdirect.org/standards/standards3/StandardsCatalogue24_view_23407.html
- mxDateTime python extension module. provides many interfaces for handling dates and times, including ISO 8601 handling: http://www.egenix.com/files/python/mxDateTime.html

Contributors

LionKimbro