The time Module

WE'LL COVER THE FOLLOWING

The **time** module provides the Python developer access to various time-related functions. The time module is based around what it known as an **epoch**, the point when time starts. For Unix systems, the epoch was in 1970. To find out what the epoch is on your system, try running the following:

```
import time
print(time.gmtime(0))
#time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3
```

I ran this on Windows 7 and it too seems to think that time began in 1970. Anyway, in this section, we will be studying the following time-related functions:

> - time.ctime
> - time.sleep
> - time.strftime
> - time.time

Let's get started!

time.ctime

The **time.ctime** function will convert a time in seconds since the epoch to a string representing local time. If you don't pass it anything, then the current time is returned. Let's try out a couple of examples:

```
import time
print(time.ctime())
#'Fri Dec 30 17:11:58 2016'

print(time.ctime(1384112639))
#'Sun Nov 10 19:43:59 2013'
```

Here we show the results of calling **ctime** with nothing at all and with a fairly random number of seconds since the epoch. I have seen sort of thing used when someone saves the date as seconds since the epoch and then they want to convert it to something a human can understand. It's a bit simpler to save a big integer (or long) to a database then to mess with formatting it from a datetime object to whatever date object the database accepts. Of course, that also has the drawback that you do need to convert the integer or float value back into a string.

time.sleep

The **time.sleep** function gives the developer the ability to suspend execution of your script a given number of seconds. It's like adding a pause to your program. I have found this personally useful when I need to wait a second for a file to finish closing or a database commit to finish committing. Let's take a look at an example.

```
import time

for x in range(5):
    time.sleep(2)
    print("%s: Slept for 2 seconds" % time.ctime())
```

We are printing timestamps with each line to see whether it actually waited for 2 seconds on not.

time.strftime

The **time** module has a **strftime** function that works in pretty much the same manner as the datetime version. The difference is mainly in what it accepts for input: a tuple or a **struct_time** object, like those that are returned when you call **time.gmtime()** or **time.localtime()**. Here's a little example:

```
import time

print(time.strftime("%Y-%m-%d-%H.%M.%S",time.localtime()))
#'2016-12-30-17.14.35'
```

This code is quite similar to the timestamp code we created in the datetime portion of this chapter. I think the datetime method is a little more intuitive in that you just create a **datetime.datetime** object and then call its **strftime** method with the format you want. With the time module, you have to pass the format plus a time tuple. It's really up to you to decide which one makes the most sense to you.

time.time

The **time.time** function will return the time in seconds since the epoch as a floating point number. Let's take a look:

```
import time

print(time.time())
#1483118130.25
```

That was pretty simple. You could use this when you want to save the current time to a database but you didn't want to bother converting it to the database’s datetime method. You might also recall that the **ctime** method

accepts the time in seconds, so we could use **time.time** to get the number of seconds to pass to ctime, like this:

```
import time

print(time.ctime(time.time()))
#'Fri Dec 30 17:16:14 2016'
```

If you do some digging in the documentation for the time module or if you just experiment with it a bit, you will likely find a few other uses for this function.

Wrapping Up

At this point you should know how to work with dates and time using Python’s standard modules. Python gives you a lot of power when it comes to working with dates. You will find these modules helpful if you ever need to create an application that keeps track of appointments or that needs to run on particular days. They are also useful when working with databases.