Python → Time → Time module

Theory: Time module

22 minutes 0 / 5 problems solved

Skip this topic Start practicing

321 users solved this topic. Latest completion was about 20 hours ago.

Sometimes you might want to have a time-related component in your program. For this purpose, Python has a built-in `time` module. Let's look at what you can do with it when working on a program that prints reminders.

§1. What time is it?

When we want to remind a user about something, we should also probably print the current time for them to know that the moment has actually come. To do that, we will import the module and use the `gmtime()` function:

```
import time

current_time = time.gmtime()
action = "to feed the cat"

print(f"It's {current_time}. Time {action}.")
```

This will print:

```
It's time.struct_time(tm_year=2020, tm_mon=9, tm_mday=16, tm_hour=13, tm_min=27, t
m_sec=28, tm_wday=2, tm_yday=260, tm_isdst=0). Time to feed the cat.
```

As you can see, the result is a little different from what we may have expected, as `gmtime()` returns a `struct_time` object. There are at least two possible ways to deal with it. Let's take a look at the information that will help us interpret the output:

value	meaning	range
tm_year	year	(for example, 2020)
tm_mon	month	from 1 to 12
tm_mday	day of the month	from 1 to 31
tm_hour	hour	from 0 to 23
tm_min	minute	from 0 to 59
tm_sec	second	from 0 to 61*
tm_wday	day of the week	from 0 to 6 (Monday is 0)
tm_yday	day of the year	from 1 to 366
tm_isdst	if DST is in a region (see below)	0 – no 1 – yes -1 – unknown

*60 is supported for representing a leap second, while 61 is there for historical reasons.

Getting back to our example, to rearrange the output, we can either use the `asctime()` function to convert the values into a string:

```
current_time = time.asctime(time.gmtime())  # first option

print(f"It's {current_time}. Time {action}.")
# It's Wed Sep 16 13:34:11 2020. Time to feed the cat.
```

Or we can use the `strftime()` function which also returns a string, but it can be used to specify the output format with the directives. For example, `%H` is used for hours, `%M` – for minutes and `%S` – for seconds.

Current topic:

Time module

Topic depends on:

- ✓ Type casting 12★
- ✓ String formatting Stage 1 7★
- ✓ Load module Stage 1 5★

Table of contents:

```
current_time = time.strftime("%H:%M", time.gmtime())  # second option

print(f"It's {current_time}. Time {action}.")
# It's 13:36. Time to feed the cat.
```

If you try that out yourself, you may notice that the time that the `gmtime()` returns is different from your current local. That is because the function gives you the time according to the UTC (which is an English-French acronym for Coordinated Universal Time) and this is probably not what you need. To print the current time for your region, use the `localtime()` function:

```
current_time = time.strftime("%H:%M", time.localtime())

print(f"It's {current_time}. Time {action}.")
# It's 16:36. Time to feed the cat.
```

§2. Time measurement

What if our user also has a dog that should be fed in an hour after the cat? In this case, with the `sleep()` function, we can just ask our program to wait for an hour and then remind the user about the dog. Let's add these lines to our program and look at the output:

```
action2 = "to feed the dog"

time.sleep(3600)  # NB! the time to wait is given in seconds

current_time = time.strftime("%H:%M", time.localtime())

print(f"It's {current_time}. Time {action2}.")
# It's 17:36. Time to feed the dog.
```

Most often `sleep()` is used when you need some time to pass between code executions.

The module also has another useful function - `time()`. It returns the time passed since the epoch (starting point for counting time) in seconds — it is a float object, but you can quickly convert it into a string of the same format as `asctime()` using `ctime()`. That is, the following lines give the same output:

```
print(time.asctime())           # Mon Oct 12 20:50:36 2020
print(time.ctime(time.time()))  # Mon Oct 12 20:50:36 2020
```

Note that depending on the platform, the reference point for the epoch might be different. On Windows and Unix systems, the epoch starts on January 1, 1970, 00:00:00 (UTC). You can check that typing `gmtime(0)`:

```
print(time.gmtime(0))  # on Windows

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

In the example above, as you can see, the `gmtime()` function takes an optional argument. It specifies the time passed since the beginning of the epoch (in seconds) which the function then converts into a `struct_time` object. If the argument is not provided, the current time returned by `time()` is used by default.

§3. Time difference

Knowing time in seconds might be useful, for instance, if you want to find the difference between two points of time. Now, let's imagine that the cat is trying to lose some weight and along with the current time we also want to print the time passed since the last time our user has fed the cat. This example is a little artificial since in a real program you would probably have some more code to execute between the first, starting point, and the next one, which marks the end of counting, but it still serves its demonstrative purpose:

```
last_time = time.time()
current_time = time.strftime("%H:%M", time.localtime())

print(f"It's {current_time}. Time {action}.")
# It's 17:36. Time to feed the cat.

time.sleep(300)

new_cur_time = time.time()

time_passed = int((new_cur_time - last_time) / 60)  # let's print the result in minutes



print(f"You have fed the cat {time_passed} minutes ago.")

# You have fed the cat 5 minutes ago.
```

In our case, it might be enough to subtract one point of time from another in seconds and represent it in minutes or hours for readability. But sometimes, working on your code, you need to know the precise time of your program execution. With the time module, this precision can be granted with the `perf_counter()` function:

```
start = time.perf_counter()

print("Oh no! The cat's breaking his diet!")

end = time.perf_counter()
total_time = end - start

print(f"Your program has executed for {total_time} seconds.")
# Oh no! The cat's breaking his diet!

# Your program has executed for 0.036370800000000036 seconds.
```

It may look similar to the `time()` function, but the time it returns is relative and is not related to the real-world-time, it is related to the processes inside hardware. This is why the `perf_counter()` should be used ***only for performance measurement***.

§4. Timezones and other peculiarities

Let's go back to our reminder program and imagine that we know that our user is now traveling, and that is why we want to let them know that the timezone has changed. This can be done using the `timezone`, which will show the time offset west of UTC, in seconds:

```
print(time.timezone / 3600)  # let's print the result in hours for readability
# -3.0
```

Alternatively, you can use the `tzname` which will print the abbreviated name of a new timezone:

```
print(time.tzname)
# ('RTZ 2 (winter)', 'RTZ 2 (summer)')
```

Sometimes, it also might be useful to know if time in that region depends on the season of the year (this is called DST – Daylight Saving Time). Let's add this information to our notification about the time changes with the help of the `daylight` which returns `0` if there is no DST and `1` otherwise:

```
print(time.daylight)
# 0
```

Keep in mind that the output of these functions depends on your region and might differ.

§5. Summary

Let's summarize what we have learned in this topic. We now know about:

- the Python `time` module;
- its `struct_time` object, which contains all information about current time from year to seconds;
- different ways to represent time (`asctime()`, `strftime()`, `ctime()`, `gmtime()`, `localtime()`);
- and several main functions that might be useful in various situations such as time and performance measurement (`time()`, `perf_counter()`), waiting for a certain time during program execution (`sleep()`) and getting information about region time peculiarities (`timezone`, `tzname`, `daylight`).

Now it's high time to go and try them in your program, and do not forget about Python docs if you strive to learn more.

Report a typo

35 users liked this theory. 2 didn't like it. What about you?

😍 🙂 😐 🙁 😡

Start practicing

Comments (1) Hints (1) Useful links (0) Show discussion