Python time Module

programiz.com/python-programming/time

Join our newsletter for the latest updates.

In this article, we will explore time module in detail. We will learn to use different time-related functions defined in the time module with the help of examples.

Python has a module named `time` to handle time-related tasks. To use functions defined in the module, we need to import the module first. Here's how:

```
import time
```

Here are commonly used time-related functions.

Python time.time()

The `time()` function returns the number of seconds passed since epoch.

For Unix system, `January 1, 1970, 00:00:00` at **UTC** is epoch (the point where time begins).

```
import time
seconds = time.time()
print("Seconds since epoch =", seconds)
```

Python time.ctime()

The `time.ctime()` function takes seconds passed since epoch as an argument and returns a string representing local time.

```
import time

# seconds passed since epoch
seconds = 1545925769.9618232
local_time = time.ctime(seconds)
print("Local time:", local_time)
```

If you run the program, the output will be something like:

```
Local time: Thu Dec 27 15:49:29 2018
```

Python time.sleep()

The `sleep()` function suspends (delays) execution of the current thread for the given number of seconds.

```
import time

print("This is printed immediately.")
time.sleep(2.4)
print("This is printed after 2.4 seconds.")
```

To learn more, visit: Python sleep().

Before we talk about other time-related functions, let's explore `time.struct_time` class in brief.

---

time.struct_time Class

Several functions in the `time` module such as `gmtime()`, `asctime()` etc. either take `time.struct_time` object as an argument or return it.

Here's an example of `time.struct_time` object.

```
time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27,
                 tm_hour=6, tm_min=35, tm_sec=17,
                 tm_wday=3, tm_yday=361, tm_isdst=0)
```

Index	Attribute	Values
0	`tm_year`	0000, ..., 2018, ..., 9999
1	`tm_mon`	1, 2, ..., 12
2	`tm_mday`	1, 2, ..., 31
3	`tm_hour`	0, 1, ..., 23
4	`tm_min`	0, 1, ..., 59
5	`tm_sec`	0, 1, ..., 61
6	`tm_wday`	0, 1, ..., 6; Monday is 0
7	`tm_yday`	1, 2, ..., 366
8	`tm_isdst`	0, 1 or -1

The values (elements) of the `time.struct_time` object are accessible using both indices and attributes.

Python time.localtime()

The `localtime()` function takes the number of seconds passed since epoch as an argument and returns `struct_time` in **local time**.

```
import time

result = time.localtime(1545925769)
print("result:", result)
print("\nyear:", result.tm_year)
print("tm_hour:", result.tm_hour)
```

When you run the program, the output will be something like:

```
result: time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27, tm_hour=15,
tm_min=49, tm_sec=29, tm_wday=3, tm_yday=361, tm_isdst=0)

year: 2018
tm_hour: 15
```

If no argument or `None` is passed to `localtime()`, the value returned by `time()` is used.

Python time.gmtime()

The `gmtime()` function takes the number of seconds passed since epoch as an argument and returns `struct_time` in **UTC**.

```
import time

result = time.gmtime(1545925769)
print("result:", result)
print("\nyear:", result.tm_year)
print("tm_hour:", result.tm_hour)
```

When you run the program, the output will be:

```
result = time.struct_time(tm_year=2018, tm_mon=12, tm_mday=28, tm_hour=8,
tm_min=44, tm_sec=4, tm_wday=4, tm_yday=362, tm_isdst=0)

year = 2018
tm_hour = 8
```

If no argument or `None` is passed to `gmtime()`, the value returned by `time()` is used.

Python time.mktime()

The `mktime()` function takes `struct_time` (or a tuple containing 9 elements corresponding to `struct_time`) as an argument and returns the seconds passed since epoch in local time. Basically, it's the inverse function of `localtime()`.

```
import time

t = (2018, 12, 28, 8, 44, 4, 4, 362, 0)

local_time = time.mktime(t)
print("Local time:", local_time)
```

The example below shows how `mktime()` and `localtime()` are related.

```
import time

seconds = 1545925769

# returns struct_time
t = time.localtime(seconds)
print("t1: ", t)

# returns seconds from struct_time
s = time.mktime(t)
print("\s:", seconds)
```

When you run the program, the output will be something like:

```
t1:  time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27, tm_hour=15, tm_min=49,
tm_sec=29, tm_wday=3, tm_yday=361, tm_isdst=0)

s: 1545925769.0
```

Python time.asctime()

The `asctime()` function takes `struct_time` (or a tuple containing 9 elements corresponding to `struct_time`) as an argument and returns a string representing it. Here's an example:

```
import time

t = (2018, 12, 28, 8, 44, 4, 4, 362, 0)

result = time.asctime(t)
print("Result:", result)
```

When you run the program, the output will be:

```
Result: Fri Dec 28 08:44:04 2018
```

Python time.strftime()

The `strftime()` function takes `struct_time` (or tuple corresponding to it) as an argument and returns a string representing it based on the format code used. For example,

```
import time

named_tuple = time.localtime() # get struct_time
time_string = time.strftime("%m/%d/%Y, %H:%M:%S", named_tuple)

print(time_string)
```

When you run the program, the output will be something like:

```
12/28/2018, 09:47:41
```

Here, `%Y` , `%m` , `%d` , `%H` etc. are format codes.

- `%Y` - year [0001,..., 2018, 2019,..., 9999]
- `%m` - month [01, 02, ..., 11, 12]
- `%d` - day [01, 02, ..., 30, 31]
- `%H` - hour [00, 01, ..., 22, 23
- `%M` - minutes [00, 01, ..., 58, 59]
- `%S` - second [00, 01, ..., 58, 61]

To learn more, visit: time.strftime().

Python time.strptime()

The `strptime()` function parses a string representing time and returns `struct_time` .

```
import time

time_string = "21 June, 2018"
result = time.strptime(time_string, "%d %B, %Y")

print(result)
```

When you run the program, the output will be:

```
time.struct_time(tm_year=2018, tm_mon=6, tm_mday=21, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=3, tm_yday=172, tm_isdst=-1)
```