Python time Module

The `time` module in Python provides functions for handling time-related tasks.

The time-related tasks includes,

- reading the current time
- formatting time
- sleeping for a specified number of seconds and so on.

Python time.time() Function

In Python, the `time()` function returns the number of seconds passed since epoch (the point where time begins).

For the Unix system, `January 1, 1970, 00:00:00` at **UTC** is epoch.

Let's see an example,

```
# import the time module
import time

# get the current time in seconds since the epoch
seconds = time.time()

print("Seconds since epoch =", seconds)

# Output: Seconds since epoch = 1672214933.6804628
```

In the above example, we have used the `time.time()` function to get the current time in seconds since the epoch, and then printed the result.

Python time.ctime() Function

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

```
import time

# seconds passed since epoch
seconds = 1672215379.5045543

# convert the time in seconds since the epoch to a readable format
local_time = time.ctime(seconds)

print("Local time:", local_time)
```

Output

```
Local time: Wed Dec 28 08:16:19 2022
```

Here, we have used the `time.ctime()` function to convert the time in seconds since the epoch to a readable format, and then printed the result.

Python time.sleep() Function

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

```
import time

print("Printed immediately.")
time.sleep(2.4)
print("Printed after 2.4 seconds.")
```

Output

```
Printed immediately.
Printed after 2.4 seconds.
```

Here's how this program works:

- `"Printed immediately"` is printed
- `time.sleep(2.4)` suspends execution for 2.4 seconds.
- `"Printed after 2.4 seconds"` is printed.

To learn more about `sleep()`, please visit Python sleep().

Python time.localtime() Function

The `localtime()` function takes the number of seconds passed since epoch as an argument and returns `struct_time` (a tuple containing **9** elements corresponding to `struct_time`) in **local time**.

```
import time
```

```
result = time.localtime(1672214933)
print("result:", result)
print("\nyear:", result.tm_year)
print("tm_hour:", result.tm_hour)
```

Output

```
result: time.struct_time(tm_year=2022, tm_mon=12, tm_mday=28, tm_hour=8, tm_min=8, tm_sec=53, tm_wday=2, tm_yday=362, tm_isdst=0)

year: 2022
tm_hour: 8
```

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

Python time.gmtime() Function

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(1672214933)
print("result:", result)
print("\nyear:", result.tm_year)
print("tm_hour:", result.tm_hour)
```

Output

```
result: time.struct_time(tm_year=2022, tm_mon=12, tm_mday=28, tm_hour=8, tm_min=8, tm_sec=53, tm_wday=2, tm_yday=362, tm_isdst=0)

year: 2022
tm_hour: 8
```

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

Python time.mktime() Function

The `mktime()` function takes `struct_time` (a tuple containing **9** elements corresponding to `struct_time`) as an argument and returns the seconds passed since epoch in local time.

The `struct_time` has the following structure:

```
(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
```

Let's see an example,

```
import time

time_tuple = (2022, 12, 28, 8, 44, 4, 4, 362, 0)

# convert time_tuple to seconds since epoch
seconds = time.mktime(time_tuple)

print(seconds)

# Output: 1672217044.0
```

Here, we have converted the `time_tuple` to seconds since the epoch.

Python time.asctime() Function

In Python, the `asctime()` function takes `struct_time` as an argument and returns a string representing it.

Similar to `mktime()`, the `time_tuple` has the following structure:

```
(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
```

Let's see an example,

```
import time

t = (2022, 12, 28, 8, 44, 4, 4, 362, 0)

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

# Output: Result: Fri Dec 28 08:44:04 2022
```

Here, we can see `time.asctime()` converts the time tuple to a human-readable string.

Python time.strftime() Function

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)
```

Output

```
12/29/2022, 08:36:22
```

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() Function

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

```
import time

time_string = "14 July, 2023"
result = time.strptime(time_string, "%d %B, %Y")

print(result)
```

Output

```
time.struct_time(tm_year=2023, tm_mon=7, tm_mday=14, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=4, tm_yday=195, tm_isdst=-1)
```

Here, `strptime()` parses a string and convert it to the `struct_time` object.

Table of Contents