

Day 46



Time Module in Python:

What is the time Module?

The time module in Python provides functions to:

- Work with time
- Get current time
- Measure execution time
- Pause program execution (delay)
- Convert time formats

It deals mostly with system time (seconds since epoch).

What is Epoch Time?

Epoch is a fixed starting point in time.

- In Python: January 1, 1970, 00:00:00 (UTC)
- Time is measured in seconds since epoch

Importing the time module:

Import time

Example: `time.time()` – Current Time in Seconds. Returns the current time in seconds since epoch.

```
1 import time
2 current_time = time.time()
3 print(current_time)
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
● PS E:\Python\Day41-50\Day46> python main.py
1766117509.418889
```

Example: `time.ctime()` – Readable Current Time. Converts seconds into a human-readable format.

```
1 import time
2 print(time.ctime())
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
• PS E:\Python\Day41-50\Day46> python main.py
Fri Dec 19 09:42:41 2025
```

Example: `time.sleep()` – Pause Execution. Pauses the program for a given number of seconds.

```
4 import time
5 print("Start")
6 time.sleep(2)
7 print("End")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
• PS E:\Python\Day41-50\Day46> python main.py
Start
End
```

Example: `time.localtime()` – Local Time Structure. Returns current local time as a `struct_time` object.

```
4 import time
5 t = time.localtime()
6 print(t)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
• PS E:\Python\Day41-50\Day46> python main.py
time.struct_time(tm_year=2025, tm_mon=12, tm_mday=19, tm_hour=9, tm_min=44, tm_sec=24, tm_wday=4, tm_yday=353, tm_isdst=0)
```

Example: `time.strftime()` – Formatting Time. Formats time into a readable string.

```
4 import time
5 t = time.localtime()
6 formatted = time.strftime("%d-%m-%Y %H:%M:%S", t)
7 print(formatted)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
• PS E:\Python\Day41-50\Day46> python main.py
19-12-2025 09:45:18
```

Example: measuring execution time.

```
8  import time
9  start = time.time()
10 for i in range(1000000):
11     pass
12 end = time.time()
13 print("Execution time:", end - start)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PC

```
PS E:\Python\Day41-50\Day46> python main.py
Execution time: 0.03531813621520996
```

Summary

- ✓ `time.time()` → current timestamp
- ✓ `time.sleep()` → delay execution
- ✓ `time.ctime()` → readable time
- ✓ `strftime()` → formatting
- ✓ `strptime()` → parsing
- ✓ `perf_counter()` → benchmarking

Creating command line utility in python :

What is a Command-Line Utility?

A command-line utility is a Python program you run from a terminal, for example:

```
python greet.py Alice
```

or later, more professionally:

```
greet Alice --uppercase
```

First CLI Program (Very Basic):

```
1 print("Hello from the command line!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Python\Day41-50\Day46> python .\hello.py
Hello from the command line!
```

Reading Command-Line Arguments (sys.argv): Python stores command-line arguments in sys.argv.

```
3 import sys
4 print(sys.argv)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Python\Day41-50\Day46> python hello.py ADitya
['hello.py', 'ADitya']
```

- sys.argv[0] → script name
- sys.argv[1] → first argument

Example: Simple CLI with arguments.

```
6 import sys
7 if len(sys.argv) < 2:
8     print("Usage: python hello.py <name>")
9     sys.exit(1)
10 name = sys.argv[1]
11 print(f"Hello, {name}!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Python\Day41-50\Day46> python hello.py
Usage: python hello.py <name>
PS E:\Python\Day41-50\Day46> python hello.py Aditya
Hello, Aditya!
```

Why argparse?

Manually handling `sys.argv` gets messy. Python provides `argparse`, which gives:

- `--help`
- error messages
- named options
- type checking

Example: `argparse` CLI

```
13 import argparse
14 parser = argparse.ArgumentParser(description="Simple greeting utility")
15 parser.add_argument("name", help="Name of the person to greet")
16 args = parser.parse_args()
17 print(f"Hello, {args.name}!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\Python\Day41-50\Day46> python hello.py
usage: hello.py [-h] name
hello.py: error: the following arguments are required: name
PS E:\Python\Day41-50\Day46> python hello.py --help
usage: hello.py [-h] name

Simple greeting utility

positional arguments:
  name      Name of the person to greet

options:
  -h, --help  show this help message and exit
PS E:\Python\Day41-50\Day46> python hello.py Aditya
Hello, Aditya!
```

--The End--