`!asyncio` --- Asynchronous I/O

asyncio

Hello World!

```
import asyncio

async def main():
    print('Hello ...')
    await asyncio.sleep(1)
    print('... World!')

asyncio.run(main())
```

asyncio is a library to write **concurrent** code using the **async/await** syntax.

asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc.

asyncio is often a perfect fit for IO-bound and high-level **structured** network code.

`a-conceptual-overview-of-asyncio`
Explanation of the fundamentals of asyncio.

asyncio provides a set of **high-level** APIs to:

- `run Python coroutines <coroutine>` concurrently and have full control over their execution;
- perform `network IO and IPC <asyncio-streams>`;
- control `subprocesses <asyncio-subprocess>`;
- distribute tasks via `queues <asyncio-queues>`;
- `synchronize <asyncio-sync>` concurrent code;

For **introspection**, asyncio provides APIs and tools for:

- inspecting the `async call graph <asyncio-graph>` of tasks and futures;

- inspecting tasks in another running Python process with `command-line tools <asyncio-introspection-tools>`;

Additionally, there are **low-level** APIs for *library and framework developers* to:

- create and manage `event loops <asyncio-event-loop>`, which provide asynchronous APIs for `networking <loop_create_server>`, running `subprocesses <loop_subprocess_exec>`, handling `OS signals <loop_add_signal_handler>`, etc;
- implement efficient protocols using `transports <asyncio-transports-protocols>`;
- `bridge <asyncio-futures>` callback-based libraries and code with async/await syntax.

not WASI.

This module does not work or is not available on WebAssembly. See `wasm-availability` for more information.

asyncio REPL

You can experiment with an `asyncio` concurrent context in the `REPL`:

```
$ python -m asyncio
asyncio REPL ...
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
'hello'
```

This REPL provides limited compatibility with `PYTHON_BASIC_REPL`. It is recommended that the default REPL is used for full functionality and the latest features.

cpython.run_stdin "" ""

3.12.5 (also 3.11.10, 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

3.13 Uses PyREPL if possible, in which case `PYTHONSTARTUP` is also executed. Emits audit events.

Reference

asyncio-runner.rst asyncio-task.rst asyncio-stream.rst asyncio-sync.rst asyncio-subprocess.rst asyncio-queue.rst asyncio-exceptions.rst

asyncio-graph.rst asyncio-tools.rst

asyncio-eventloop.rst asyncio-future.rst asyncio-protocol.rst asyncio-platforms.rst asyncio-extending.rst

asyncio-api-index.rst asyncio-llapi-index.rst asyncio-dev.rst asyncio-threading.rst

> [!NOTE] The source code for asyncio can be found in `Lib/asyncio/`.

□□□ #2
□ □□□ □□□ 2026-10-31 17:04:01 UTC
□ □□□ □□□ 2026-10-31 17:04:03 UTC