Developing with asyncio

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 1)

Unknown directive type "currentmodule".

```
.. currentmodule:: asyncio
```

Asynchronous programming is different from classic "sequential" programming.

This page lists common mistakes and traps and explains how to avoid them.

Debug Mode

By default asyncio runs in production mode. In order to ease the development asyncio has a *debug mode*.

There are several ways to enable asyncio debug mode:

- Setting the :envvar:`PYTHONASYNCIODEBUG` environment variable to `1`.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 26); *backlink*

  Unknown interpreted text role "envvar".

- Using the :ref:`Python Development Mode <devmode>`.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 28); *backlink*

  Unknown interpreted text role "ref".

- Passing `debug=True` to :func:`asyncio.run`.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 30); *backlink*

  Unknown interpreted text role "func".

- Calling :meth:`loop.set_debug`.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 32); *backlink*

  Unknown interpreted text role "meth".

In addition to enabling the debug mode, consider also:

- setting the log level of the :ref:`asyncio logger <asyncio-logger>` to :py:data:`logging.DEBUG`, for example the following snippet of code can be run at startup of the application:

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 36); *backlink*

  Unknown interpreted text role "ref".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 36); *backlink*

Unknown interpreted text role "py:data".

```
logging.basicConfig(level=logging.DEBUG)
```

- configuring the :mod:`warnings` module to display :exc:`ResourceWarning` warnings. One way of doing that is by using the :option:`-W` `default` command line option.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 42); *backlink*

  Unknown interpreted text role "mod".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 42); *backlink*

  Unknown interpreted text role "exc".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 42); *backlink*

  Unknown interpreted text role "option".

When the debug mode is enabled:

- asyncio checks for :ref:`coroutines that were not awaited <asyncio-coroutine-not-scheduled>` and logs them; this mitigates the "forgotten await" pitfall.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 49); *backlink*

  Unknown interpreted text role "ref".

- Many non-threadsafe asyncio APIs (such as :meth:`loop.call_soon` and :meth:`loop.call_at` methods) raise an exception if they are called from a wrong thread.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 53); *backlink*

  Unknown interpreted text role "meth".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 53); *backlink*

  Unknown interpreted text role "meth".

- The execution time of the I/O selector is logged if it takes too long to perform an I/O operation.
- Callbacks taking longer than 100 milliseconds are logged. The :attr:`loop.slow_callback_duration` attribute can be used to set the minimum execution duration in seconds that is considered "slow".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 60); *backlink*

  Unknown interpreted text role "attr".

Concurrency and Multithreading

An event loop runs in a thread (typically the main thread) and executes all callbacks and Tasks in its thread. While a Task is running in the event loop, no other Tasks can run in the same thread. When a Task executes an `await` expression, the running Task gets suspended, and the event loop executes the next Task.

To schedule a :term:`callback` from another OS thread, the :meth:`loop.call_soon_threadsafe` method should be used. Example:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 76); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 76); *backlink*

Unknown interpreted text role "meth".

```
loop.call_soon_threadsafe(callback, *args)
```

Almost all asyncio objects are not thread safe, which is typically not a problem unless there is code that works with them from outside of a Task or a callback. If there's a need for such code to call a low-level asyncio API, the :meth:`loop.call_soon_threadsafe` method should be used, e.g.:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 81); *backlink*

Unknown interpreted text role "meth".

```
loop.call_soon_threadsafe(fut.cancel)
```

To schedule a coroutine object from a different OS thread, the :func:`run_coroutine_threadsafe` function should be used. It returns a :class:`concurrent.futures.Future` to access the result:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 89); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 89); *backlink*

Unknown interpreted text role "class".

```
async def coro_func():
     return await asyncio.sleep(1, 42)

# Later in another OS thread:

future = asyncio.run_coroutine_threadsafe(coro_func(), loop)
# Wait for the result:
result = future.result()
```

To handle signals and to execute subprocesses, the event loop must be run in the main thread.

The :meth:`loop.run_in_executor` method can be used with a :class:`concurrent.futures.ThreadPoolExecutor` to execute blocking code in a different OS thread without blocking the OS thread that the event loop runs in.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 105); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 105); *backlink*

Unknown interpreted text role "class".

There is currently no way to schedule coroutines or callbacks directly from a different process (such as one started with :mod:`multiprocessing`). The :ref:`Event Loop Methods <asyncio-event-loop>` section lists APIs that can read from pipes and watch file descriptors without blocking the event loop. In addition, asyncio's :ref:`Subprocess <asyncio-subprocess>` APIs provide a way to start a process and communicate with it from the event loop. Lastly, the aforementioned :meth:`loop.run_in_executor` method can also be used with a :class:`concurrent.futures.ProcessPoolExecutor` to execute code in a different process.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 110); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 110); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 110); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 110); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 110); *backlink*

Unknown interpreted text role "class".

Running Blocking Code

Blocking (CPU-bound) code should not be called directly. For example, if a function performs a CPU-intensive calculation for 1 second, all concurrent asyncio Tasks and IO operations would be delayed by 1 second.

An executor can be used to run a task in a different thread or even in a different process to avoid blocking the OS thread with the event loop. See the :meth:`loop.run_in_executor` method for more details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 131); *backlink*

Unknown interpreted text role "meth".

Logging

asyncio uses the :mod:`logging` module and all logging is performed via the `"asyncio"` logger.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 142); *backlink*

Unknown interpreted text role "mod".

The default log level is :py:data:`logging.INFO`, which can be easily adjusted:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 145); *backlink*

Unknown interpreted text role "py:data".

```
logging.getLogger("asyncio").setLevel(logging.WARNING)
```

Detect never-awaited coroutines

When a coroutine function is called, but not awaited (e.g. `coro()` instead of `await coro()`) or the coroutine is not scheduled with :meth:`asyncio.create_task`, asyncio will emit a :exc:`RuntimeWarning`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst, line 156); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 156); *backlink*

Unknown interpreted text role "exc".

```
import asyncio

async def test():
    print("never scheduled")

async def main():
    test()

asyncio.run(main())
```

Output:

```
test.py:7: RuntimeWarning: coroutine 'test' was never awaited
  test()
```

Output in debug mode:

```
test.py:7: RuntimeWarning: coroutine 'test' was never awaited
Coroutine created at (most recent call last)
  File "../t.py", line 9, in <module>
    asyncio.run(main(), debug=True)

  < .. >

  File "../t.py", line 7, in main
    test()
  test()
```

The usual fix is to either await the coroutine or call the :meth:`asyncio.create_task` function:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 189); *backlink*

Unknown interpreted text role "meth".

```
async def main():
    await test()
```

Detect never-retrieved exceptions

If a :meth:`Future.set_exception` is called but the Future object is never awaited on, the exception would never be propagated to the user code. In this case, asyncio would emit a log message when the Future object is garbage collected.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 199); *backlink*

Unknown interpreted text role "meth".

Example of an unhandled exception:

```
import asyncio

async def bug():
    raise Exception("not consumed")

async def main():
    asyncio.create_task(bug())

asyncio.run(main())
```

Output:

```
Task exception was never retrieved
future: <Task finished coro=<bug() done, defined at test.py:3>
  exception=Exception('not consumed')>

Traceback (most recent call last):
  File "test.py", line 4, in bug
    raise Exception("not consumed")
Exception: not consumed
```

:ref:`Enable the debug mode <asyncio-debug-mode>` to get the traceback where the task was created:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)asyncio-dev.rst`, line 227); *backlink*

Unknown interpreted text role "ref".

```
asyncio.run(main(), debug=True)
```

Output in debug mode:

```
Task exception was never retrieved
future: <Task finished coro=<bug() done, defined at test.py:3>
    exception=Exception('not consumed') created at asyncio/tasks.py:321>

source_traceback: Object created at (most recent call last):
  File "../t.py", line 9, in <module>
    asyncio.run(main(), debug=True)

< .. >

Traceback (most recent call last):
  File "../t.py", line 4, in bug
    raise Exception("not consumed")
Exception: not consumed
```