<https://realpython.com/cpython-source-code-guide/>

[https://realpython.com/python-memory-management/](https://realpython.com/python-memory-management/#the-global-interpreter-lock-gil)

<https://realpython.com/pypy-faster-python/>

<https://realpython.com/python-gil/>

<https://realpython.com/pyspark-intro/>

<https://realpython.com/python-sockets/>

<https://realpython.com/python-concurrency/>

<https://realpython.com/python-async-features/>

<https://realpython.com/async-io-python/>

The PyObject, the grand-daddy of all objects in Python, contains only two things:

* ob\_refcnt: reference count
* ob\_type: pointer to another type

The reference count is used for garbage collection. Then you have a [pointer](https://realpython.com/pointers-in-python/) to the actual object type. That object type is just another struct that describes a Python object (such as a dict or int).

sys.getrefcount()

The general concept of asyncio is that a single Python object, called the event loop, controls how and when each task gets run. The event loop is aware of each task and knows what state it’s in. In reality, there are many states that tasks could be in, but for now let’s imagine a simplified event loop that just has two states. The ready state will indicate that a task has work to do and is ready to be run, and the waiting state means that the task is waiting for some external thing to finish, such as a network operation.

[https://realpython.com/python-memory-management/](https://realpython.com/python-memory-management/#the-global-interpreter-lock-gil)

<https://realpython.com/python-gil/>

<https://realpython.com/python-concurrency/>