**Multithreading**

**Threads:**

The [Thread](https://docs.python.org/3/library/threading.html#threading.Thread) class represents an activity that is run in a separate thread of control. There are two ways to specify the activity: by passing a callable object to the constructor, or by overriding the [run()](https://docs.python.org/3/library/threading.html#threading.Thread.run) method in a subclass. No other methods (except for the constructor) should be overridden in a subclass. In other words, only override the\_\_init\_\_() and [run()](https://docs.python.org/3/library/threading.html#threading.Thread.run) methods of this class.

threading.**Thread**(group=None, target=None, name=None, args=(), kwargs={}, \*, daemon=None)

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the [run()](https://docs.python.org/3/library/threading.html#threading.Thread.run) method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form “Thread-N” where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If not None, daemon explicitly sets whether the thread is daemonic. If None (the default), the daemonic property is inherited from the current thread.

If the subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.\_\_init\_\_()) before doing anything else to the thread.

**Methods of Thread Class**

1. **start**()

Start the thread’s activity. It must be called at most once per thread object. It arranges for the object’s [run ()](https://docs.python.org/3/library/threading.html#threading.Thread.run) method to be invoked in a separate thread of control. This method will raise a [RuntimeError](https://docs.python.org/3/library/exceptions.html#RuntimeError) if called more than once on the same thread object.

1. **run**()

Method representing the thread’s activity. You may override this method in a subclass. The standard [run()](https://docs.python.org/3/library/threading.html#threading.Thread.run) method invokes the callable object passed to the object’s constructor as the *target* argument, if any, with sequential and keyword arguments taken from the *args* and *kwargs* arguments, respectively.

1. **join**(*timeout=None*)

Wait until the thread terminates. This blocks the calling thread until the thread whose [join()](https://docs.python.org/3/library/threading.html#threading.Thread.join) method is called terminates – either normally or through an unhandled exception – or until the optional timeout occurs.

When the *timeout* argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As [join()](https://docs.python.org/3/library/threading.html#threading.Thread.join) always returns None, you must call [is\_alive()](https://docs.python.org/3/library/threading.html#threading.Thread.is_alive) after [join()](https://docs.python.org/3/library/threading.html#threading.Thread.join) to decide whether a timeout happened – if the thread is still alive, the [join()](https://docs.python.org/3/library/threading.html#threading.Thread.join)call timed out.

A thread can be [join()](https://docs.python.org/3/library/threading.html#threading.Thread.join)ed many times.

[join()](https://docs.python.org/3/library/threading.html#threading.Thread.join) raises a [RuntimeError](https://docs.python.org/3/library/exceptions.html#RuntimeError) if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to [join()](https://docs.python.org/3/library/threading.html#threading.Thread.join) a thread before it has been started and attempts to do so raise the same exception.

**threading.Lock**

The class implementing primitive lock objects. Once a thread has acquired a lock, subsequent attempts to acquire it block, until it is released; any thread may release it.

Note that Lock is actually a factory function which returns an instance of the most efficient version of the concrete Lock class that is supported by the platform.

**Methods of Lock class**

**acquire** (*blocking=True*, *timeout=-1*)

Acquire a lock, blocking or non-blocking.

When invoked with the *blocking* argument set to True (the default), block until the lock is unlocked, then set it to locked and return True.

When invoked with the *blocking* argument set to False, do not block. If a call with *blocking* set to True. would block, return False immediately; otherwise, set the lock to locked and return True.

When invoked with the floating-point *timeout* argument set to a positive value, block for at most the number of seconds specified by *timeout* and as long as the lock cannot be acquired. A *timeout* argument of -1 specifies an unbounded wait. It is forbidden to specify a *timeout* when *blocking* is false.

The return value is True if the lock is acquired successfully, False if not (for example if the *timeout* expired).

**release**()

Release a lock. This can be called from any thread, not only the thread which has acquired the lock.

When the lock is locked, reset it to unlocked, and return. If any other threads are blocked waiting for the lock to become unlocked, allow exactly one of them to proceed.

When invoked on an unlocked lock, a [RuntimeError](https://docs.python.org/3/library/exceptions.html#RuntimeError) is raised.

There is no return value.

**==================================================================**

threading.**active\_count**()

Return the number of [Thread](https://docs.python.org/3/library/threading.html#threading.Thread) objects currently alive. The returned count is equal to the length of the list returned by [enumerate()](https://docs.python.org/3/library/threading.html#threading.enumerate).

threading.**current\_thread**()

Return the current [Thread](https://docs.python.org/3/library/threading.html#threading.Thread) object, corresponding to the caller’s thread of control. If the caller’s thread of control was not created through the [threading](https://docs.python.org/3/library/threading.html#module-threading) module, a dummy thread object with limited functionality is returned.

threading.**get\_ident**()

Return the ‘thread identifier’ of the current thread. This is a nonzero integer. Its value has no direct meaning; it is intended as a magic cookie to be used e.g. to index a dictionary of thread-specific data. Thread identifiers may be recycled when a thread exits and another thread is created.

threading.**enumerate**()

Return a list of all [Thread](https://docs.python.org/3/library/threading.html#threading.Thread) objects currently alive. The list includes daemonic threads, dummy thread objects created by [current\_thread()](https://docs.python.org/3/library/threading.html#threading.current_thread), and the main thread. It excludes terminated threads and threads that have not yet been started.

threading.**main\_thread**()

Return the main [Thread](https://docs.python.org/3/library/threading.html#threading.Thread) object. In normal conditions, the main thread is the thread from which the Python interpreter was started.