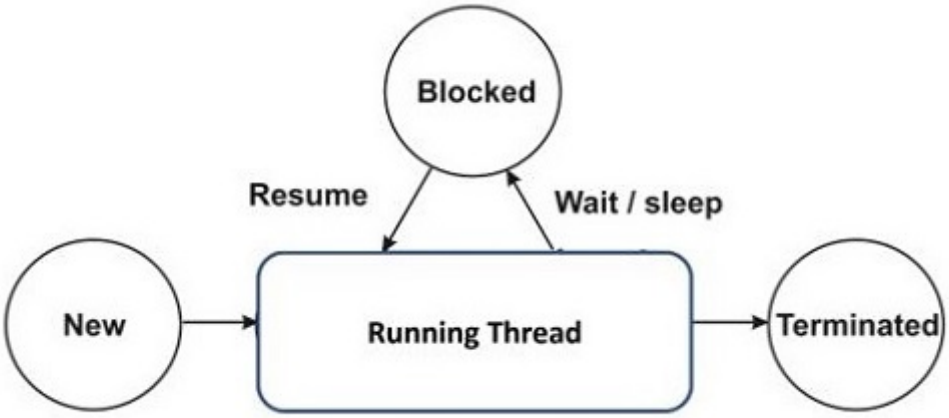
Python - Thread Life cycle

A thread object goes through different stages. When a new thread object is created, it must be started. This calls the run() method of thread class. This method contains the logic of the process to be performed by the new thread. The thread completes its task as the run() method is over, and the newly created thread merges with the main thread.

While a thread is running, it may be paused either for a predefined duration or it may be asked to pause till a certain event occurs. The thread resumes after the specified interval or the process is over.



Python's standard library has two modules, "_thread" and "threading", that include the functionality to handle threads. The "_thread" module is a low-level API. In Python 3, the **threading module** has been included, which provides more comprehensive functionality for thread management.

Python The _thread Module

The **_thread** module (earlier **thread** module) has been a part of Python's standard library since version 2. It is a low-level API for thread management, and works as a support for many of the other modules with advanced concurrent execution features such as threading and multiprocessing.

Python - The threading Module

The newer threading module provides much more powerful, high-level support for thread management.

The 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() method in a subclass.

```
threading.Thread(target, name, args, kwarg, daemon)
```

Parameters

- **target** – function to be invoked when a new thread starts. Defaults to None, meaning nothing is called.
- **name** – is the thread name. By default, a unique name is constructed such as "Thread-N".
- **daemon** – If set to True, the new thread runs in the background.
- **args and kwargs** – optional arguments to be passed to target function.