# I/O Bound Programs: Takeaways



## Syntax

- Initializing an in-memory database:

```
memory = sqlite3.connect(':memory:')
```

- Reading the contents of the disk database:

```
disk = sqlite3.connect('lahman2015.sqlite')
dump = "".join(line for line in disk.iterdump())
```

- Copying the database from disk into memory:

```
memory.executescrpt(dump)
```

- Creating a new thread:

```
thread = threading.Thread(target=task, args=(team,))
thread.start()
```

- Joining threads:

```
t1 = threading.Thread(target=task, args=(team,))
t2 = threading.Thread(target=task, args=(team,))
t3 = threading.Thread(target=task, args=(team,))
 # Start the first three threads
t1.start()
t2.start()
t3.start()
t1.join() # Wait until t1 finishes.
t2.join() # Wait until t2 finishes.  If it already finished, then keep going.
t3.join() # Wait until t3 finishes.  If it already finished, then keep going.
```

- Creating a lock:

```
lock = threading.Lock()
def task(team):
    lock.acquire()
     # This code cannot be executed until a thread acquires the lock.
    print(team)
    lock.release()
t1 = threading.Thread(target=task, args=(team,))
t2 = threading.Thread(target=task, args=(team,))
t1.start()
t2.start()
```

## Concepts

- CPU bound tasks are tasks where our Python program is executing something. CPU bound tasks will:
    - Execute faster if you optimize the algorithm.
    - Execute faster if your processor has a higher clock speed (can execute more operations).
- I/O bound tasks aren't using your CPU at all and waiting for something else to finish. I/O bound tasks are tasks where:
    - Our program is reading from an input (like a CSV file).
    - Our program is writing to an output (like a text file).
    - Our program is waiting for another program to execute something (like a SQL query).
    - Our program is waiting for another server to execute something (like an API request).
- A task is blocked when it's waiting for something to happen. When a thread is blocked, it isn't running any operations on the CPU.
- The hard drive is the slowest way to do I/O because it reads in data more slowly than memory and is much farther away from the CPU than memory.
- Threading allows us to execute tasks that are I/O bound more quickly. Threading makes CPU usage more efficient because when one thread is waiting around for a query to finish, another thread can process the result.
- Locking ensures that only one thread is accessing a shared resource at any time. The `threading.Lock.acquire()` method acquires the Lock and prevents any other thread from proceeding until it can also acquire the lock. The `threading.Lock.release()` method releases the Lock so other threads can acquire it.

- Examples of shared resources are:
    - The system stdout.
    - SQL databases.
    - APIs.
    - Objects in memory.

## Resources

- threading.Thread class
- Global Interpreter Lock
- Threading