Daily Coding Problem #10

Problem

This problem was asked by Apple.

Implement a job scheduler which takes in a function `f` and an integer `n`, and calls `f` after `n` milliseconds.

Solution

We can implement the job scheduler in many different ways, so don't worry if your solution is different from ours. Here is just one way:

First, let's try the most straightforward solution. That would probably be to spin off a new thread on each function we want to delay, sleep the requested amount, and then run the function. It might look something like this:

```
import threading
from time import sleep

class Scheduler:
    def __init__(self):
        pass

    def delay(self, f, n):
        def sleep_then_call(n):
            sleep(n / 1000)
            f()
        t = threading.Thread(target=sleep_then_call)
        t.start()
```

While this works, there is a huge problem with this method: we spin off a new thread each time we call delay! That means the number of threads we use could easily explode. We can get around this by having only one dedicated thread to call the functions, and storing the functions we need to call in some data structure. In this case, we use a list. We also have to do some sort of polling now to check when to run a function. We can store each function along with a unix epoch timestamp that tells it when it should run by. Then we'll poll some designated tick amount and check the list for any jobs that are due to be run, run them, and then remove them from the list.

```
from time import sleep
import threading

class Scheduler:
    def __init__(self):
        self.fns = [] # tuple of (fn, time)
        t = threading.Thread(target=self.poll)
        t.start()
```

```
    def poll(self):
        while True:
            now = time() * 1000
            for fn, due in self.fns:
                if now > due:
                    fn()
            self.fns = [(fn, due) for (fn, due) in self.fns if due > now]
            sleep(0.01)

    def delay(self, f, n):
        self.fns.append((f, time() * 1000 + n))
```

We'll stop here, but you can go much farther with this. Some extra credit work:

- Extend the scheduler to allow calling delayed functions with variables
- Use a heap instead of a list to keep track of the next job to run more efficiently
- Use a condition variable instead of polling (it just polls lower in the stack)
- Use a threadpool or other mechanism to decrease the chance of starvation (one thread not being able to run because of another running thread)
