0. We calculate l, the loss of water due to evaporation:

$$\frac{990 - l}{1000 - l} = 0.98$$

$$\begin{aligned} 990 - l &= 0.98(1000 - l) \\ 990 - 980 &= -0.98l + l \\ 10 &= 0.02l \end{aligned}$$

$$\therefore l = 500$$

   So the remaining mass is $1000 - 500 = 500$.

1. Googling "postgre detect slow running queries" returned `https://www.cybertec-postgresql.com/en/3-ways-to-detect-slow-queries-in-postgresql/`, which seemed well written at a glance.

   It suggests using the slow query log. The log is turned off by default, there are instructions to turn it on globally or for specific users or databases.

   There is also an explanation of the `auto_explain` module, which is again useful for tracking down isolated long running queries but additionally logs execution plans which can be examined to find the actual cause of long running queries. The example given shows queries that are essentially the same, however the first uses an index scan whereas the second uses a sequential scan, resulting in running times will be around a millisecond versus more like a second, respectively.

   The third option is `pg_stat_statements`. The usage reminded me of the `top` command for monitoring memory usage. You get a view that gives you fine grained, cumulative data on all the queries that run. It is not always great at identifying individual slow running queries, however.

2. `sort $1 | uniq -c | sort -nr | head -5 | cut -c9-`

3. Please see `Question3.js`. I assumed that there was no requirement to do it on one line this time.

4. I am no expert on networks so I read `http://www.linfo.org/network_segment.html`, which states that "each segment can be protected from the other segments by using firewalls...through which data moving between segments must pass." Therefore to test whether the fillwall is up, I would suggest pinging a known IP address outside of the segment, one of a public Internet site, say, from a computer inside the segment, assuming that it is known that ICMP, the protocol that supports ping, is not disallowed by the firewall.

5. Activities other than CPU usage can drain the battery. The screen, for example, together with services such as GPS and of course network usage. Making sure the application is not waking the screen or using these services unnecessarily, especially when there is no user input, is a good start.

   The question suggests that the reason for battery drain is CPU usage, however, therefore the best course of action is to reduce that. The Android SDK provides several options for profiling apps in development, documented at `https://developer.android.com/studio/profile`. iOS developers on the other hand can use XCode's Instruments, documented at `https://help.apple.com/instruments/mac/current/`.

6. Please see `Question6.c`.

7. The naïve answer would be 2 2 2, because the `++x` expression increments the value of `x` before it is evaluated and therefore evaluates to 2, the `x` expression leaves the value alone and also evaluates to 2, and the `x++` expression also evaluates to 2 prior to incrementing the value. However, the actual answer is 3 3 1, the order of evaluation being effectively right to left because of the cdecl calling convention.

8. -

9. Please see `https://aleph-one.com/Question9`.

10. Please see `Question10.js`.

11. Please see `Question11.sql`.

12. Please see `Question12.py`.

13. Please see `Question13.py`.