Robert A Fischer (Artie) - MWM - Smarty Pants Tester

artiefischer@gmail.com

Question 1

> You have a list/array of strings that represent dates and looks like this: [‘Oct 7, 2009’, ‘Nov 10, 2009’, ‘Jan 10, 2009’, ‘Oct 22, 2009’, …] The day is one or two digits (1, 2, … 31), with no preceding zero. There is always a comma after the day. The year is always four digits. Write a routine (in any language) that will order this list of strings in date descending order

```
// Input: [‘Oct 7, 2009’, ‘Nov 10, 2009’, ‘Jan 10, 2009’, ‘Oct 22, 2009’, …]

// Output = ['Dec 1, 2019', 'Sep 20, 2010', 'Nov 10, 2009', 'Oct 22, 2009', 'Oct 7, 2009', Jan 10, 2009']

const dateArr = ['Oct 7, 2009', 'Nov 10, 2009', 'Jan 10, 2009', 'Oct 22, 2009', 'Dec 1, 2019', 'Sep 20, 2010', 'Aug 2, 1912']

const dateSorting = (arr) => {
  const monthArr = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
  let result = []
  let tmp
  let dateObjArr = dateArrToObj(arr, monthArr)
  let sorted = sortDates(dateObjArr)

  // loop through sorted array and format w/ string literal - push to result array
  for (let i = 0; i < sorted.length; i++) {
    tmp = `${monthConversionToString(sorted[i].month, monthArr)} ${sorted[i].day}, ${sorted[i].year}`
    result.push(tmp)
  }
  return result
}

// converts date string to object {month, day, year}
const dateArrToObj = (arr, monthArr) => {
  let dateObjArr = []
  let day = 0
  for (let i = 0; i < arr.length; i++) {
    dayArr = arr[i].split(' ')[1].split('')
    for (let j = 0; j < dayArr.length; j++) {
      if (dayArr[1] == ',') {
          day = dayArr[0]
      } else {
        day = dayArr[0] + dayArr[1]
      }
```

```
    }
    dateObjArr.push({
      'month': monthConversionToNumber(arr[i].split(' ')[0], monthArr),
      'day': +day,
      'year': +(arr[i].split(' ')[2])
    })
  }
  return dateObjArr
}

// sort the date objects by key value
const sortDates = (arr) => {
  let ordered = []
   ordered = arr.sort((a, b) => {
    if (a.year === b.year) {
      if (a.month === b.month) {
        return a.day < b.day
      } return a.month < b.month
    } else {
      return a.year < b.year
    }
  })
  return ordered
}

// converts month string to integer value
const monthConversionToNumber = (str, monthArr) => {
  let num = 0
  for (let i = 0; i < monthArr.length; i++) {
    if (str === monthArr[i])
    num = i
   }
 return num
}

// converts the integer value back to month string
const monthConversionToString = (num, monthArr) => {
  let str = ''
  for (let i = 0; i < monthArr.length; i++) {
    if (num === i)
    str = monthArr[i]
   }
 return str
}

// call the f(x) with input array
dateSorting(dateArr)
```

Repl Solution

Golang Playground Solution for Problem 1

Question 2

> What are some ways to improve the security of a Unix/Linux system? Include general security guidelines and any specifics related to web servers and db servers

The most general approach to secure systems is to ensure access rights and passwords (hashed, stored, and checked against previously used passwords) are enforced on a lean and up-to-date machine. In a Unix system this can be even further 'shadowed' by utilizing the `etc/shadow` path to create a second level of hashing for the user group passwords. Additional protection can be applied to the system using a 'proactive security' utility tool such as `netstat -na`. This provides not only a performance analysis but also looks at open ports that may be susceptible to attack. A few other ways to improve security on the machine include: disabling the root user allows for a protected layer between users and their permissions, changing the default port removes an easy target for intruders (think changing the lock on your new house when you move in), and disable login after a certain number of attempts.

For databases typical security measures look for ways to prevent access to the database or a flaw in the programming through which to attack. To prevent SQL injections be sure to differentiate the SQL query from the input data from the site. This includes sanitizing and validating input field data, using a known SQL library to form queries such as Knex for Node or the http/net package in Go which automatically prevent from SQL injections. Also, keep SQL error or warning messages from the client, and the number one factor: limit access to your database from excellent password policy to the 'principal of least priviledge'.

For web server security many of the above approaches also apply from limited privileges to secure passsword. Other steps to secure web servers includes protecting information related to registry and event logs from anyone without permission to view the information. There are a large number of measures to lock down access from different points of attacks such as: hiding the name of the last user, restricting anonymous access, removing a default administrator, and regularly auditing the system as well as the policies guarding the system to uncover potential flaws (even on successful logins) and looking into unsuccesful attempts to see possible attacks.

There exists many built-in or easily adopted tools and frameworks to protect systems, databases, and servers. Generally I would stick to the guidelines of: excellet password policy, limited login attempts, system auditing, and restricting information to only those who absolutely need access.

Question 3

- The following answer only captures 9 of the 11 rows. Unfortunately, I was not able to capture the paradox entries because of the parent_id reference. To see a full .sql file of attempts please see: SQL.sql on github

```
WITH RECURSIVE last_run(parent_id, id_list, name_list) AS (
  SELECT parent_id, ARRAY[id] AS id_list, ARRAY[name] AS name_list FROM
category
  WHERE parent_id IS NULL
  UNION ALL
  SELECT category.parent_id, array_cat(ARRAY[category.id],
last_run.id_list), array_cat(ARRAY[category.name], last_run.name_list)
FROM last_run
  JOIN category ON last_run.id_list[1] = category.parent_id
  )
```

```
SELECT id_list, array_to_string(name_list, ', ') AS name_list FROM
last_run, category
WHERE last_run.name_list[1] = category.name
ORDER BY id_list;
```

- Output table for the above query

```
 id_list |       name_list
---------+-----------------------
{1}      | animal
{2}      | mineral
{3}      | vegetable
{4,1}    | dog, animal
{5,1}    | cat, animal
{6,4,1}  | doberman, dog, animal
{7,4,1}  | dachshund, dog, animal
{8,3}    | carrot, vegetable
{9,3}    | lettuce, vegetable
(9 rows)
```

Question 4

> turn a 100 x 100px red square 90 degrees with a click 🎈

See this JSFiddle for Red Square

or: https://jsfiddle.net/rafischer1/1yk0ouLc/19/

```
<div id="red"></div>

<br />
<article style="border: 3px solid aquamarine; padding: 2%; width: 30%;">
Browser support: <br /> [ie10+, edge 12+, FireFox 5+, Chrome 4+, safari
4+, ios saf 3.2+, opera all, android/blackberry/mobile all seem pretty
good to go] <br />
<a href="https://caniuse.com/#feat=css-animation" target="_blank">CANIUSE?
Table</a>
</article>
```

```
#red {
  background: radial-gradient(#FF4040, #FF0000);
  height: 100px;
  width: 100px;
}

#red:active {
```

```
  -webkit-animation: spin .2s linear;
  -moz-animation:    spin .2s linear;
  -o-animation:      spin .2s linear;
  animation:         spin .2s linear;
}

@-moz-keyframes spin {
    from { -moz-transform: rotate(0deg); }
    to { -moz-transform: rotate(90deg); }
}

@-webkit-keyframes spin {
    from { -webkit-transform: rotate(0deg); }
    to { -webkit-transform: rotate(90deg); }
}

@-o-keyframes spin {
    from { -webkit-transform: rotate(0deg); }
    to { -webkit-transform: rotate(90deg); }
}

@keyframes spin {
    from {transform:rotate(0deg);}
    to {transform:rotate(90deg);}
}
```

Question 5

> In your view, what are the pros and cons of TDD (test driven development). When do you think TDD makes more/less sense (if ever)?

TDD always and all the time. I learned a valuable lesson about rushing past TDD in my current job search. I had an application ineracting with a PSQL database and was using it as a 'profile' site. A possible employer opened the site and started clicking around - the site worked fine for about two minutes and then completely crashed. I did not get the job but I was able to spend a full afternoon debugging my golang backend routing. Eventually I pinpointed the problem: an unclosed database connection on one of the queries was leading my developer account on heroku to max out on connections (20 for the free account). As soon as the 'aha' and 'of course' moments passed I realized how simple that would have been to find in a small test package. TDD is time consuming and often feels like a whole new language to learn on top of the already daunting learning curve as it exists but the pros of having maintanable and debuggable ready-made code is worth the time especially in a production application. Even in a small, purely development oriented module TDD is helpful in defining your desired outcomes/results and while it is fun and often meaningful to improvise and go off on a purposeful tanget at times, TDD is a built-in process to keep the project on track.

Question 6

'Trying stuff until it works' - O'Reilly Joke Covers