COMP 2406: Fundamentals of Web Applications

Fall 2013 Mid-Term Exam Solutions

1. **(false)** HTTP cookies are only sent to a web server when explicitly requested.
2. **(false)** Cookies are normally parsed by client-side JavaScript.
3. **(true)** Arbitrary (even undefined) properties of objects can be accessed in JavaScript without generating a runtime error.
4. **(true)** `var` causes a symbol to be defined within the inner most function context.
5. **(false)** Write access to object properties are resolved by looking at the object's properties and then by looking at the chain of objects pointed to by the objects on the `prototype` chain.
6. **(true)** The `apply` function invocation pattern allows you to set the value of `this`.
7. **(true)** In general, HTML forms can be submitted successfully even if the browser has disabled JavaScript.
8. What HTTP command is normally used to request the contents of a page? **A: GET**
9. How could I modify this application so that the player's name is displayed on every room page rather than "player"?
   A: Change line 7 of `room.jade` **to have** `#{player}` **instead of** `player`
10. (2) In the `http` module, `createServer()` should be originally assigned as a method of what object? Why?
    A: It should have been assigned as a method of the `exports` **object, because it is the exports object that is returned when a module is** `require`**'d.**
11. (2) What port will this application listen on? Explain briefly how you know.
    A: It will listen on the port specified by the `PORT` **environment variable or on port 3000 if it is not defined. We know this because that is the value set for the port property in line 34.**
12. (15) How will program functionality change or what sort of error do you expect to see when the following lines are deleted from this application?
    i. app.js, lines 1-2
    A: The symbols `express`**,** `http`**, and** `app` **will no longer refer to objects (and will be undefined), and so a reference error will be generated the first time one of their methods is accessed, which is on line 34.**
    ii. app.js, line 4
    A: The symbols `i` **and** `theRoom`**, and** `loggedInUsers` **will no longer be defined in the app.js file scope and** `loggedInUsers` **will no longer be set to the empty object. No errors will be produced, however, as the loop on 45–48 will still work, with** `i` **and** `theRoom` **just being defined in the global scope.** `loggedInUsers` **not being defined won't make any difference as it is not referenced anywhere in the program.**
    iii. app.js, line 6
    A: Removing this line will mean the Engineering room is no longer defined. This will first cause problems when line 28 is called in response to the user logging in and attempting going to the Engineering room, as contents will be undefined.
    iv. app.js, line 11
    A: The server will return no response when / is requested and the user is logged in.

v. app.js, line 17
A: The user will never be logged in because the player property of the session object will never be set. Thus after logging in the engineering page will immediately redirect to / to have the user try to log in again.

vi. app.js, line 21
A: The user can never log out because the session will never be destroyed. Instead on quit it will redirect to / which will then redirect to the room they were previously in.

vii. app.js, line 35
A: No templates will be able to be found for jade, so the first render call, for / most likely, will generate some sort of error.

viii. app.js, line 37
A: No HTTP requests will be logged to the console.

ix. app.js, line 38
A: The body of HTTP POST requests won't be parsed, meaning `req.body` will not be defined. This will cause a reference error the moment the user tries to log in (line 17).

x. app.js, line 46
A: `theRoom` will not be defined (will give a value "undefined". In most programming languages this means we'd get an error the moment we try to append / onto it on line 47. However, this is JavaScript, so instead we'll get the string "/undefined". As it turns out, the program will run fine until the user logs in. At that point it will return an error that you cannot get /engineering (because we only defined handlers for /undefined).

xi. index.jade, line 4
A: The initial page, /, won't have a title shown in the browser's title bar. The rendered page will look the same otherwise.

xii. index.jade, line 12
A: There won't be a place to enter the player's name, and so the form will get submitted with an empty form value. This will cause an error on the server when it tries to access `req.body.player` in `start()`. (Yes, you should always check whether data coming from a client is correct and complete. This is an example of why.)

xiii. room.jade, lines 10-12
A: No exits will be listed for every room.

xiv. room.jade, lines 13-14
A: There will be no quit button on every room and thus no way to quit back to the login screen (except for restarting your browser).

xv. package.json, line 9
A: Express won't be installed in node_modules by npm, meaning the application will generate an error on the line where express in required (line 1 of app.js).

13. (4) List all functions—including methods—that are being *invoked* (directly called) on lines 50-51 and explain very briefly what each does. (Do not include any functions that are just being defined, are being called implicitly, or will be called during a later function invocation.)
A: The only functions that are being called on these lines are 1) `createServer()`, a method of `http` that creates and returns a server object, 2) `.listen()` which is a method of the returned server object that starts the web server listening on a given port, and 3) `app.get()` which returns the port to listen on. Note the anonymous function and all of the calls inside of it are called by `.listen()` after the server has started running.

14. (4) Jade code can include JavaScript code. What user-defined symbols can this code access? In other words, in what JavaScript scope does Jade code run in?

A: The only user-defined symbols Jade code can access are those symbols passed as properties of the object passed as the second parameter of the `res.render()` call that invokes the Jade template. Thus Jade runs in a scope that is isolated from the rest of the program (it is not "nested" in any other lexical context).

15. (4) What function is called to handle "/engineering"? How does this function get the information needed to populate the `room.jade` template? Explain.
A: The `handler()` function returned by `makeRoomHandler()` handles "/engineering". It gets the necessary information from the parameters passed to `makeRoomHandler()` in order to define a handler for "/engineering". Note these parameters are retained even though `makeRoomHandler()` has exited because the returned instance of `handler()` can still access them.

16. (10) Outline the control flow of this application, starting from when a user requests "/" from the application in a browser. Outline what the user sees and what happens in response to each input. Include what happens when the user logs in, navigates between rooms, etc. Be sure to cover each HTTP request (GET or POST) and each HTML page that is sent in response. Assume no errors.
A:

- **The user's browser sends a GET request for /**
- `index()` **processes the request, returns the initial login page (in HTML, rendered from the index.jade template) to the browser in response to the GET.**
- **The user enters their name and clicks the Start button. This generates a POST request for /start.**
- `start()` **processes the incoming post request and returns a redirect request to /engineering.**
- **The browser processes this redirect and submits a GET request for /engineering**
- `handler()` **instantiated for handling /engineering handles the request, returning the engineering page as defined by the room.jade template.**
- **If the user chooses to exit the room to sickbay, the click on the sickbay link generates a GET request for /sickbay. The response is handled similarly to that for /engineering.**
- **The user can move freely between sickbay and engineering by clicking on the only exit link. Each request generates a GET request to the server for the appropriate room which is then handled by an instance of `handler()`.**
- **If the user clicks the Quit button, it generates a POST request for /quit. This is handled on the server by `quit()` which destroys the session and returns a redirect response to / to the browser. The browser then sends a GET request for /.**

---

app.js:

```
1  var express = require('express'), http = require('http'),
2      app = express();
3
4  var i, theRoom, loggedInUsers = {};
5  var rooms = { activeRooms: ['sickbay', 'engineering'],
6                engineering: { title: "Engineering", roomExits: ['sickbay'] },
7                sickbay: { title: "Sickbay", roomExits: ['engineering'] } };
8
9  function index(req, res) {
10     if (req.session.player) {
11         res.redirect("/" + req.session.currentRoom);
12     } else {
13         res.render('index', { title: 'COMP 2406 Small Adventure Demo',
14                               error: req.query.error }); }}
15
```

```
function start(req, res) {
    req.session.player = req.body.player;
    res.redirect("/engineering") }

function quit(req, res) {
    req.session.destroy();
    res.redirect("/"); }

function makeRoomHandler(name, contents) {
    handler = function(req, res) {
        if (req.session.player) {
            req.session.currentRoom = name;
            res.render("room.jade", {title: contents.title,
                                     roomExits: contents.roomExits,
                                     player: req.session.player});
        } else { res.redirect("/"); }}
    return handler; }

app.set('port', process.env.PORT || 3000);
app.set('views', __dirname);
app.set('view engine', 'jade');
app.use(express.logger('dev'));
app.use(express.bodyParser());
app.use(express.cookieParser('COMP2406 is confusing!'));
app.use(express.session());

app.get('/', index);
app.post("/start", start);
app.post("/quit", quit);
for (i = 0; i<rooms.activeRooms.length; i++) {
//    theRoom = rooms.activeRooms[i];
    app.get('/' + theRoom,
            makeRoomHandler(theRoom, rooms[theRoom])); }

http.createServer(app).listen(app.get('port'), function(){
    console.log('Express server listening on port ' + app.get('port')); });
```

index.jade:

```
doctype 5
html
  head
    title= title
  body
    h1= title
    p Welcome to the #{title}
    p Please choose your player name
    div
      form(action="/start", method="post")
          div
            input(type="text", name="player")
            label.add-on(for="player") Player Name
          button(type="submit") Start
```

room.jade:

```
doctype 5
html
  head
    title= title
  body
    h1= title
    p You're on a ship, player.
```

```
    p Go to:
    ul
      each theExit in roomExits
        li
          a(href= theExit) #{theExit}
    form(action="/quit", method="post")
      button(type="submit") Quit
```

package.json:

```
{
  "name": "application-name",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "3.4.0",
    "jade": "*"
   }
}
```