Net ID:________________________ **Name:**____

á

CSCI-UA.0480-001 – Applied Internet Technology

Midterm Exam

October 20th, 2016

Instructor: Joseph Versoza

Keep this test booklet closed until the class is prompted to begin the exam

- Computers, calculators, phones, textbooks or notebooks are **not allowed** during the exam
- Please turn off your phone to avoid disrupting others during the exam
- Some questions allow you to make assumptions about what's already written or to use **shorthand for html by omitting close tag**, so read the instructions carefully
- Tear off the last page – it can be used as scrap/scratch paper and as a reference
- Choose to skip one of the last 4 questions (7 through 10) by marking it as DO NOT GRADE

1. Write out the full **HTTP interaction** involved in the following scenario. (9 points)

   a) A **user enters** the following **URL in the URL bar** of their **browser**: http://asliceof.pizza/coupons/123.
   b) The browser ends up displaying a page, but the page that it goes to is: http://asliceof.pizza/special-offer.
   c) The resulting page only has the following html: <h2>you get extra anchovies!</h2> (!?)
   d) (The browser behaves as if the **server told** the **browser** that it should **go to a page other than the one requested**).

   In the tables below, write out the HTTP requests and responses (there are 4 total) that transpired:

   - Specify "**request**" or "**response**" in the **first row**, and write the **entire HTTP** request or response **below** it
   - Write the **requests** and **response** in the **order that they occur** starting off with the initial request from the browser (note that the table columns are numbered)
   - Assume version HTTP/1.1
   - **Ignore** any requests/responses related "static" resources, such as **favicons**, **css**, **images**, etc. ...
   - If there are multiple valid status codes, any of those codes can be used
   - Only add the headers that are necessary for the interaction specified above

Request or Response?	1 **Request**	2 **Response**

Entire Source of Request or Response	`GET /coupon/123/special HTTP/1.1`
	
	`(can also have host header)`
		`HTTP/1.1 301 Moved Permanently`
		`Location: /special-offer`
		
		`(can also be any 3xx)`

Request or Response?	3 **Request**	4 **Response**

Entire Source of Request or Response	`GET /special-offer HTTP/1.1`
	
	`(can also have host header)`
		`HTTP/1.1 200 OK`
		`Content-Type: text/html`
		
		`<h2>You haz a pineapple!</h2>`

2. Give a short, one-sentence answer to the following questions about cookies and session management. (5 points)

   a) How are cookies created – that is... what mechanism causes your browser to create a cookie (be as specific as possible; include any references necessary for HTTP requests/responses)?

   The browser sends back a Set-Cookie header with name/value pairs and cookie options... all within an http response

   b) Why would you use the `HttpOnly` option for a cookie?

   To prevent JavaScript (specifically 3rd party) from reading cookies via document.cookie

   c) Once a cookie is created by the browser for a particular domain, what happens on every subsequent request to that domain from that browser (be as specific as possible; include any references necessary for HTTP requests/responses)?

   On every request, the browser sends a Cookie header with all of the cookies that it has for that domain separated by semicolons as the value of the header.

   d) Describe two characteristics of a good (read: *secure*) session id:

   It should not be easily guessable, and it should not be easily "stolen"

   e) Describe two situations where the server may create a new session id and send it back to the client:

   Browser did not send cookies, browser sent cookies... but no session id, or session id is sent... but not in session store.

3. **Read the code** in the left column. **Answer the questions** about the code in the right columns. Show your work if possible (for partial credit) (20 points)

Code

```
function playerInfo(includeScore) {
    console.log(this.name);
    if(includeScore) {
        console.log(this.score);
    }
}

var player = {name: 'pickles'};
playerInfo(false);
playerInfo.call(player, true);
var info = new playerInfo(false);
```

Question #1

a) What is the output of this program (error or nothing are possible, explain why if error)?

```
undefined
pickles
undefined
undefined
```

Question #2

b) **Rewrite** the **2nd to last line** (`playerInfo.call`) in the space below so that it uses **`apply`** instead of `call`.

```
playerInfo.apply(player,[true]);
```

Code

```
var makeCounter = function() {
    var count = 0;
    return {
        inc: function() {
            count += 1;
        },
        print: function() {
            this.inc();
            console.log('#' + count);
        }
    };
};
var counter = makeCounter();
for(var i = 0;i < 4; i++) {
    counter.print();
}
```

Question #1

c) What is the output of this program (error or nothing are possible, explain why if error)?

```
#1
#2
#3
#4
```

Question #2

d) What would that output of these two lines be if they were added to the end of the code in the first column?

```
console.log(typeof makeCounter);
console.log(typeof counter);
```

```
function
object
```

Code

```
function extractMediaType(f) {
    var i = f.lastIndexOf('.');
    var ext = f.substring(i + 1);
    var type;
    if(ext === 'css') {
        type = 'text';
    } else if (ext === 'jpg'){
        type = 'image';
    }
    console.log(format(type, ext));

    var format = function (t, s) {
       return t + '/' + s;
    };
}
extractMediaType('wavey.jpg');
```

Question #1

e) What is the output of this program (error or nothing are possible, explain why if error)?

```
error, format isn't a
function (declaration
is hoisted, but
definition is not)
```

Question #2

f) Rewrite the code that sets the variable, `type`, (based on an extension) so that it **doesn't** use an if / else if statement.

```
var MediaTypes = {
  css:'text',
  jpg:'image'
};

type = mediaTypes[ext];
```

Code

```
var arr1 = [98, 99, 100];

Array.prototype.last = function() {
    if(this.length > 0) {
        var i = this.length - 1;
        return this[i];
    }
};
console.log(arr1.last());

var arr2 = [];
console.log(arr2.last());

var r = arr2.hasOwnProperty('last');
console.log(r);
```

Question #1

g) What is the output of this program (error or nothing are possible, explain why if error)?

```
100
undefined
false
```

Question #2

h) What is the **[[prototype]]** of `Array.prototype.last`? What is the **[[prototype]] of the [[prototype]]** of `Array.prototype.last`?

```
Function.prototype
Object.prototype
```

Code

```
function App(port, connectMsg) {
    this.msg = connectMsg;
    var net = require('net');
    var s = net.createServer(this.fn);
    console.log('started');
    s.listen(port);
}
App.prototype.fn = function(sock) {
    console.log(this.msg);
};
var app = new App(3000, 'connected');
```

Question #1

i) Name 2 ways to connect to this running server (that is, name 2 *clients* you can use to communicate with this running server).

```
browser, curl, nc,
etc.
```

Question #2

j) When a client connects to this server, the server should log out `'connected'`. However, it logs out `undefined` instead. **Draw an arrow** to the line that needs to be rewritten, and **rewrite** it below:

```
var s = net.createServer(
  this.fn.bind(this)
)
```

4. What are the two HTTP request methods that we've used to send data to the server? When a form is submitted with either method, specify where in the *actual* HTTP request the data from the form elements is located. On the server side, what object / property in the Express framework contains that data? Lastly, when would you use one request method over the other? (4 points)

Method Name	**Where in HTTP Request is It Located?**	**Name of Obj /Prop in Express API**
`GET`	`Query string of url`	`req.query`
`POST`	`Body of http request`	`Req.body`

When would we use one request method over the other?	`GET for reading resources, POST for creating … or GET when you want "bookmarkable" links / allow for url hacking`

5. Answer the following database related questions. (4 points)

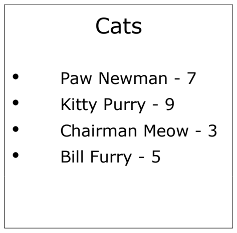
   a) Aside from a database, name two other potential mechanisms for persistent data storage (solutions that persist data only while the application are running are also acceptable):

   `the "cloud", local file system, in-memory`

   b) In the context of MongoDB, describe the following: **document**, **collection** and **database**

   `document is composed of key value pairs, collection is a group of documents, database is a group of collections`

6. You have an **Express application** that stores cat names. The path, `/cats`, should display a list of all of the cat names, along with their number of lives, that are stored in the database. The page that is displayed should look like this:


Cats

- Paw Newman - 7
- Kitty Purry - 9
- Chairman Meow - 3
- Bill Furry - 5

Finish the **route handler** and **template** below for `/cats`. (4 points)

a) the mongoose schema is defined in the 1st column (`db.js`)
b) **add the line to send back a response based on a template in the 2nd column** (`app.js`)
c) **finish the template so that it displays an unordered list of all of the cat names and their number of lives that are in the database** (`views/cats.hbs`)
d) you can assume that a all of the other required files and code (`app.listen(...)`, `layout.hbs`, etc.) are present

```
// db.js
// ----------

var Cat = new
mongoose.Schema({
 name: String,
 lives: Number
});

mongoose.model('Cat', Cat);
```

```
// app.js
// ----------

var Cat = mongoose.model('Cat');

app.get('/cats', function(req, res){

  var cb = function(err, cats, count){

     res.render('cats', {cats: cats});

  }


  Cat.find({}, cb);


});
```

```
// views/cats.hbs
// ----------
<h1>Cats</h1>

<ul>
{{#each}}
<li>{{name}} – {{lives}}</li>
{{/each}}
</ul>
```

Choose 3 out of the next 4 questions to answer. Mark the question you want to skip by writing DO NOT GRADE at the top of the question.

7. You have a directory called **`static`** in your Express project, and it contains html and css files. You'd like to implement a feature that lets you serve the files in that `static` folder without specifying explicit paths for each file. (18 points)

   Here's what your app will do:

   a) send the contents of files in the `static` folder back as http response bodies based on the url that was requested
   b) for example requesting http://your.site/baz.html would read the contents of `static/baz.html` and send it back as part of the response without having to define a route handler for `/baz.html`
   c) if the path specified in the url does not match a file in your static folder, then let the rest of your application handle the request (for example, if the original request was for '`/foo`', but `foo` doesn't exist in the `static` folder, your app may still be able to handle that path through a regular route handler like `app.get('/foo', …)`
   d) to implement this:
      - for every incoming request, regardless of path...
      - try to find the resource that the request is asking for in your app's static folder
      - you can assume that if the path doesn't have an extension of html or css, that the file is not in `static`
      - if the file exists, serve up the file's content (which can be html or css)
      - make sure the browser knows how to handle the contents of the request body!
      - if there's **any issue opening the file** (for example, if it doesn't exist)
      - just **allow the request to continue being handled** by the *rest* of your application
      - you **cannot use `express-static`** (this will essentially replicate the functionality of express static) or **`res.sendFile`**
      - **hints**:
        - you'll probably need to figure out the extension of the file... when you extract the extension, make sure that the dot (.) is not included, and also make sure to handle the case where there's no extension!
        - you'll need to use the `fs` module to open a file; this should be familiar from homework...
        - make sure you check the error object; it will let you know if there was an error reading the file:
        - fs module usage summary / refresher:

```
var fs = require('fs');
fs.readFile(fileName, callbackFunction);
// callbackFunction has two parameters:
// err – undefined/null if read worked, an object if there was an error
// for example, if they file were not found err would be an object
// data - the contents of the file on a successful read
```

   Write code that does this below. **Assume that all the Express setup requirements are already taken care of. Just add the code to implement this feature below**.

```
var fs = require('fs');
var extMediaType = {
    html:'text/html',
    css:'text/css',
    txt:'text/plain'
};

app.use(function(req, res, next) {
    var path = './static' + req.path;
    var ext;
    if(req.path.lastIndexOf('.') !== -1) {
        ext = req.path.substring(req.path.lastIndexOf('.') + 1);
    } else {
        next();
    }

    fs.readFile(path, function(err, data){
        if(err) {
            next();
        } else {
            res.set('Content-Type',extMediaType[ext] );
            res.send(data);
        }
    });
});
```

8. Implement two higher order functions, **`mapWith`** and **`any`**. **You must use two of the following built-in Array methods to implement these functions**: `forEach`, `filter`, `reduce`, or `map`. You can only use each one once. Remember that `break` does not work with any of these methods. (18 points)

   a) **`mapWith`**`(fn)`
      - `mapWith` **has a single parameter**, `fn`, which is a function that takes in a single argument and returns a value
      - `mapWith` **returns a function** that:
        - has 1 parameter, an `Array`
        - returns a new `Array` where every element is the result of calling `fn` on elements from the incoming `Array`

      This is different from *regular* map. The regular version of `map` immediately calls the callback function on every element in an Array to return a new `Array`. `mapWith`, on the other hand, gives back a function rather than executing the callback immediately (think of the difference between `bind` and `call`/`apply`). `mapWith` is basically a function that turns another function into a *mapping* function (a function that works on `Array`s). Here's how it works:

```
function square(n) {return n * n;}      // original square function that works on Numbers
mapWithSquare = mapWith(square);        // create a 'mapped' version of the square function
console.log(mapWithSquare([1, 2, 3]));  // now square can work on Arrays of Numbers!

mapWithParseInt = mapWith(parseInt);
console.log(mapWithParseInt([' 123', '45', '67   ']));
```

   b) **`any`**`(arr, func)` – returns true if *any* (at least one) of the elements in the Array, `arr`, pass the test, `func`. The test, `func`, is a function that takes a single argument, and it returns either true or false. Here's an example:

```
// prints out false (no elements pass the test function)
console.log(any([1, 2, 3, 4], function(num) {
    return num > 100;
}));

// prints out true (at least one of the elements passes the test function)
console.log(any([{name: 'joe'}, {color: 'green'}, {x: 42}], function(obj) {
    return obj.hasOwnProperty('x');
}));
```

```
function mapWith(fn) {
    var newFn = function(arr) {
        // var res = arr.map(fn);
        var res = [];
        arr.forEach(function(ele) {
            res.push(fn(ele));
        });
        return res;
    };
    return newFn;
}


function any(arr, fn) {
    return arr.reduce(function(accum, ele) {
       return fn(ele) ? true : accum;
    }, false);
}
```

9. Write a constructor that makes **`Request` objects**. Unlike our homework, this object **will not parse** an HTTP Request string. Instead, its **properties will be set** through **individual arguments** passed to the constructor. Additionally, it will have the **ability to send an actual http request** and **print** out the resulting response! (18 points)

   You will do this by using the net module as a client (see the example client code in the first column below) and by examining the example usage of a `Request` object in the second column.

<u>`Client Code with net Module Example`</u>

```
var net = require('net');
var client = new net.Socket();

// connect to port and domain
client.connect(3000, 'localhost', function() {

  // this code is only executed when this
  // client successfully connects to a server
  console.log('Connected');

  // write sends data to the server connected
  // to
  client.write('Example being sent');
});

client.on('data', function(data) {
  // this code is executed when the server
  // sends
  // data back to the client
  console.log('Received: ' + data);
  client.end();
});
```

<u>`Example Usage of Request Constructor and Object`</u>

```
// create a new Request object with constructor
var req = new Request('GET', 'www.google.com', 80);

// resulting object has these properties
console.log(req.method);  // GET
console.log(req.domain);  // www.google.com
console.log(req.port);    // 80
console.log(req.path);    // /

// the properties are defined on the object itself
console.log(req.hasOwnProperty('method')); // true

// the method, send, is defined elsewhere???
console.log(req.hasOwnProperty('send'));   // false

// send the request
req.send()

// prints out response from server
// HTTP/1.1 200 OK … etc.
```

a) Note that the resulting `Request` object has the following **properties**: **`method`**, **`domain`**, **`port`** and **`path`**
b) These properties are set from the **constructor's parameters**: **`method`**, **`url`** and **`port`**
c) `domain` and `path` are taken from `url` – you can assume that no protocol is included (no http://), everything before the 1st / is the `domain`, and everything after is the `path`. For example: foo.bar/baz/qux ^ `domain` is `foo.bar` and `path` is `/baz/qux`
d) If the domain doesn't contain a /, assume the whole url is the `domain`, and the `path` is `/`
e) The `Request` object's **`send`** method **connects to the server** specified by the `domain` and `port`, **sends a valid http** request (no need to add a Host header), and **prints out the response** when it receives data
f) You can copy most parts of the client code wholesale for your implementation, but you will have to make a few modifications

In particular, **watch out for the value of `this`** when using callbacks (either bind this to another name in the closure, make the variables you need available in the closure or use bind).

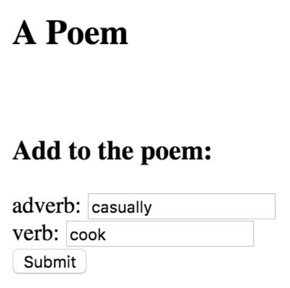
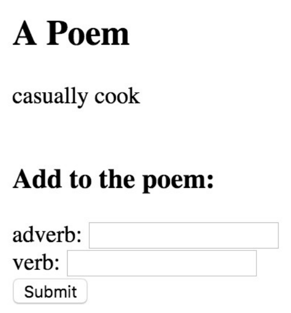
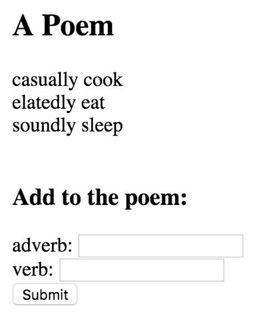
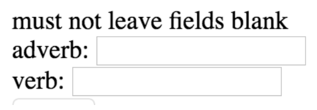
```
function Request(method, url, port) {
    var i = url.indexOf('/');
    this.path = '/';

    if(i == -1) {
        i = url.length;
    } else {
        this.path  = url.substring(i);
    }

    this.method = method;
    this.domain = url.substring(0, i);
    this.port = port;
}

Request.prototype.send = function() {
    var client = new net.Socket();
    var that = this;
    client.connect(this.port, this.domain, function(){
        var s = that.method + ' ' + that.path + ' HTTP/1.1\r\n\r\n';
        client.write(s);
    });
    // client.on('data') part from above
};
```

10. **Write the route handlers and the templates for a small Express application.** The application is a collaborative poem where users enter an adverb and a verb combination. Every combination entered by every user is displayed. (18 points)

 A Poem Add to the poem: adverb: casually verb: cook Submit	 A Poem casually cook Add to the poem: adverb: verb: Submit	 A Poem casually cook elatedly eat soundly sleep Add to the poem: adverb: verb: Submit	 must not leave fields blank adverb: verb:
The path `/poem` displays all of the lines of the poem (starting with no lines), along with a form to add a new line.	Submitting the form adds a line to the poem and takes you back to the original page (showing your new line added).	Lines can continually submitted to to construct a poem.	Leaving a field blank when submitting a form results in the form being re-rendered with an error message.

a) Assume that all required modules and configuration is already present (body-parser, hbs, etc.); you only have to write the routes, templates and any globals you'll need for persisting data
b) You can also assume that a views directory is setup with a layout.hbs file
c) When you write your templates, you can omit closing tags as a shorthand way of writing html
d) There's only one page in the entire application: `/poem`
e) You should not be able to refresh the page and cause a form submission to resubmit (with the exception of errors – see part g)
f) The poem looks the same for all users (that is, if I add a line on my browser, it will show up when you view the page on your browser). It's ok if all of the lines of the poem disappear when the server restarts
g) If either of the fields are left blank, display an error message above the form (in this case, it's ok if a refresh causes a resubmit)

```
var lines = [];
app.get('/poem', function(req, res) {
  res.render('index', {lines: lines});
});

app.post('/poem', function(req, res) {
  if(req.body.adverb && req.body.verb)  {
    lines.push({adverb:req.body.adverb, verb:req.body.verb});
    res.redirect('/poem');
  } else {
    res.render('index', {lines: lines, error:'must not leave fields blank'});
  }
});

<h2>A Poem</h2>
{{#each lines}}
{{adverb}} {{verb}}<br>
{{/each}}
</ul>


<h3>Add to the poem:</h3>
{{error}}
<form method="POST" action="">
               adverb: <input type="text" name="adverb" value="">  <br>
               verb: <input type="text" name="verb" value="">      <br>
               <input type="submit"> <br>
</form>
```

Name____________________________________ Net ID___________

Objects / Methods Reference

Array

properties

```
length
```

methods

```
pop()
reverse()
sort([compareFunction])
splice(index, howMany[, element1[, …[, elementN]]])
slice(index, howMany[, element1[, …[, elementN]]])
join([separator = ','])
concat(value1[, value2[, ...[, valueN]]])
indexOf(searchElement[, fromIndex = 0])
forEach(callback[, thisArg])
map(callback[, thisArg])
filter(callback[, thisArg])
reduce(callback[, initialValue])
some(callback[, thisArg])
every(callback[, thisArg])
```

String

properties

```
length
```

methods

```
split([separator][, limit])
toUpperCase()
slice(beginSlice[, endSlice])
replace(regexp|substr, newSubStr|
function[, flags])
substring(begin[, end])
indexOf(ch)
lastIndexOf(ch)
```

Object / Object.prototype

```
Object.getPrototypeOf(obj)
Object.prototype.hasOwnProperty(prop)
```

Request Object

properties

```
body
headers
method
path
query
session
url
```

methods

```
get
```

Response Object

methods

```
append
end
redirect
render
send
sendFile
set
status
writeHead
```