Net ID:_____ **Name:**_____

CSCI-UA.0480-010 – Applied Internet Technology

Midterm Exam

March 10th, 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
- **YOU CAN USE COMMENTS TO FILL IN SPECIFIC CODE FRAGMENTS**, such as generating code for setting up express-static; see reference page at end of exam
- **YOU CAN ASSUME THAT** an app object exists, and that, at the end of your program, app.listen(3000) is called; see reference page at end of exam for full details
- the last question description is long; it's attached to the reference material
- you can use a **shorthand for html by omitting close tags**

1. 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 HTTP request the data from the form elements is located. On the server side, what object / property represents 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
POST	Request body	req.body
GET	Request url (or query string)	req.query

When would we use one request method over the other?	POST for creating (or modifying) a resource / data, GET for reading a resource / data

2. Answer the following database related questions. (3 points)

   a) What's the **output** of the last command in this mongodb commandline session (**you can omit _id from the output**)?

```
> db.books.insert({title: 'anthem', author:'ayn rand'});
> db.books.insert({title: 'the fountainhead', author:'ayn rand'});
> db.books.insert({title: 'dune', author:'frank herbert'});
> db.books.update({title: 'the fountainhead'}, {author: 'pained hand'});
> db.books.find({author: 'ayn rand'});

{"title" : "anthem", "author" : "ayn rand" }
```

   b) In the context of MongoDB, what is the relationship between **databases**, **collections** and **documents / objects**?

Databases contain many collections... collections are groupings of documents

   c) MongoDB is a **document store**... name two other categories of **NoSQL** databases.

Key Value, Graph, Object, Column, etc.

3. **Name** 5 Classes of HTTP **Response Status Codes** and **describe** what each code means (the first is filled for you). (4 points)

1. 1xx _____ Informational
2. 2xx _____ Successful
3. 3xx _____ Redirect
4. 4xx _____ Client Error
5. 5xx _____ Server Error

4. Answer the following questions about **node modules** and **npm**.

   a) **Name and describe three JavaScript modules** that you've used (modules *required* in your programs) that are **not** `express`, `express-handlebars/hbs or express-static`. The description should indicate the functionality or features that the module provides. (3 points)

Module Name	Short Module Description
request	Library for programmatically making http requests
readline-sync	Synchronous library for collecting commandline input from user
body-parser	Populates req.body with data parsed from http request body

Others include fs, http, express-session, mongoose, etc.

   b) When installing a module with `npm` without any flags/options (for example `npm install express-handlebars`), the module is installed locally by default. **When** would you would install a module **globally** (system wide) and when would you install a module **locally** (specific to your project). (1 point)

All library dependencies local to project, any development tools (linters, build tools, code generators, etc.) globally.

5. The following statements are **true** or **false**; fill in the blanks appropriately. (6 points)

a) `!!0`	**false**	b) `[1, 2, 3].hasOwnProperty('toString')`	**false**
c) `4 === (undefined || 4)`	**true**	d) `The default port for http is 8000`	**false**
e) `typeof NaN === 'number'`	**true**	f) `'5' + 5 === '55'`	**true**

6. Name two features of **Express** that make it **easier** to work with than the **`http` module** (1 point)?

1. in http static files requires manual file reading with fs module, can use express-static for express
2. express automatically handles trailing slashes
3. express support templating (with jade, but also with other templating engines)
4. express has route handlers, whereas http requires a manual check against method explicitly

7. You have an Express application with several route handlers. However, **in addition to your defined route handlers, you'd like to be able to serve up arbitrary html pages**, by name, whose **content is located in files within your project folder**, under a **directory** called **`public`**. You want to be able to do this without having to explicitly define a separate route handler for each file in the public folder, though! (8 points)

   For example, imagine that your app has route handlers defined for `/foo` and `/bar`. In you `public` folder, you have `baz.html`, `qux.html`, and other html files. Your app should respond to `localhost:3000/foo` because of the explicit route handler defined... and it should respond to `localhost:3000/baz.html` without an explicit route handler. Your app should:

   a) **Check the path** of every incoming HTTP request...
   b) **Try to find the file** specified in the request path (attempt to find it within the your project's public folder)
   c) If the **file exists**, serve up the file's **content** as **HTML**...
   d) But if there's **any issue opening the file** (for example, if it doesn't exist)
   e) Just **allow the request to continue being handled** by the *rest* of your application
   f) Some hints / notes:
      - There's already something that does this (it shall remain nameless in this question); don't use it... implement your own!
      - You'll need to use the fs module to open a file:
        ```
        var fs = require('fs');
        fs.readFile(fileName, callbackFunction);
        // callbackFunction has two parameters:
        // err - undefined if read worked,an object if there was an error (like file not found)
        // data - the contents of the file on a successful read
        ```
      - When you send back the content of the file that you read, you'll have to explicitly set the content-type header to text/html: `res.set({ ...})`
      - Remember that `req.path` contains the path of the request!
   g) Write code that does this below. **Assume that all the Express setup requirements are already taken care of**, along with some defined routes. **Just add the code to implement this feature below**?

```
app.use(function(req, res, next) {
    fs.readFile('public' + req.path, function(err, data) {
        if(err) {
            next();
        } else {
            res.set({'content-type':'text/html'});
            res.send(data);
        }
    });
});
```

8. **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) (10 points)

Code	Question #1	Question #2

`function displayNumAndX(num) {`
`  console.log(this.x);`
`  console.log(num);`
`}`
 	
`displayNumAndX(80);`
 	
`var foo = {x:20, f:displayNumAndX};`
 	
`foo.f(42);`
	What is the output of this program (error or nothing are possible)? (2 points)
	
	undefined
	80
	20
	42
		Use either `call` or `apply` to invoke the function `displayNumAndX` with the argument 5 using the object below as this (1 point):
		
		`var bar = {x:0, y:0};`
		
		displayNumAndX.call(bar, 5);

`function outer() {`
`  inner('boo!');`
 	
`  var inner = function(thing) {`
`  console.log(thing);`
`  };`
`  return inner;`
`}`
`outer();`
	What is the output of this program (error or nothing are possible)? (1 point)
	
	Error – inner is not a function
		Why would the function, `outer`, be considered a higher order function? (1 point)
		
		It returns a function

`function Animal(legs) {`
`  this.legs = legs;`
`}`
 	
`Animal.prototype.hungry = true;`
 	
`var dog = new Animal(4);`
`var slug = new Animal(0);`
`console.log(dog.legs, dog.hungry);`
`console.log(slug.legs, slug.hungry);`
 	
`Animal.prototype.hungry = false;`
`dog.legs = 6;`
`console.log(dog.legs, dog.hungry);`
`console.log(slug.legs, slug.hungry);`
	What is the output of this program (error or nothing are possible)? (3 points)
	
	4 true
	0 true
	6 false
	0 false
		What would the following output? (1 point)
		
		`console.log(typeof Animal);`
		
		function
		
		Besides using constructors, what's another way to create an object that "inherits" the properties of another object? (1 point)
		
		Object.create

9. Implement two higher order functions, **`pluckWith`** 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. (10 points)

a) **`pluckWith`**`(prop)` – **returns a function** that takes a list of objects and picks out a value from each object that corresponds to the property name specified by the argument, `prop`. The result its a list of values. What was that? It's easier for me to show you:

```
var teams = [
    {team:'celtics', city:'boston'},
    {team:'pistons', city:'detroit'},
    {team:'kings', city:'sacramento'},
]

// creates a new function that picks out the team name from every object in a list of
// team objects... and gives back the team names as a list
var pluckTeamName = pluckWith('team');

// creates a new function that picks out the city from every object in a list of
// team objects and gives back the cities as a list
var pluckCity = pluckWith('city');

// our new functions can take a list of objects as an argument; let's call them on teams:

// prints out ['celtics', 'pistons', 'kings']
console.log(pluckTeamName(teams));

// prints out [''boston', 'detroit', 'sacramento']
console.log(pluckCity(teams));
```

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. Let's see another example:

```
// prints out true
console.log(any(teams, function(obj) {
    return obj.team == 'pistons';
}));

// prints out false
console.log(any(teams, function(obj) {
    return obj.team == 'knicks';
}));
```

```
function pluckWith(prop) {
    return function(arr) {
        return arr.map(function(ele) { return ele[prop] });
    }
}

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

10. Write the functions/constructors necessary to make the following code run. The output of each line is indicated by each line's

comment. Use the output and the example usage to guide your implementation. (10 points)

```
var p = new Person('Alice', 'Alvarez');
var s1 = new Student('Bob', 'Berman', 'bb123');
var s2 = new Student('Carol', 'Chan', 'cc456');

console.log(p.constructor);                 // prints: [Function: Person]
console.log(s1.constructor);                // prints: [Function: Student]

console.log(p.getFullName());               // prints: Alice Alvarez
console.log(s1.getFullName());              // prints: Bob Berman
console.log(s2.getFullName());              // prints: Carol Chan
console.log(s1.hasOwnProperty('first'));    // prints: true

console.log(p.doHomework);                  // prints: undefined
console.log(p.netID);                       // prints: undefined

s1.doHomework();                            // prints: bb123 says: "I'm busy doing homework"
s2.doHomework();                            // prints: cc456 says: "I'm busy doing homework"
```

```
function Person(first, last) {
    this.first = first;
    this.last = last;
}

Person.prototype.getFullName = function() {
   return this.first + ' ' + this.last;
}

function Student(first, last, netID) {
    Person.call(this, first, last);
    this.netID = netID;
}

Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;
Student.prototype.doHomework = function() {
    console.log(this.netID + " says: \"I'm busy doing homework\"");
}
```

11. **You just got a pet snake** (wait, what!!!???), and you're having trouble naming it. So, of course, you decide to crowd source your

snake's name by creating a site that allows users to suggest a name for it!!! (15 points)

See the description of this question on the reference page at the end of this exam.

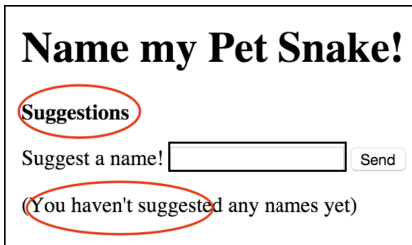
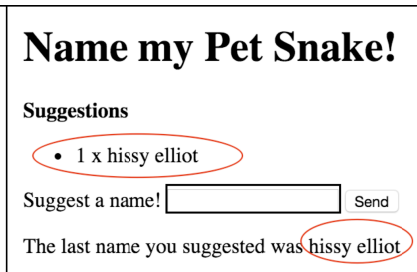
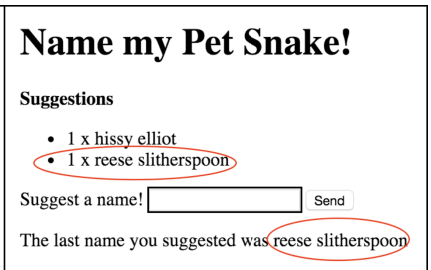
```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
var session = require('express-session');
app.set('view engine', 'hbs');
app.use(bodyParser.urlencoded({extended:false}));
var sessionOptions = {
    secret: 'secret for signing session id',
    saveUninitialized: false,
    resave: false
};
app.use(session(sessionOptions));

var pollResults = {};

app.get('/poll', function(req, res) {
    res.render('poll', {names:pollResults, last:req.session.lastSnakeName});
});

app.post('/poll', function(req, res) {
    if(pollResults[req.body.snakeName]) {
        pollResults[req.body.snakeName] += 1;
    } else {
        pollResults[req.body.snakeName] = 1;
    }
    req.session.lastSnakeName = req.body.snakeName;
    res.redirect('/poll');
});

app.listen(3000);
```

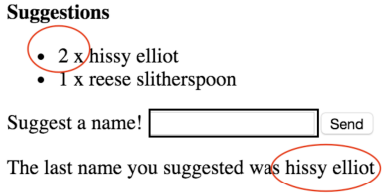
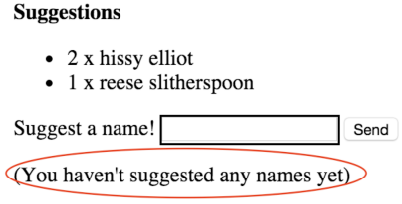
```
<h1>Name my Pet Snake!</h1>

<strong>Suggestions</strong>
<ul>
{{#each names}}
<li>{{this}} x {{@key}}</li>
{{/each}}
</ul>

<form method="POST" action="">
Suggest a name!
<input name="snakeName" type="text">
<input value="Send" type="submit">
</form>
{{#if last}}
The last name you suggested was {{last}}
{{else}}
(You haven't suggested any names yet)
{{/if}}


{{{body}}}
```

Description for question #11

You just got a pet snake (wait, what!!!???), and you're having trouble naming it. So, of course, you decide to crowd source your snake's name by creating a site that allows users to suggest a name for it. The application...

a) has a single page, **`/suggestions`**
b) it shows all of the suggested names along with the number of times that the name has been suggested
c) this list of suggestions is viewable by all users (and it's the same list for every user)
d) it has a form to submit snake name suggestions
e) it also shows the last name that was suggested by that user
f) if the user opens another browser, the last name suggested will be wiped out for that browser (but will remain for the previous)
g) **write the entire application, including templates, route handlers, and required modules**
   - for required modules, y**ou can use the shorthand** specified in the reference for requiring and configuring... but **only configure / require the bare minimum modules that you need**
   - identify the path and name of the file your code goes in (relative to your project's root folder)
   - you do not need to style the page
h) see the sample interactions below

		
Initial state: no suggestions, no last suggestion submitted.	If the user types in and submits `hissy elliot`, the resulting page shows that there's one suggestion, and that the user's last suggestion was `hissy elliot`.	If the user types in another name, it also shows up on the list, and the last suggested name is replaced by it.

	
If the name that the user enters already exists, then the quantity is incremented, and the last suggestion is updated.	If the user views the page in a different browser, the list stays the same, but the last suggested disappears!

Shorthand / Comments

You can use the following comments as shorthand to insert the code below. You can also assume that the app object is already created for you... and listen is called.

// body-parser

```
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({extended:false}));
```

// express-static

```
app.use(express.static('public'));
```

// hbs

```
app.set('view engine', 'hbs');
```

// express-session

```
var session = require('express-session');
var sessionOpts = {
    secret: 'secret stuff',
    saveUninitialized: false,
    resave: false
};
app.use(session(sessionOpts));
```

Assume this code exists!

```
var express = require('express');
var app = express();

# your code here!

app.listen(3000);
```

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])

Object

getPrototypeOf(obj)

hasOwnProperty(prop)

Request Object

properties

url

headers

method

path

query

body

session

Response Object

methods

writeHead

end

send

render

redirect

set

status