Applied Internet Technology (CSCI-UA.0480) - Sample Questions

- **A reference is provided on the last page.**
- **This does not represent the length of the actual midterm (this has more questions)**

1. Two broad categories of databases are relational and NoSQL. NoSQL databases can further be categorized by the way that they store their data:

   a) Name 3 categories / types of NoSQL databases

   `key-value, document, column, graph, object, etc.`

   b) What type of NoSQL database is MongoDB?

   `document`

2. In your Express projects, what are the following two files used for and what are their contents?

   a) `.gitignore` **Specifies what files should not be tracked (ignored) by git. The contents of the file are filenames (and *globs*), each separated by a newline**

   b) `package.json` **Stores the module dependencies of your project. The content is JSON generated from calling npm init and npm install --save**

3. Read the code in the 1st column. Answer questions about the code in the 2nd and 3rd columns.

Code	**Question 1**	**Question 2**

`var vegetable = 'kohlrabi';`
`var fruit = 'rambutan';`
 	
`var say_food = function() {`
`    vegetable = 'broccolini';`
`    var fruit = 'lychee';`
`    console.log(vegetable, fruit);`
`};`
 	
`console.log(vegetable, fruit);`
`say_food();`
`console.log(vegetable, fruit);`
	What is the output of the code on the left?
	
	`kohlrabi rambutan`
	`broccolini lychee`
	`broccolini rambutan`
		What is the type of the variable, `say_food`?
		
		`function`

`function foo() {`
`    console.log(arguments[1]);`
`    console.log(arguments[0]);`
`}`
 	
`foo('bar', 'baz', 'qux', 'quux');`
	What is the output of the code on the left?
	
	`baz`
	`bar`
		Besides `arguments` and the actual defined parameters, what other variable is added to the context of a function (that is, made available to the body of the function) when it is called?
		
		`this`

`function calculate(num) {`
`    var magicNumber = 5;`
`    return function(n) {`
`        return n * (num - magicNumber);`
`    };`
`}`
 	
`var calculateTen = calculate(10);`
`console.log(calculateTen(3));`
	What is the output of the code on the left?
	
	`15`
		What concept does the code on the left illustrate (what JavaScript feature allows the inner anonymous function to access particular variables)?
		
		closures

4. Create a form that `POSTS` to the path `/user/add`
   a) the form should have 2 input elements, a firstName and a lastName
   b) .....as well as a submit button

```
<form method="POST" action="/user/add">
first: <input type='text' name="firstName">
last: <input type='text' name="lastName">
<input type="submit">
</form>
```

5. You just bought the domain, `wholikespizza.com` (um, congratulations?)! You decide to create a single serving site on the domain. Your site contains just one page, and that page simply has the text '`ME`' on it. Create a barebones Express application to run your site by following the specifications below:

   a) all of the code will be in a single file called `app.js`
   b) your app will **respond to `GET`** requests on **`/`** (the root directory of your site)
   c) the **body** of your app's response will be the text, '**`ME`**' (**no views** or templates necessary)
   d) it should **listen on port 80**
   e) remember to write up all of the setup code necessary for a barebones Express application (even starting with the require!)
   f) write the contents of `app.js` below:

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

// create app
var app = express();

// create route
app.get('/', function(req, res) {
       res.send('ME');
});

// listen
app.listen(80);
```

6. Describe how inheritance works in JavaScript. What mechanism is used to create objects that inherit properties from *parent* objects. If a property is not found in an object, where will JavaScript continue to look to find that property?

JavaScript uses prototypes for inheriting properties from another object (its prototype). An object's prototype is set through either the argument passed in to `Object.create(proto)` or through the constructor (which is just a function called with `new`) that the object was created from (via its constructor's `prototype` property).

JavaScript will search up the prototype chain until it reaches `Object.prototype`.

7. Name one HTTP request headers and one HTTP response headers. Explain what each represents:

   a) `user-agent - For a request, a string representing information about the client, such as name, version number, etc. Typically, for browsers, it's the browser name and version.`

   b) `content-type - For a response, the internet media type of the content.`

8. Name three ways that functions are invoked, and explain what the variable, `this`, is set to in each case. The first box is already filled in, complete the remaining five boxes:

Invocation	`this`
***Regular* function call**	**the global object (in the case of node, the module level *global* object)**
Method call	**the object the method was called on**
call/apply	**the `this` argument passed in to call/apply (you can set explicitly)**

9. Using the following code, write out the output and **briefly** explain the reason for the output in the table below.

```
var obj = Object.create(Array.prototype);
obj.foo = 'bar';
console.log((typeof obj.foo));
console.log((typeof obj.baz));
console.log((typeof obj.push));
console.log(obj.hasOwnProperty('foo'));
console.log(obj.hasOwnProperty('push'));
console.log(obj.hasOwnProperty('toString'));
```

#	Output	Reason
1	string	obj.foo's value is a string
2	undefined	obj does not have a property named baz
3	function	obj's prototype is Array.prototype, which contains the method, push (a function)
4	TRUE	foo was defined on obj, so it's an 'own property'
5	FALSE	push comes from obj's prototype, so it's not an 'own property'
6	FALSE	toString comes from Object.prototype, so it's not an 'own property'

10. You're a mad scientist that loves stitching together parts of Arrays to make new Franken-arrays! To aid in your Mary Shelley-esque experiments, you create a function called `joinHalves`.

a) The function should have **two parameters**, **`firstArray`** and **`secondArray`**, with both expected to be `Arrays` (of course!)
b) There is no need to validate or check the type of the incoming arguments.
c) The function will **return a new `Array`** consisting of the **first half of `firstArray`** and **second half of `secondArray`**
d) If there are an odd number of elements, use `Math.floor` or `Math.ceil` appropriately so that lesser elements are taken / added to the new `Array`. For example
   - `firstArray: [1, 2, 3, 4, 5, 6, 7]` ... would contribute `[1, 2, 3]` to the beginning of the new Array
   - `secondArray: ['a', 'a', 'a']`... would contribute `['a']` to the second half of the new Array
   - The result of `join_halves([1, 2, 3, 4, 5, 6, 7], ['a', 'a', 'a']) would be [1, 2, 3, 'a']`

```
var joinHalves = function(firstArray, secondArray) {
    // extract the first part from the first array
    var firstHalf = firstArray.slice(0, Math.floor(firstArray.length / 2));

    // get the second half from the second array
    var secondHalf = secondArray.slice(Math.ceil(secondArray.length / 2));

    // put the two together in frankenArray
    frankenArray = firstHalf.concat(secondHalf);
    return frankenArray;
};

// or ...
var joinHalves = function(a, b) {
       return a.slice(0, Math.floor(a.length / 2)).concat(b.slice(Math.ceil(b.length / 2)));
};
```

11. Debug the following code. You have an Express app using handlebars as a templating engine to generate html. An image on one of your pages is showing up as a broken image icon...

a) What are some steps that you would take to debug this problem. Be exact – discuss the tools that you would use and what you're looking for.

   - View source or check the network tab in developer tools to see what the url for the image is.
   - Check what status code the image returns by requesting it in the browser, looking at the network tab or using curl
   - Log out the url that the app is receiving (perhaps using middleware)

b) What are some possible errors (that is, where can these errors occur?); again, be as exact as you can.

   - If it's a 404
     - maybe the link to the image is just incorrect (typo, wrong extension, absolute vs relative)
     - maybe there's no route handler for the image path or express-static wasn't enabled
     - maybe it doesn't exist on the file system
   - If it's a 500 and you're manually reading files in, then there may be an error/exception in the file reading code

12. Describe two reasons for using a separate templating engine for rendering an HTML document rather than emitting HTML directly as a string from within your application code?

Generating html in your application code can get very complex, even for simple documents.

Separating application logic from presentation logic helps reduce side-effects by avoiding a close integration between logic and presentation.

13. Create a site using Express and handlebars templates:

a) It should **respond to `GETS` on** two URLs: **`/old`** and **`/new`**
b) **`/old`** will **redirect** to **`/new`**
   - the response status code can be any of the redirect class status codes (or the default if your implementation doesn't require an explicit status code to be sent)

c) `/new` will display a list of names
   - as an unordered list in HTML)
   - these names can be stored as a global variable in your application
   - the names in the list are: 'Gil', 'Jill', 'Bill' and 'Phil'
   - this global variable can then be passed as context when your template is rendered
   - assume that the **template** or view that you specify will be **called `names.handlebars`**

d) Write the contents of the following files in your project to implement the specifications outlined above:

`app.js`

```
// omit setup code (imagine that it is already present above for express and handlebars)

// define your two routes below this line

nameList = ['Gil', 'Jill', 'Bill', 'Phil'];

app.get('/old', function(req, res) {
    res.redirect('/new');
});

app.get('/new', function(req, res) {
    res.render('names', {'nameList':nameList});
});
```

`/views/names.handlebars`

```
<ul>
{{#each nameList}}
       <li>{{this}}</li>
{{/each}}
</ul>
```

`/views/layouts/main.handlebars`

```
<!doctype html>

<html>
<head>
    <title></title>
</head>

<body>
    {{{ body }}}
</body>
</html>
```

14. Answer the questions in the 2nd column about the code in the 1st column:

Code	**Questions**

`function Foo(num) { this.x = num;}`
`Foo.prototype.printX = function()`
`{ console.log(this.x); }`
`function Bar(num1, num2) {`
`    Foo.call(this, num1);`
`    this.y = num2;`
`}`
`Bar.prototype =`
`Object.create(Foo.prototype);`
`var b = new Bar(100, 200);`
`b.printX();`
`console.log((typeof Foo))`
	What is the output of this program?
	
	100
	function
	
	What would this refer to if the keyword new were omitted?
	
	The global object

15. What does the **`Array`** method, **`map`**, do? **Implement your own version of `map`** (call it `myMap`). Your method will have an `Array` as one parameter, and a function as the other parameter. It will do the same thing that the actual `Array map` method does (but with the `Array` object passed in as a parameter, rather than as an object that the method is called on). For example, the following two should be equivalent:

(assuming numbers is an `Array`, like `[2, 4, 6, 8]`)

```
a) numbers.map(myCallback);
b) myMap(numbers, myCallback);
```

Implement `myMap` below:

```
function map(arr, transform) {
    var transformed = [];
    arr.forEach(function(element) {
        transformed.push(transform(element));
    });
    return transformed;
}
```

16. Write an example URL, and label each part of the URL below (`http://` is already filled out; there are 6 parts total)

Labels: `1)__`**`protocol/schema`**`_____`   `2)______`**`port`**`_________`   `3)__`**`query string`**`____`

Example URL: **`http://`** `_`**`wholikespizza.com`**`___` `__`**`:8080`**`__` `___`**`/pizza/`**`_________`**`?crust=thin`**`_` `__`**`#menu`**`____`

Labels: `2)_____`**`domain`**`_________`   `5)____`**`path`**`________`   `6)_`**`fragment id`**`___`

17. Imagine that you have a collection called **`links`** in your database. It contains documents that look like:

```
{ name: …, url: ….}
```

a) Using the commandline client, add a new link document with name: snowman … and url: http://unicodesnowmanforyou.com.

```
db.links.insert( {name: 'snowman', url: 'http://unicodesnowmanforyou.com'} )
```

b) Assuming that there are other links in the collection, write a query using the commandline client that retrieves a single link (it doesn't matter which one, just a single link, though)

```
db.links.findOne()
```

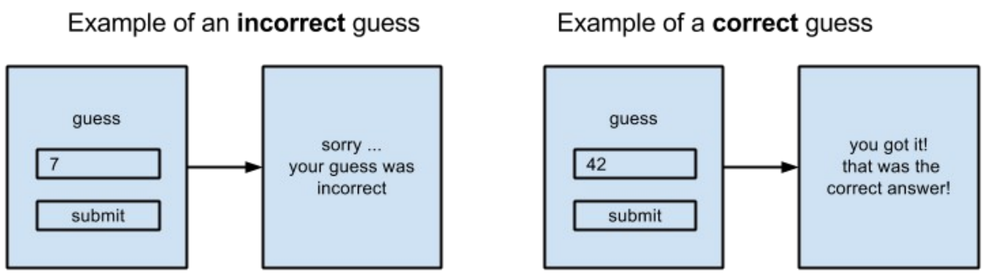
c) Using the Mongoose API, get all of the links in the collection and simply log out the results with console.log once the query gives back a result. Assuming that a schema and model constructor already exist, and you already have the setup code:

```
var Link = mongoose.model('Link');
```

```
Link.find(function(err, links, count) {}){
    console.log(links);
}
```

18. **Create a web app** that's a **number guessing game**. At minimum, it should have:

* a regular Express app (along with its basic setup/configuration) to handle incoming requests
* a global variable in your app that represents a secret number
* a **form** where a user can submit a number
* after the submission, some way of determining whether or not the person's matches the secret number stored globally
* a **page** or **pages** that **shows if you've guessed right**... or if you've guessed incorrectly
* it's up to you to determine how many routes you'd like to make, as well as what requests you'd like to handle
* here are some examples of how the user may flow through win and lose scenarios:


Example of an **incorrect** guess
guess
7
submit
sorry ... your guess was incorrect
Example of a **correct** guess
guess
42
submit
you got it! that was the correct answer!

a) **Write the setup code** and **routes** that would go in your **`app.js`** file. However, in place of the setup code for handlebars and the module that allows you to access the body property of the request, just write in `"// handlebars setup" or "// request.body"` where appropriate.

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

// handlebars setup
// request.body

var secret = 42;

app.get('/', function(req, res) {
        res.render('index');
});

app.post('/', function(req, res) {
        if (req.body.num == secret) {
                res.redirect('/win');
        } else {
                res.redirect('/lose');
        }
});

app.get('/win', function(req, res) {
        res.render('win');
});

app.get('/lose', function(req, res) {
        res.render('lose');
});

app.listen(3000);
```

b) Imagine that your handlebars layout file is already created (along with the app.js file you've coded above). Use the routes and callbacks you've defined in your `app.js` file to create the **template files and the mark-up** (this depends on your implementation) that you'd need to use so that all pages in your app are rendered correctly. **Include both** the template's **path** (relative to the project root) and **name**, as well as its **contents**. For example: (5 points)

```
name and path – path/to/myViews/myFancyTemplate.handlebars
contents – <p>Some content</p>
```

```
<!-- index.handlebars –>
<form method="post" action="/">
<input type="text" name="num">
<input type="submit">
</form>

<!-- win.handlebars -->
You got it!

<!-- lose.handlebars -->
Sorry …
```

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
```