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)**
- **This does not represent the exact topics being covered**

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

Code	Question 1	Question 2

`let vegetable = 'kohlrabi';`
`let fruit = 'rambutan';`
 	
`const say_food = function() {`
`  vegetable = 'broccolini';`
`  let 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? Error is possible.
	
	`kohlrabi rambutan`
	`broccolini lychee`
	`broccolini rambutan`
		What is the type of the variable, `say_food`? (that is, what is the result of `typeof say_food`.
		
		`Function (object is technically correct too)`

`function foo(bar, ...qux) {`
`  console.log('bar', bar);`
`  console.log('qux', qux);`
`}`
 	
`let arr = ['a', 'b', 'c', 'd'];`
 	
`foo(...arr);`
	What is the output of the code on the left?
	
	`bar a`
	`qux [ 'b', 'c', 'd' ]`
		What would the value of the parameter, `bar`, be if `foo` were called without arguments – `foo();`?
		
		`undefined`

`function f(num) {`
`  const magic = 5;`
`  function g(n) {`
`  return n * (num - magic);`
`  }`
`  return g;`
`}`
 	
`const calculateTen = f(10);`
`console.log(calculateTen(3));`
	What is the output of the code on the left?
	
	`15`
		What is the term / name used to describe a function and the environment that I was created it (that is, what JavaScript feature allows the inner anonymous function to access variables in the outer function's local scope)?
		
		closures

`class A {`
`  foo() { console.log('foo'); }`
`}`
 	
`class B extends A {`
`  constructor() {`
`  super();`
`  this.bar = 'bar';`
`  }`
`  qux() {`
`  return 'qux';`
`  }`
`}`
 	
`const b = new B();`
	Using the class definitions to the left, what do the following **expressions** evaluate to?
	
	`b.hasOwnProperty('foo')`
	`b.hasOwnProperty('bar')`
	`b.hasOwnProperty('qux')`
	
	`false`
	`true`
	`false`
		Using the class definitions to the left, what do the following **expressions** evaluate to?
		`Obj.getProto` as shorthand for `Object.getPrototypeOf`
		
		`Obj.getProto(b) ===`
		`  B.prototype;`
		
		`Obj.getProto(B.prototype) ===`
		`  A.prototype);`
		
		`true`
		`true`

`const o = {`
`  start: 1`
`};`
 	
`function sumStart(n, m) {`
`  return this.start + n + m;`
`}`
 	
`const res = sumStart.bind(o, 2);`
 	
`console.log(res(3, 4));`
	What is the output of the code on the left?
	
	`6`
		What is the difference between `bind` and `call` / `apply`.
		
		`call and apply execute a function immediately while bind returns a function`

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. Create a form that `POST`S to the path `/user/add`
   a) the form should have 2 text inputs, `first` and a `last` (representing first and last name)
   b) .....as well as a submit button

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

4. Using the form from above, and the following Express route:

```
app.post('/user/add', function(req, res) {
 res.redirect('/success');
});
```

   What will resulting request from the browser and the response from the server look like, assuming that the user filled in "Frederick Formington" as first and last? For **request** headers, just type [Request headers go here] instead of actual headers. Assume that the data form-url-encoded, meaning that name and value pairs are sent in the request as `name1=value1&name2=value2`.

```
Request
=====

POST /user/add HTTP/1.1
[Request headers go here]

name1=value1&name2=value2
```

```
Response
=====

HTTP/1.1 301 Permanent Redirect
Location: /success
```

5. What is the output of this code, that the contents of /tmp/foo.txt is bar and that a client connects with netcat, and the user types in world?

```
const net = require('net');
const fs = require('fs');

function handleRead(err, data) {
    console.log('file data', data);
}

function handleData(sock, greeting, binaryData) {
    console.log('before readFile');
    console.log(greeting, 'received', (binaryData + '').trim());
    fs.readFile('/tmp/foo.txt', 'utf8', handleRead);
    console.log('after readFile');
    sock.end();
}

function handleConnect(sock) {
    console.log('before data');
    sock.on('data', handleData.bind(null, sock, 'hello'));
    console.log('after data');
}

const server = net.createServer(handleConnect);
console.log('before listen');
server.listen(3000);
console.log('after listen');
```

```
before listen
after listen
before data
after data
before readFile
hello received world
after readFile
file data bar
```

6. 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 3000**
   e) assume that the setup code below is already present
   f) finish writing the contents of `app.js` below:

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

// create app
const app = express();

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

// listen
app.listen(3000);
```

7. 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), or from a class (the class creates a prototype or sets a prototype based on the extends keyword).

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

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

   a) There's more than one answer to both of these; here are some potential answers

   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.

   Host – the domain name of the server that the request is sent to

   b) Content-Type – for a response, the internet media type of the content (is it html, an image, etc.)

   Set-Cookie – instructs the client to create a cookie

   Location – the url that a 3xx response will redirect to

9. 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: **there are more than three answers**

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)**
arrow functions	**whatever `this` is in the surrounding context**

10. 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'`

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. Create a site using Express and handlebars templates:

    a) It should **respond to `GETS` on** three URLs: **`/old`** , **`/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.hbs`**
        - all pages on the site should have a header that says NAMES!!! - do this in such a way that the header doesn't have to be written in each individual template

    d) All **existing route handlers** and any **future route handlers** should log out the http method of the request. (That is, you will not have to put this logging functionality explicitly in your routes, but instead, use an express feature that will do this for all routes)
    e) 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 routes below this line

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

app.use((req, res, next) => {
    console.log(req.method);
    next();
});

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

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

`/views/names.hbs`

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

`/views/layouts/layout.hbs`

```
<!doctype html>
<html>
    <head><title>A Site</title></head>
<body>


    <h1>NAMES!!!</h1>
    {{{ body }}


</body>
</html>
```

13. 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.

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. The following code should print out: `[ 'foo!', 'bar!', 'baz!' ]` ...However, there's an error in the implementation!

```
const obj = {
   punctuate: function() {
      // return this.words.map(w => this.s === undefined ? '' : w + this.s);
      // or use bind(this) on the anonymous function below

      return this.words.map(function(w) {
          if (this.s === undefined) {
              return '';
          } else {
              return w + this.s;
          }
      });
  },
  s: '!',
  words: ['foo', 'bar', 'baz']
};
console.log(obj.punctuate());
```

a) What does the code above print out?

`['', '', '']`

b) Fix the call to map (by crossing out / drawing arrows, etc.) so that the function works correctly.

16. **(Relevant only if cookies are covered)** Answer the following questions about cookies:

a) In the context of web development (of course!), what's a cookie?

It's a small piece of data that the server tells the client to store. In browsers, this data can be stored as a text file or within a database. Cookies are used to maintain state between http requests.

b) What are some practical applications of cookies?

Maintaining a session id or remembering a user's settings / preferences on the client side.

17. Describe the user interaction / web page markup that may have resulted in this HTTP request being issued from the browser:

```
GET /foo?bar=baz HTTP/1.1
Host: qux.corge
```

A user entered the url http://qux.corge/foo?bar=baz in the url bar of the browser

or

A user submitted a form that had a method GET, action /foo, and an input element named bar.

18. Write the two functions specified below:

Function 1: 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']`

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

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

    // put the two together in frankenArray
    const frankenArray = [...firstHalf, ...secondHalf]
    // frankenArray = firstHalf.concat(secondHalf);

    return frankenArray;
}
```

Function 2: 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:

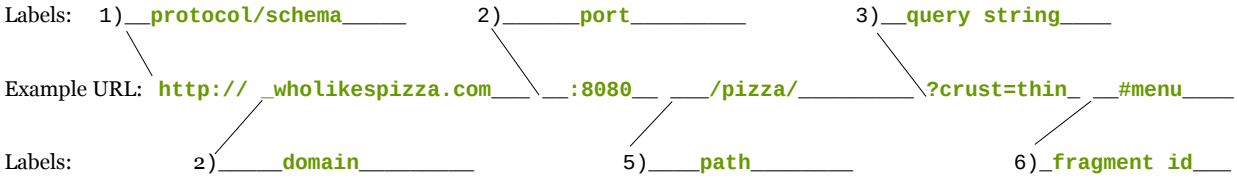
(assuming numbers is an `Array`, like `[2, 4, 6, 8]`)

```
e) numbers.map(myCallback);
f) myMap(numbers, myCallback);
```

Implement `myMap` below:

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

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



20. Write the following function: `repeatCall(fn, n, arg1, arg2, up to argN)`. This function calls the function, fn, n times.

`fn` - the function to be called repeatedly
`n` - the number of times to call function, fn
`arg1, arg2 up to argN` - the arguments to pass to function, fn, when it is called

Example 1:

```
repeatCall(console.log, 4, "Hello!");

// prints out:
// Hello!
// Hello!
// Hello!
// Hello!
```

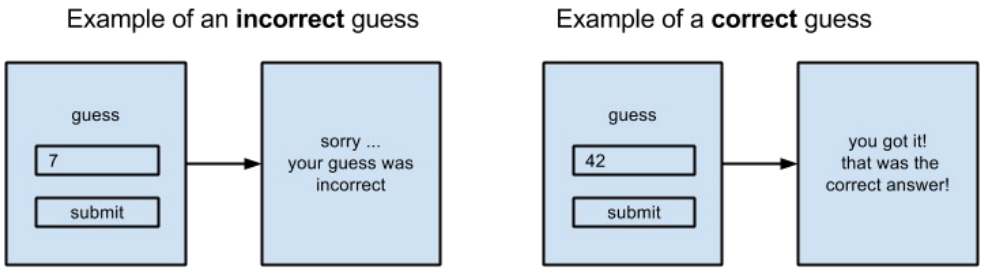
Example 2:

```
repeatCall(console.log, 2, "foo", "bar", "baz", "qux");

// prints out:
// foo bar baz qux
// foo bar baz qux
```

```
function repeatCall(fn, n, ...args) {
    for(var i = 0; i < n; i++) {
        fn(...args);
    }
}

// or the old way, using arguments and apply
function repeatCall(fn, n) {
    var args = Array.prototype.slice.call(arguments, 2);
    for(var i = 0; i < n; i++) {
        fn.apply(null, args);
    }
}
```

21. **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:



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. Assume setup code for express is present.

```
// handlebars setup and request.body

const 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 …
```

22. (Relevant only if databases are covered) 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:

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

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

23. (Relevant only if databases are covered) 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
    ```

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