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` Specify 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.

<table>
<thead>
<tr>
<th>Code</th>
<th>Question 1</th>
<th>Question 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>var vegetable = 'kohlrabi';</code></td>
<td>What is the output of the code on the left?</td>
<td>kohlrabi rambutan broccolini lychee broccolini rambutan</td>
</tr>
<tr>
<td><code>var fruit = 'rambutan';</code></td>
<td></td>
<td>function</td>
</tr>
<tr>
<td><code>var say_food = function() {</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>vegetable = 'broccolini';</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>var fruit = 'lychee';</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(vegetable, fruit);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(vegetable, fruit);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>say_food();</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(vegetable, fruit);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>vegetable = 'broccolini';</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>var fruit = 'lychee';</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>say_food();</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(vegetable, fruit);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>var say_food = function() {</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>vegetable = 'broccolini';</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>var fruit = 'lychee';</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(vegetable, fruit);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(vegetable, fruit);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>say_food();</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(vegetable, fruit);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>function foo() {</code></td>
<td>What is the output of the code on the left?</td>
<td>baz bar</td>
</tr>
<tr>
<td><code>console.log(arguments[1]);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(arguments[0]);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>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?</td>
<td>this</td>
</tr>
<tr>
<td><code>foo('bar', 'baz', 'qux', 'quux');</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>function calculate(num) {</code></td>
<td>What is the output of the code on the left?</td>
<td>15</td>
</tr>
<tr>
<td><code>var magicNumber = 5;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>return function(n) {</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>return n * (num - magicNumber);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>};</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>var calculateTen = calculate(10);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>console.log(calculateTen(3));</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

   ```javascript
   // 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:

<table>
<thead>
<tr>
<th>Invocation</th>
<th>this</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular function call</td>
<td>the global object (in the case of node, the module level global object)</td>
</tr>
<tr>
<td>Method call</td>
<td>the object the method was called on</td>
</tr>
<tr>
<td>call/apply</td>
<td>the this argument passed in to call/apply (you can set explicitly)</td>
</tr>
</tbody>
</table>
9. Using the following code, write out the output and **briefly** explain the reason for the output in the table below.

```javascript
var obj = Object.create(Array.prototype);
obj.foo = 'bar';
console.log(typeof obj.foo); // string
console.log(typeof obj.baz); // undefined
console.log(typeof obj.push); // function
console.log(obj.hasOwnProperty('foo')); // true
console.log(obj.hasOwnProperty('push')); // false
console.log(obj.hasOwnProperty('toString')); // false
```

<table>
<thead>
<tr>
<th>#</th>
<th>Output</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td>obj.foo's value is a string</td>
</tr>
<tr>
<td>2</td>
<td>undefined</td>
<td>obj does not have a property named baz</td>
</tr>
<tr>
<td>3</td>
<td>function</td>
<td>obj's prototype is Array.prototype, which contains the method, push (a function)</td>
</tr>
<tr>
<td>4</td>
<td>true</td>
<td>foo was defined on obj, so it's an 'own property'</td>
</tr>
<tr>
<td>5</td>
<td>false</td>
<td>push comes from obj's prototype, so it's not an 'own property'</td>
</tr>
<tr>
<td>6</td>
<td>false</td>
<td>toString comes from Object.prototype, so it's not an 'own property'</td>
</tr>
</tbody>
</table>

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`.

```javascript
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;
};
```

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 GET 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>function Foo(num) { this.x = num; }</td>
<td>What is the output of this program?</td>
</tr>
<tr>
<td>Foo.prototype.printX = function() { console.log(this.x); }</td>
<td>100 function</td>
</tr>
<tr>
<td>function Bar(num1, num2) {</td>
<td>What would this refer to if the keyword new were omitted?</td>
</tr>
<tr>
<td>Foo.call(this, num1);</td>
<td>The global object</td>
</tr>
<tr>
<td>this.y = num2;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>Bar.prototype = Object.create(Foo.prototype);</td>
<td></td>
</tr>
<tr>
<td>var b = new Bar(100, 200);</td>
<td></td>
</tr>
<tr>
<td>b.printX();</td>
<td></td>
</tr>
<tr>
<td>console.log((typeof Foo))</td>
<td></td>
</tr>
</tbody>
</table>

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:

```
(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)

Example URL: **http://wholikespizza.com** :8080 **/pizza/**?crust=thin #menu

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);
    }
```
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 show 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**

![Diagram of a guess and a wrong guess]

**Example of a correct guess**

![Diagram of a guess and a correct guess]

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.

```javascript
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)

```html
<!-- 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