University of Maryland College Park
Department of Computer Science
CMSC335 Spring 2022
Exam #3 Key

FIRSTNAME LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

Instructions

Grader Use Only

Problem #1 (Miscellaneous)	24	
Problem #2 (Array Methods)	36	
Problem #3 (Custom Type Definition)	50	
Problem #4 (Express)	90	
Total	200	

Problem #1 (Miscellaneous)

1. (3 pts) The following code fragment is part of a JavaScript program.

```
let timeInMilliSeconds = 0;
setTimeout(() => console.log("Hello"), timeInMilliSeconds);
```

When will the "Hello" message be printed?

a. Always immediately (no wait time).
b. It depends on how many tasks are on the Macrotask queue.
c. It will take less than 2 seconds.
d. It will take between 2 and 4 seconds.

Answer: b.

2. (3 pts) In which Event loop queue are promise tasks placed?

a. In the Microtask queue.
b. In the Macrotask queue.
c. In either the Macrotask or the Microtask queue.
d. In the third queue, the GeneralTask queue.

Answer: a.

3. (3 pts) The **fetch** API places tasks:

a. In the Microtask queue.
b. In the Macrotask queue.
c. In either the Macrotask or the Microtask queue.
d. In the third queue, the GeneralTask queue.

Answer: a.

4. (3 pts) We can use Ajax to generate:

a. An HTTP request (only get requests).
b. An HTTP request (only post requests).
c. An HTTP request (they can be get or post requests).
d. None of the above.

Answer: c.

5. (3 pts) When a JavaScript object is frozen using the freeze() method:

a. We cannot modify the value of properties, but we can add properties.
b. We cannot modify the value of properties, and we cannot add/delete properties.
c. We cannot modify the value of properties, we cannot add properties, but we can delete properties.
d. We can modify the object after a particular amount of time has passed.

Answer: b.

6. (3 pts) Is it possible to extend a JavaScript type (e.g., Array)? In other words, is the following possible?

```
class SuperArray extends Array { }
```

a. True
b. False

Answer: a.

7. (3 pts) With the fetch API we can:

a. Only issue get requests.
b. Issue both get and post requests.
c. Only issue post requests.
d. Not issue any get or post requests.

Answer: b.

8. (3 pts) Which types of JavaScript modules were discussed in class? Circle those that apply.

   a. ECMAScript Harmony (ES6)
   b. CommonJS
   c. DeferredES1 modules
   d. None of the above.

Answer: a. and b.

Problem #2 (Array Methods)

A **projects** array keeps track of programming projects for a class. The following is an example of some entries the array could have:

```
const projects = [
    {name: "Btree", language: "Java", lines: 20},
    {name: "LinkedList", language: "C", lines: 10},
    {name: "Viewer", language: "PHP", lines: 50},
];
```

To answer the following questions, you may not use any for/while/do-while loops. The only function you can use is **reduce**. Also, you may only use => functions. Your code should work with different data (not just the entries shown above).

1. (8 pts) Complete the following assignment, so the total number of lines in the array is printed. For example, for the above array the output will be **80**

```
const totalLines = projects.reduce(
console.log(totalLines);
```

Answer:

```
projects.reduce((result, elem) => {
    return result + elem.lines;
}, 0);
```

2. (10 pts) Complete the following assignment so the names of the projects are printed after the string "All Names: " is printed. For example, for the above array the output will be **All Names: BtreeLinkedListViewer**

```
const allProjectsNames = projects.reduce(
console.log(allProjectsNames);
```

Answer:

```
projects.reduce((result, elem) => {
    return result + elem.name;
}, "All Names: ");
```

3. (10 pts) Complete the following assignment so the smallest number of lines associated with any project is printed. For example, for the above array, the output will be 10.

```
const smallestNumberOfLines = projects.reduce(
});
console.log(smallestNumberOfLines);
```

Answer:

```
projects.reduce((result, elem) => {
    return result < elem.lines ? result : elem.lines;
});
```

4. (8 pts) Complete the following assignment so the project with the largest number of lines is printed. For example, for the above array, the output will be **{ name: 'Viewer', language: 'PHP', lines: 50 }**

```
const projectLargetNumberOfLines = projects.reduce(
console.log(projectLargetNumberOfLines);
```

Answer:

```
projects.reduce((result, elem) => {
    return result.lines > elem.lines ? result : elem;
});
```

Problem #3 (Custom Type Definition)

Write a **DataManager JavaScript** class (similar to what you have in Java) according to the specifications below. The class keeps track of a URL and JSON information. The JSON data is downloaded using the **fetch** API. Below, we provided a driver (and output) illustrating some of the class methods you need to implement. For this problem you don't need to add any require() statements.

1. **Instance variables -** Two private instance variables: url and json.
2. **One private static variable named defaultURL -** The value of this variable is **http://localhost/data.json**
3. **Constructor -** It has one parameter (an URL) and initializes the url instance variable with the parameter value. If the parameter value is missing, the **defaultURL** value will be used. Use the coalescing (??) operator during the implementation of this code.
4. **set url method -** It updates the url instance variable with the parameter value.
5. **get url method -** It returns the url instance variable value.
6. **downloadJSON() -** This method downloads data using the fetch API and the url instance variable. The downloaded data will be assigned to the json instance variable. You can assume the provided URL has JSON data. You need to use await and async for the implementation of this method.
7. **printJSON() -** Prints to the console the value of the json instance variable.

Driver:	Output:

```
(async () => {
    const dataManager = new DataManager();
    await dataManager.downloadJSON();
    dataManager.printJSON();
})();
```
	```
	[
	  { cat: 'gato' },
	  { dog: 'perro' },
	  { bird: 'ave' },
	  { Lion: 'Leon' }
	]
	```

Answer:

```
class DataManager {
    static #defaultURL = "http://localhost/data.json";
    #url;
    #json;

    constructor(url) {
        this.#url = url ?? DataManager.#defaultURL;
    }

    set url(url) {
        this.#url = url;
    }

    get url() {
        return this.#url;
    }

    async downloadJSON() {
        const result = await fetch(this.#url);
        this.#json = await result.json();
    }

    printJSON() {
        console.log(this.#json);
    }
}
```

Problem #4 (Express)

Complete the routes/endpoints below. You can assume the code below precedes the routes you will define. A cheat sheet can be found below (do not remove the page that has it).

```
const express = require("express");
const app = express();
const path = require("path");
const portNumber = 5000, httpOKstatusCode = 200;
const bodyParser = require("body-parser");
const httpNotFoundStatusCode = 404;
```

1. **getSchool -** When users type the url **http://localhost:5000/getSchool** the webpage will display "UMCP" using <h2> tags. The content-type must be HTML.

```
app.get("/getSchool
```

Answer:

```
app.get("/getSchool", (request, response) => {
  response.writeHead(httpOKstatusCode, { "Content-type": "text/html" });
  let answer = "<h2>UMCP</h2>";
  response.end(answer);
});
```

2. **getCityInfo -** When users type the url **http://localhost:5000/getCityInfo**, the webpage will display the JSON string representation of an object with the property **city** (with the value "CP") and the property **population** with the value 30000.

```
app.get("/getCityInfo
```

Answer:

```
app.get("/getCityInfo", (request, response) => {
   let obj = {city: "CP", population: 30000};
   response.end(JSON.stringify(obj));
  });
```

3. **getLocation -** When users type the url **http://localhost:5000/getLocation?course=cmsc335**, the webpage will display the message "ESJ" representing the location of the class. For any other course (e.g., http://localhost:5000/getLocation?course=cmsc216) the message "Iribe" will be displayed.

```
app.get("/getLocation
```

Answer:

```
app.get("/getLocation", (request, response) => {
   response.writeHead(httpOKstatusCode, {"Content-type": "text/html"});
   let answer = "ESJ";
   if (request.query.course !== "cmsc335") {
      answer = "Iribe";
   }
   response.end(`Room for the course ${request.query.course} is ${answer}`);
});
```

4. **getParkingInfo -** When users type the url **http://localhost:5000/getParkingInfo/tennis/sat**, the webpage will display the message "green" if tennis/sat is present, "blue" if chess/sun is present, and "silver" for any other values (e.g., /basketball/mon).

```
app.get("/getParkingInfo
```

Answer:

```
app.get("/getParkingInfo/:sport/:day", (request, response) => {
   let parkingLot;
   let sport = request.params.sport;
   let day = request.params.day;

   if (sport === "tennis" && day == "sat") {
      parkingLot = "green";
   } else if (sport === "chess" && day === "sun") {
      parkingLot = "blue";
   } else {
      parkingLot = "silver";
   }
   response.end(`Parking Lot: ${parkingLot}`);
});
```

5. **umdPage -** When users type the url **http://localhost:5000/umdPage**, the webpage will display the webpage associated with the website http://www.umd.edu

```
app.get("/umdPage
```

Answer:

```
app.get("/umdPage", (request, response) => {
   response.redirect("http://www.umd.edu");
});
```